2004

Web-based Circuit Design Simulation Package For Solving Electrical Eng

Shadi Harb
University of Central Florida

WEB-BASED CIRCUIT DESIGN SIMULATION PACKAGE FOR SOLVING ELECTRICAL ENGINEERING CIRCUITS

by

SHADI HARB
B.S. Jordan University of Science & Technology, 2000

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of the Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2004

ABSTRACT

A Web-based circuit design package has been improved and evaluated to provide students with an enhanced and innovative teaching tools package for the electrical circuit design course. The project objectives can be summarized as follows: 1) developing enhanced problem solving skills using a Web-based environment, 2) developing the design skills and sharpening the critical thinking process, 3) developing a generic and comprehensive teaching/learning circuit package as an extention to the Electrical Engineering virtual lab environment, which gives students the capability to practice and experience all the circuit design skills with minimum cost and effort. The project provides the students with an enhanced and powerful graphical computer aided design (CAD) tool by which students can carry out an online simulation of AC and DC designs with the capability to plot simulation results graphically. The proposed prototype is implemented by JAVA, which is used to to implement Web-based applications with different platform support. The project provides students with an enhanced graphical user interface (GUI) by which they can build any electrical circuit using either text or schematic entry format, generate the Netlist, which describes all circuit information (circuit topology, circuit attributes and so on), and simulate the design by parsing the Netlist to CIRML format, which is sent over the network to the remote server. The server will process the CIRML data and run the simulation using PSPICE and eventually send back the simulation results to the client for display.

ACKNOWLEDGMENTS

I sincerely thank my advisor Dr. Issa Batarseh and the other committee members, who supervised my course study and gave me the opportunity to work on the research project towards my master degree. I am grateful for their support and help. Above all, I would like to express my gratitude to my family, colleagues and friends who keep encouraging me during my study period. Without their support, I could never make my achievements.

TABLE OF CONTENTS

LIST OF FIGURES vii

LIST OF TABLES xi

LIST OF ACRONYMS/ABBREVIATIONS xii

CHAPTER ONE: INTRODUCTION 1

1.1 Background 1

1.2 Research Significance 2

1.3 Objectives 3

1.4 Thesis Outlines and Results 3

CHAPTER TWO: TECHNICAL BACKGROUND AND INFORMATION 5

2.1 JAVA Programming Language 5

2.2 PSPICE Circuit Simulation 7

CHAPTER THREE: SYSTEM REQUIRMENTS AND SPECIFICATIONS 10

3.1 System Overview 10

3.2 System Architecture 11

3.2.1 Front-End Client Software Package 11

3.2.2 Background Server Software Package 12

3.3 System Feature Specifications 15

3.3.1 Front-End GUI Interface 15

3.3.2 GUI Manipulation 50

3.3.3 Component Types and Definitions 58

CHAPTER FOUR: SYSTEM DESIGN AND IMPLEMENTATION 70

4.1 System Platform Design 70
4.1.1 JAVA as a Developing Programming Language 70
4.1.2 JBUILDER as a Development System Environment 70
4.2 System Architecture Design 71
4.2.1 Front-End Client Software Package (FECSP) 71
4.2.2 Background Server Software Package (BSSP) 71
4.3 Global Data Structure Improving 73
4.3.1 Problems in the Previous Design and Code 73
4.3.2 Modified Global Data Structure 73
4.4 Module Design 74
4.4.1 Client Applet Design 74
4.4.2 Netlist Manipulation 74
4.4.3 System Main Services 82
4.4.4 System Main Classes Overview 83
CHAPTER FIVE: TESTING AND VERIFICATION 90
5.1 Functional Testing 90
5.2 Functional Feature Tracking Report 90
5.3 Netlist Testing 100
5.4 Netlist Testing Examples 100
5.5 Browser Compatibility Testing 103
CHAPTER SIX: FUTURE WORK AND IMPROVMENTS 104
6.1 GUI Schematic 104
6.1.1 Component Rotation 104

6.1.2 Component Resizing 104
6.1.3 Component Display 105
6.2 Data Structure and Components Library 105
6.2.1 Data Structure 105
6.2.2 Components Libraries 105
6.3 Simulation Analysis 106
6.3.1 Circuit Error Checker 106
6.3.2 Simulation Analysis 106
6.3.3 Circuit Simulation and Evaluation Service 106
6.4 User File Managments System 107
CHAPTER SEVEN: CONCLUSION 108
APPENDIX A: USER MANUAL 109
APPENDIX B: HELP TOPICS 112
APPENDIX C: SOURCE CODE 166
LIST OF REFERENCES 724

LIST OF FIGURES

Figure 2-1: PSPICE Simulation Steps 9

Figure 3-1: Overview for Client/Server Architecture 11

Figure 3-2: High Level System Architecture 14

Figure 3-3: Front-End GUI Interface Overview 16

Figure 3-4: Drawing Area Panel 17

Figure 3-5: Drawing Tools Panel 18

Figure 3-6: File Manager Tools Panel 19

Figure 3-7: Simulation Tools Panel 20

Figure 3-8: Simulation Results Panel 21

Figure 3-9: Graphical Results Panel 21

Figure 3-10: Standard Toolbar 22

Figure 3-11: Simulation Toolbar 22

Figure 3-12 : Components Toolbar 23

Figure 3-13: Canvas Toolbar 23

Figure 3-14: Schematic Pages Toolbar 24

Figure 3-15: Drawing Toolbar 24

Figure 3-16: Visited Elements Toolbar 25

Figure 3-17:GUI Status Bar 25

Figure 3-18: GUI Menu bar 26

Figure 3-19: Single Element Popup Menu 27

Figure 3-20: Multiple Elements Popup Menu 27

Figure 3-21: Connections Popup Menu 28
Figure 3-22: Grid Area Popup Menu 28
Figure 3-23: Resistor Input Window 29
Figure 3-24: DC Voltage Source Input Window 30
Figure 3-25: AC Voltage Source Input Window 31
Figure 3-26: VCVS Input Window 32
Figure 3-27: Current Marker Input Window 32
Figure 3-28: Canvas Properties Dialog 33
Figure 3-29: Display Properties Dialog 34
Figure 3-30: Insert Text Dialog 35
Figure 3-31: Canvas Background Color Dialog 36
Figure 3-32: Save File Chooser Dialog 37
Figure 3-33: Server File Manager Dialog 38
Figure 3-34: Simulation Settings Dialog 39
Figure 3-35: Netlist Display Dialog 40
Figure 3-36: Netlist Editor Dialog 41
Figure 3-37: CIRML Display Dialog 42
Figure 3-38: Output Result Display Dialog 43
Figure 3-39: Online Help Topics Window 44
Figure 3-40: Welcome Message 45
Figure 3-41: Startup Tip Message 46
Figure 3-42: Save Schematic Page Question Message 46
Figure 3-43: Floating Node Alert Message 47

Figure 3-44: Ground Does Not Exist Alert Message 47
Figure 3-45: Dependent Controller Not Found Alert Message 48
Figure 3-46: Schematic Does Not Exist Error Message 48
Figure 3-47: Only Connection Lines Exist Error Message 49
Figure 3-48: Short Circuit Error Message 49
Figure 3-49: Passive Elements Panel 58
Figure 3-50: Resistor Symbol 59
Figure 3-51: Capacitor Symbol 59
Figure 3-52: Inductor Symbol 59
Figure 3-53: Transformers Panel 60
Figure 3-54: Positive/Positive Transformer Symbol 60
Figure 3-55: Positive/Negative Transformer Symbol 61
Figure 3-56: Negative/Positive Transformer Symbol 61
Figure 3-57: Negative/Negative Transformer Symbol 62
Figure 3-58: Independent Sources Group Panel 62
Figure 3-59: DC Voltage Source Symbol 63
Figure 3-60: DC Current Source Symbol 63
Figure 3-61: AC Voltage Source Symbol 63
Figure 3-62: AC Current Source Symbol 64
Figure 3-63: Dependent Sources Panel 64
Figure 3-64: Voltage Controlled Voltage Source Symbol 65
Figure 3-65: Voltage Controlled Current Source Symbol 65
Figure 3-66: Current Controlled Current Source Symbol 66

Figure 3-67: Current Controlled Current Source Symbol 66
Figure 3-68: Connections Panel 67
Figure 3-69: Connection Line Symbol 67
Figure 3-70: Connection Node Symbol 67
Figure 3-71: Ground Symbol 68
Figure 3-72: Voltage Marker Symbol 68
Figure 3-73: Differential Voltage Marker Symbol 69
Figure 3-74: Current Marker Symbol 69
Figure 4-1: Detailed System Architecture Overview 72
Figure 4-2: Sample Schematic for Netlist Generation Steps 77
Figure 4-3: Connection Line Optimization, Step 1 77
Figure 4-4: Connection Line Optimization, Step 2 78
Figure 4-5: Connection Line Optimization, Final Step 79
Figure 4-6: Net Name Assignment Step 80
Figure 4-7: An Overview of Hierarchal Structure of all System Classes 87
Figure 5-1: Example 1 Schematic 101
Figure 5-2: Example 1 Netlist Result 102
Figure 5-3: Example 2 Schematic 102
Figure 5-4: Example 2 Netlist Result 103

LIST OF TABLES

Table 1: GUI Test Plan 91

Table 2: Schematic Capture Test Plan 91

Table 3: File Management Service Test Plan 94

Table 4: Simulation Service Test Plan 96

LIST OF ACRONYMS/ABBREVIATIONS

a) WCS	Web-based Circuit Solver
b) FECSP	Front End Client Software Package
c) BSSP	Background Server Software Package
d) CKTEC	Circuit Error Checker
e) CSAES	Circuit Simulation and Evaluation Service
f) CSS	Client Simulation Service
g) SSS	Server Simulation Service
h) CFMS	Client File Management Service
i) SFMS	Server File Management Service
j) CIRML	Circuit Markup Language (.CIR format)
k) R	Resistor
l) C	Capacitor
m) I	Inductor
n) DCVS	DC Voltage Source
o) DCCS	DC Current Source
p) ACVS	AC Voltage Source
q) ACCS	AC Current Source
r) VCVS	Voltage Controlled Voltage Source
s) VCCS	Voltage Controlled Current Source
t) CCVS	Current Controlled Voltage Source

u) CCCS Current Controlled Current Source

v) GUI Graphical User Interface

w) TPI Third Party Interface

CHAPTER ONE: INTRODUCTION

1.1 Background

Due to the vast and rapid advances in information technology, the Internet - with its wide availability and its low cost - has been utilized as the ideal medium for education to provide students with Web-based platforms. Distance learning has the ability to develop quickly and to provide students with the skills and knowledge over time and space. Although E-learning tools are useful in many areas, introducing these tools for engineering learning is a daunting challenge, since engineering education can be fulfilled only by video tapes and web-based discussion groups. It needs more interactive labs and innovative online packages that help students to carry out creative designs based on critical thinking. Engineering learning is an accumulative process and many fundamental undergraduate courses need advanced skills and tools of teaching enable students to grasp the basic concepts of these courses. As a result, there is a great demand to develop and improve traditional online multimedia tools to accommodate these changes and support the undergraduate engineering students with effective and innovative educational methodologies.

Over the last few years, many research efforts were focused on developing multimedia and Web-based educational material for circuit design, and most of these efforts focused on developing Graphical User Interface (GUI) for “display” with limited interactivity. Based on our research, none of these Web-based tools provided the user with generalized problem solving design packages, and most of them only addressed specific and predefined design problems.

Our ongoing research, which is conducted at University of Central Florida, aims to develop an online interactive Web-based simulation package to allow engineering students to build, develop and simulate from the “fly” any circuit design problem using interactive graphical user interface (GUI). These innovative tools will help the students to better understand abstract engineering principles. Moreover, this Web-based material is accessible everywhere and anytime with minimal updating and maintenance costs. With this Web-based tool, students can build any circuit using either the schematic or the textual entry. For example, to draw a circuit schematically, students can drop and drag components from the elements panel, make connections, rotate components and so on. After that, students can simulate the design. The client application of this package can convert the schematic design to a Netlist and send it to the server with the proper CIRML format. The server will carry out the simulation and return the results back to the user as a textual result (ASICII format) with additional capability to plot the results graphically.

1.2 Research Significance

There is no doubt that the availability of Internet educational packages enhance the tools and skills of teaching by providing an extensive number of examples and problems with minimum cost and effort, which give students the ability to master their courses principles. Moreover, our ongoing project will eventually enhance the E-learning teaching methodologies by providing students with an innovative and interactive teaching/learning tools for several undergraduate engineering circuit courses. We believe that with this visualized Web-based package and its wide accessibility, students will be motivated to challenge themselves with more

design problems and to experience critical thinking and the problem solving skills. With this self-evaluation tool, students can evaluate and monitor their study progress.

1.3 Objectives

Our ongoing project is aimed at building online material with the following objectives:

i) Improving the functionality of an existing GUI by building a new database data structure from scratch, using more advanced java swing classes, removing redundant codes and building new modules and functions.

ii) Providing the user with a fully capability graphical user interface that is easy to use and efficient to access remotely.

iii) The simulation results will be shown to the user as textual results (ASICII format) with the capability to plot the results graphically.

As a result, the developed GUI applet is an efficient operation and is fast at loading and running JAVA classes.

1.4 Thesis Outlines and Results

Introduction of the thesis provided a background, research significance and objectives in Chapter 1. Technical background and information about JAVA programming and the PSPICE simulation package are elaborated upon in Chapter 2, while system requirements and specifications are set in Chapter 3. System design and implementation are described in Chapter 4, and testing and verification strategies and methodologies are provided for the software acceptance testing of the whole package in Chapter 5. Lastly, ideas and notions for future

improvements and developments are presented in Chapter 6. A user manual is supplemented in APPENDIX A to show the user instructions on how to use the online material package. Eventually the front end applet is fully implemented and developed with more enhanced features and functionalities. The overall system and its documentation are hosted at the following address: http://batarseh.cecs.ucf.edu/cirsim/index.htm.

CHAPTER TWO: TECHNICAL BACKGROUND AND INFORMATION

A literature review about the programming language used in the project is presented in this chapter. Information about the PSPICE simulation and analysis package is presented as well.

2.1 JAVA Programming Language

JAVA is a high level third generation programming language with the same level of the other popular programming languages such as FORTRAN and C/C++. It is a fully object-oriented, distributed, robust, secure, architectural, neutral, portable, interpreted, high performance and multithreading programming language system. JAVA has provided new technologies such as applets and servlets, which are used to build interactive Web-based applications. Applets can be used to develop fully interactive Web-based client applications, while servlets manipulate the server services of computations and connectivity. The main characteristics in JAVA can be described as follows:

i) Simplicity and Manageability

JAVA is a high level, user-friendly language and is an easy tool to write free bug codes. With JAVA, there is no need for header files, pointers, structures, unions, operator overloading, virtual classes and so on.

ii) Fully Object-Oriented

Object oriented methodology in JAVA has proven to produce better and more efficient code writing, testing and maintaining. It is very similar to that of C++ except for multiple inheritance, which is manipulated in JAVA differently using an interface structure that gives less confusion and more efficient code writing for programmers and maintainers.

iii) Portability

JAVA produces “byte code” instructions rather than dependent native code. The “byte code” is interpreted by a third party interface called “Java Virtual Machine,” which in turn converts it to the native one. Also, the dependent architecture codes are eliminated in the JAVA compiler.

iv) Security

JAVA is designed to allow secure execution of codes within the network even when using malicious pieces of codes. It has a robust exception handling mechanisms to deal with expected and unexpected errors. It has different levels of privileges such as system and user privileges.

v) Dynamic language

JAVA has an adaptation feature to add freely more functions and instances in the libraries, and also finding out run-time type information is direct.

vi) Multi-Threading

JAVA is inherently multithreaded, and any single java program can have different executed threads running independently and simultaneously. With this feature, JAVA can accommodate many responses from the user and can avoid exploiting the whole CPU time.

vii) Garbage Collection

Allocating and reallocating memory no longer needs to be done by a programmer, as JAVA allocates memory by the stack and the heap mechanism, and the garbage collector reclaims the allocated memory as needed.

2.2 PSPICE Circuit Simulation

PSPICE is an electrical and electronic simulation and analysis package that is widely used in academia and industry. It was developed at University of California at Berkley, and it was used initially by mainframes. Later, it was extended to be operated by PCs, Macintosh and Workstations. PSPICE can do simulation and analysis for many electrical and electronic circuit designs using different behavior analysis such as DC, AC, transient, sweep and so on. Also, it uses nonlinear techniques to handle the active components such as transistors and diodes. PSPICE has a schematic capture entry, which can directly access the components and the symbol libraries. The schematic drawings can be saved as a ".sch" file extension. PSPICE generates a ".net" file to describe the topology of the circuit after performing the electrical circuits rules check (ERC) and a ".cir" symbolic description file, which issues simulation instructions for PSPICE. The simulation results can be viewed either textually or graphically depending on the

performed analysis. Graphical results can be displayed by invoking the Probe software after simulation. Because of the modularity of PSPICE, the ".cir" input file can be generated from a third party interface and then interpreted separately by the simulator. Since PSPICE doesn't necessarily need direct user interaction to execute the simulation, any third party application can automate the simulation process. In our project, the third party application is called "Server Application," which accepts the ".cir" format and executes the simulation. Once the simulation is done and the output files are generated, it will return the results to the client for display either textually or graphically, depending on the performed analysis. The simulation steps used by PSPICE to do the analysis on analog circuits are presented in Figure 2-1.

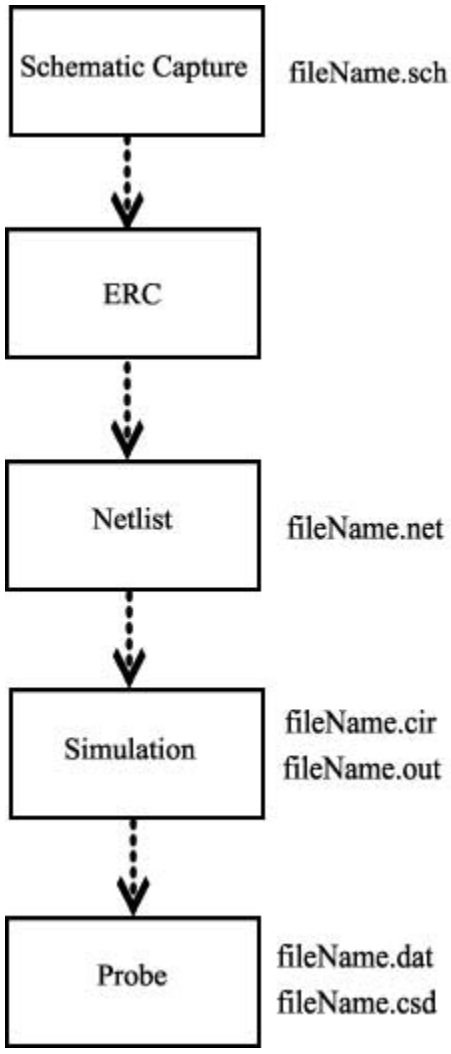


Figure 2-1: PSPICE Simulation Steps

CHAPTER THREE: SYSTEM REQUIRMENTS AND SPECIFICATIONS

3.1 System Overview

The online software package is a Web-based client/server system architecture as shown in Figure 3-1. The client side provides the user with an interactive and innovative graphical user interface (GUI), which runs inside the browser to build and design any circuit problem. The user can draw the circuit schematically with drag/drop components, make connections, edit and rotate components and so on. The Client Application program is attached to the GUI, which makes it a more powerful and interactive interface. It has two main run-time services, the simulation service and the file management service. The first service has another counterpart at the server side. It is responsible for generating the Netlist file and parsing it to the proper ".cir" format, which is accepted by the PSPICE simulator. The second service provides the user with all the file management tools needed to access either the local schematic files or the remote server ones.

The client communicates to the server through the network. Meanwhile, the server is running the Server Application program, which has two main services: the simulation service and the file management service. The first service cooperates side by side with the simulation service at the client side. It does the actual simulation by getting help from a third party application such as PSPICE, which provides the system with the actual circuit analysis and results. After finishing the simulation, the evaluation results will be fed back to the client for display. The second service is designated to manage the files on the server with cooperation from the client's side of the file management service. This service provides the user with many file management actions such as "open," "save," "delete," "mkDir" and "rmDir".

Figure 3-1: Overview for Client/Server Architecture

3.2 System Architecture

The online software material has two main software packages, which are mutually interacted as client/server application: the Front-end Client Software Package and the Background Server Software Package. An overview of the high-level architecture of the software package is shown in Figure 3-2 later in this Chapter.

3.2.1 Front-End Client Software Package

The front-end client software package has two main applications: the front-end GUI and the client application.

i) The Front-End GUI

The front-end GUI provides the user with all the capabilities and features to build a circuit such as drag/drop, edit, rotate components, make connections and so on.

ii) The Client Application

The client application runs two main services: the Circuit Simulation and Evaluation Service and the File Management Service.

a) The Circuit Simulation and Evaluation Service

The simulation service can save and restore the binary data of the schematic at the local memory and sent it over the network to the remote server to run the simulation.

b) The File Management Service

The file management service manipulates all the file management tools either to access the schematic files locally at the client or remotely at the server.

3.2.2 Background Server Software Package

The background server software package is running at the background of the whole online package. It consists of two main applications: the server application and the third party simulation package.

i) The Server Application

The server application consists of two main services: the Circuit Simulation and Evaluation Service and the File Management Service.

a) The Circuit Simulation and Evaluation Service

The simulation service consists of two main parts, the Netlist parser and the third party interface.

i The Netlist PARSER

The Netlist parser translates the Netlist file and generates the CIRML format.

ii The Third Party Interface

The third party interface cooperates side by side with the client simulation service. It runs the actual simulation process by invoking a third party simulation program such as PSPICE. When the simulation is finished, the results are returned back to the client for display.

b) The File Management Service

The file management service deals with all the file management actions on the schematic files for both client and server.

ii) The Third Party Simulation Package

The third party package (PSPICE), which resides on the server, does the actual simulation and generates the simulation output files.

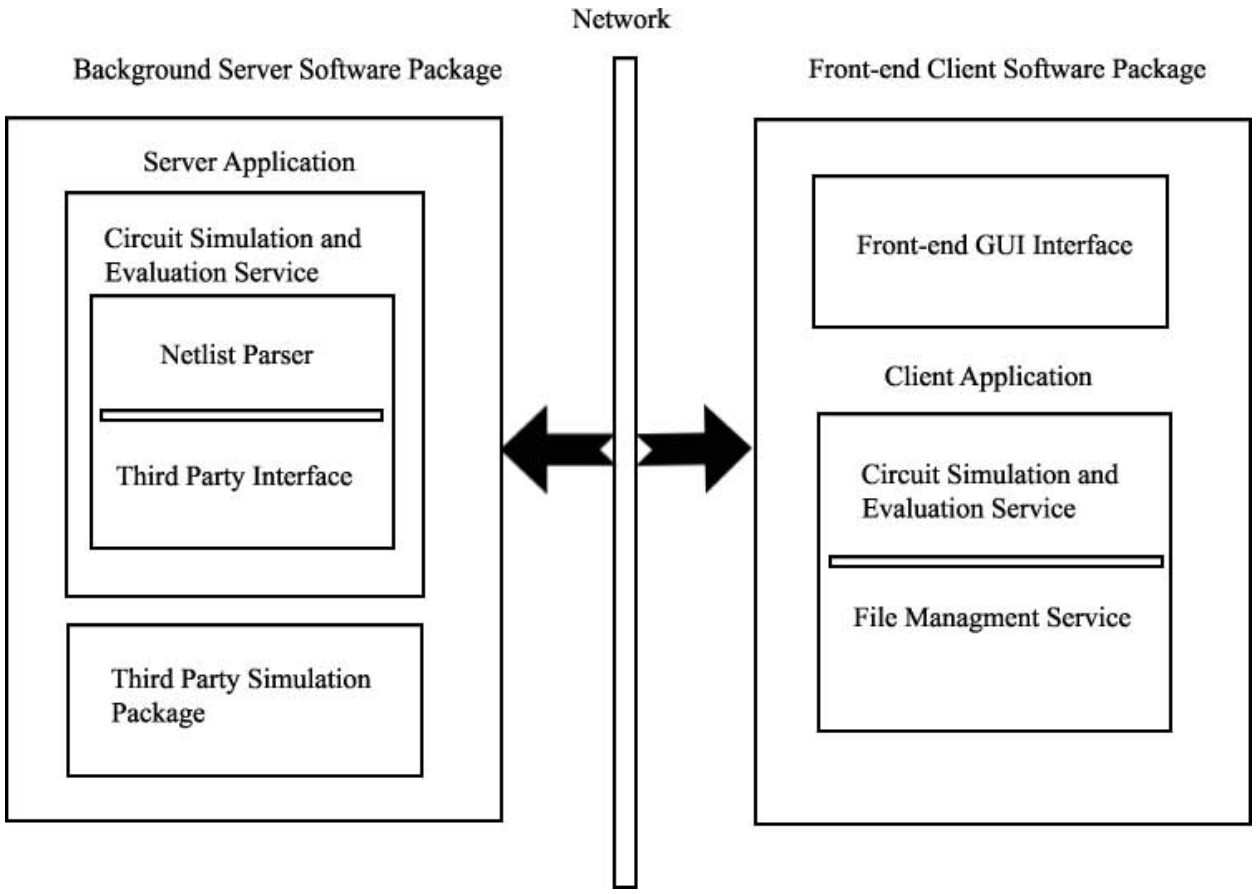


Figure 3-2: High Level System Architecture

3.3 System Feature Specifications

3.3.1 Front-End GUI Interface

The interface provides the user with the circuit component symbols to build basic circuit designs problems. It contains graphics and images such as panels, toolbars and statusbars. The GUI overview is shown in Figure 3-3.

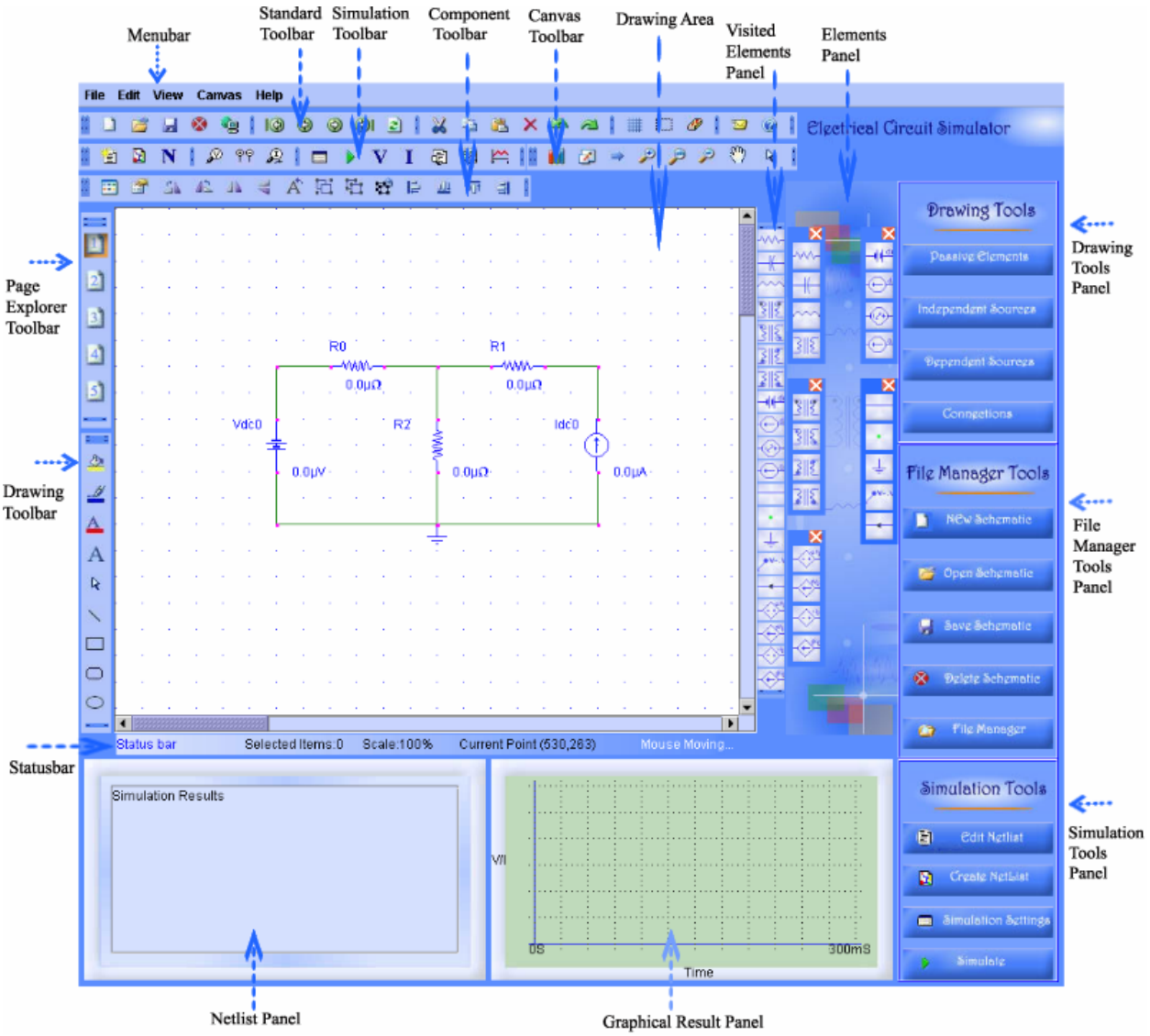


Figure 3-3: Front-End GUI Interface Overview

i) GUI Panels

The GUI interface has the following main panels: Drawing Area Panel, Drawing Tools Panel, Simulation Tools Panel, File Management Tools Panel, Netlist Panel and Graphical Result Panel. A detailed description of all Panels actions is shown in APPENDIX B: Help Topics.

a) Drawing Area Panel

The gird area panel is located at the left side of the GUI. It gives the user the ability to construct any circuit schematically such as editing a component, changing its attributes, moving it, rotating it and so on. Figure 3-4 shown the drawing area panel.

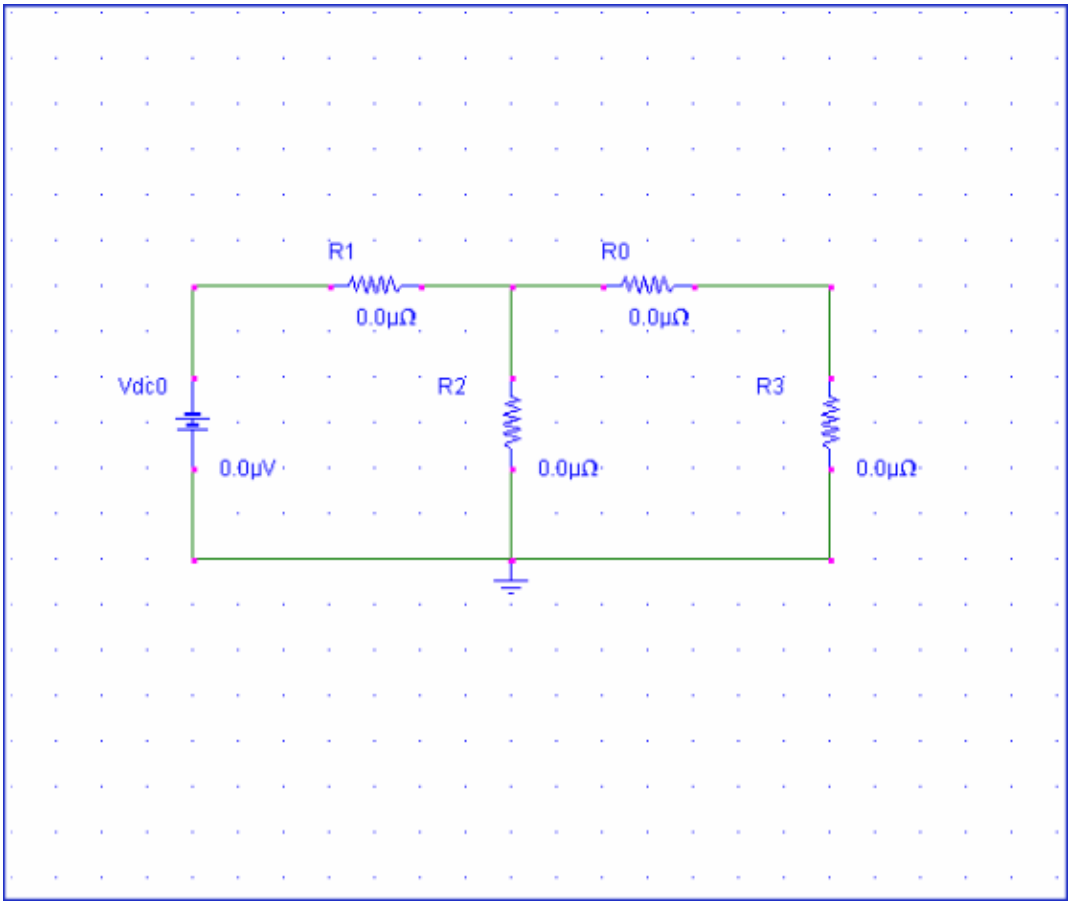


Figure 3-4: Drawing Area Panel

b) Drawing Tools Panel

The drawing tools panel has four action buttons, which are designated to display the circuit components groups: the passive components group, the independent sources group, the dependent sources group and the connections group. Figure 3-5 shows the drawing tools panel.

Figure 3-5: Drawing Tools Panel

c) File Management Tools Panel

The file management tools panel has five action buttons, which are designated to manage the schematic files either locally at the client or remotely at the server. The first button is “New Schematic Page,” which creates new schematic pages. Five pages are the maximum number for each workplace. The second button is “Open Schematic Page,” which loads the local schematic file and displays it on the drawing area panel. The third button is “Save Schematic Page,” which saves the current schematic page at the local client as “.sch” file. The forth button is “Delete

Schematic Page," which deletes the current schematic page from the workplace. The final button is "File Manager," which displays the file management window that has different action buttons to deal with the schematic files at the server. Figure 3-6 shows the file manager tools panel.

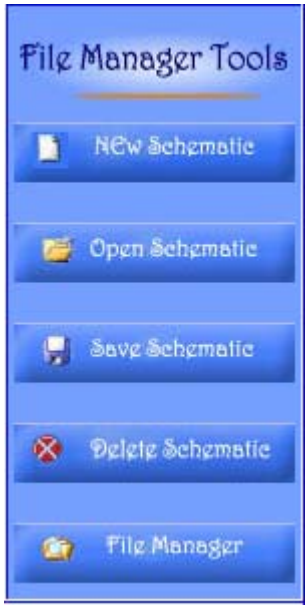


Figure 3-6: File Manager Tools Panel

d) Simulation Tools Panel

The simulation tools panel has four action buttons. The first is "Edit Netlist," which displays a Netlist text editor to write the circuit Netlist as text entry. The second button is "Create Netlist," which generates the Netlist for the current schematic page. The third button is "Simulation Settings," which shows the simulation settings window to edit the simulation parameters. The final button is "Simulate," which invokes the simulation service to run the simulation at the server and return back the results to the client for display. Figure 3-7 shows the simulation tools panel.

Figure 3-7: Simulation Tools Panel

e) Simulation Results Panel

The simulation results panel displays all the text outputs generated from the simulation service such as Netlist output, CIRML output and the Simulation Result output. Figure 3-8 shows the simulation results panel.

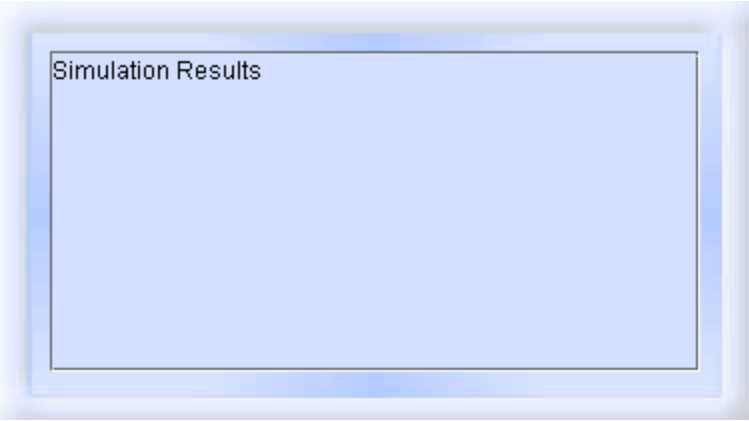


Figure 3-8: Simulation Results Panel

f) Graphical Result Panel

The graphical result panel displays the graphical simulation results, which are invoked by the probe command. Figure 3-9 shows the graphical result panel.

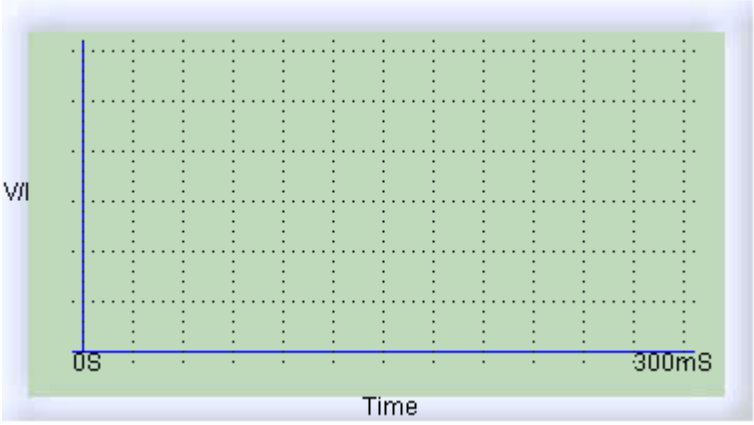


Figure 3-9: Graphical Results Panel

ii) GUI Toolbars

The GUI has the following toolbars: Standard Toolbar, Simulation Toolbar, Schematic Pages Toolbar, Drawing Toolbar and Visited Elements Toolbar as described in the following sections. A detailed description of all toolbars actions are shown in APPENDIX B: Help Topics.

a) Standard Toolbar

The standard toolbar has the following action buttons: “New Page,” “Open Page,” “Save Page,” “Close Page,” “Remote File Manager,” “First Page,” “Previous Page,” “Next Page,” “Last Page,” “Refresh,” “Cut,” “Copy,” “Paste,” “Delete,” “Undo,” “Redo,” “Show/Hide Grid,” “Select All,” “Clear All,” “Email us” and “Help.” Figure 3-10 shows the standard toolbar.

Figure 3-10: Standard Toolbar

b) Simulation Toolbar

The simulation toolbar has the following action buttons: “Edit Netlist,” “Create Netlist,” “Show/Hide Nodes,” “Voltage Marker,” “Voltage Differential Marker,” “Node Marker,” “Simulation Settings,” ”Run,” ”Show Voltages,” “Show Currents,” “Show Output Result,” “Display CIRML” and “Draw Result.” Figure 3-11 shows the simulation toolbar.

Figure 3-11: Simulation Toolbar

c) Components Toolbar

The component toolbar has the following several action buttons: “View Properties,” “Edit Properties,” “Rotate Right,” “Rotate Left,” “Flip Vertical,” “Flip Horizontal,” “Rotate Text,” “Group,” “Ungroup,” “Regroup,” “Align Left” and “Align Bottom,” “Align Top” and “Align Right.” Figure 3-12 shows the component toolbar.

Figure 3-12 : Components Toolbar

d) Canvas Toolbar

The canvas toolbar has the following several action buttons: “Canvas Colors,” “Canvas Size,” “Go to,” “Zoom In,” “Fit Zoom,” “Zoom Out,” “Scroll Hand” and “Default Cursor.” Figure 3-13 shows the canvas toolbar.

Figure 3-13: Canvas Toolbar

e) Schematic Pages Toolbar

The schematic pages toolbar can explore among five schematic pages at a time. Figure 3-14 shows the schematic pages toolbar.

Figure 3-14: Schematic Pages Toolbar

f) Drawing Toolbar

The drawing toolbar contains the following action buttons: “Fill Color,” “Line Color,” “Text Color,” “Edit Text,” “Default Cursor,” “Draw Line,” “Draw Rectangle,” “Draw Round Rectangle” and “Draw Oval.” Figure 3-15 shows the drawing toolbar.

Figure 3-15: Drawing Toolbar

g) Visited Elements Toolbar

The visited elements panel shows the visited circuit components action buttons in the current schematic page. Figure 3-16 shows the visited elements toolbar.

Figure 3-16: Visited Elements Toolbar

iii) GUI Statusbar

The GUI statusbar is located at the bottom of the drawing area panel, which displays the current cursor position, number of active components, the drawing scale and the mouse events on the grid area. Figure 3-17 shows the GUI status bar.

Figure 3-17:GUI Status Bar

iv) GUI Menubar

The GUI Menubar is located at the top of the GUI, it has several menus such as “File,” “Edit,” “View” and “Help.” Each menu has several GUI actions. Figure 3-18 shows the GUI menu bar.

Figure 3-18: GUI Menu bar

v) GUI Popup Menus

Popup menus can be displayed by selecting either a component or a group of components and right-clicking on the grid area. Popup menus contain several actions. Some of these actions are designated for a single component, and some of them are designated for a group of components. The last type of actions can be applied on both kinds of actions. The main GUI interface has four types of popup menus: the Single Element Popup Menu, the Multiple Elements Popup Menu, the Connection Line Popup Menu and the Grid Area Popup Menu. A detailed description of all popup menus actions are shown in APPENDIX B: Help Topics.

a) Single Element Popup Menu

The single element popup menu is designated to a single element. It holds the following actions: View (“Display Properties”, “Go To”), Shape(“Rotate Right”, “Rotate Left”, “Flip Vertical”, “Flip Horizontal”), “Edit Properties”, “Copy”, “Cut” and “Delete” as shown in Figure 3-19.

Figure 3-19: Single Element Popup Menu

b) Multiple Elements Popup Menu

The multiple elements popup menu is designated to a group of elements, it holds the following actions: View (“Display Properties”, “Go To”), Shape(“Rotate Right”, “Rotate Left”, “Flip Vertical”, “Flip Horizontal”, “Group”, “Ungroup”, “Regroup”, “Align Top”, ”Align Bottom”, “Align Right”, “Align Left), “Copy”, “Cut” and “Delete” as shown in Figure 3-20.

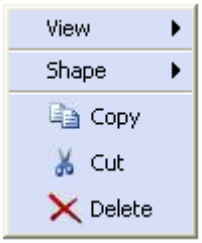


Figure 3-20: Multiple Elements Popup Menu

c) Connections Popup Menu

The connections popup menu is designated for circuit connections such as a connection line. It consists of the following three actions: “Cut,” “Copy” and “Delete,” as show in Figure 3-21.

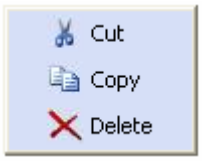


Figure 3-21: Connections Popup Menu

d) Grid Area Popup Menu

The grid area popup menu is designated for the drawing area. it consists of the following actions: “Clear All,” “Zoom In,” “Zoom Out,” “Go to…,” “Undo,” and “Paste” as shown in Figure 3-22.

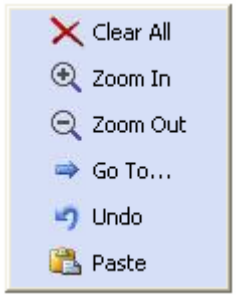


Figure 3-22: Grid Area Popup Menu

vi) GUI Dialogs and Popup Messages

a) Components Input Windows

The user interface has 21 component input windows. They are designed to edit the component attributes. They can be displayed by two ways: First, select the component and double-click on it, and second, right-click and choose the properties action from the popup menu.

Each window displays the value of the component and different editable parameters. The user input windows can be classified to five groups depending on the type of the component: Passive Elements Input Window, Independent DC Sources Input Window, Independent AC Sources Input Window, Dependent Sources Input Window and Current Marker Input Window.

i Passive Elements Input Window

The passive elements user input window is designed to edit the passive component (R, C, I) attributes. The window contains three fields, which represent the component attributes that the user can change. The label field represents the component name, the value and the unit fields, which represent the component value and unit respectively depending on the component type. Figure 3-23 represents an example of a passive element (Resistor) input window.

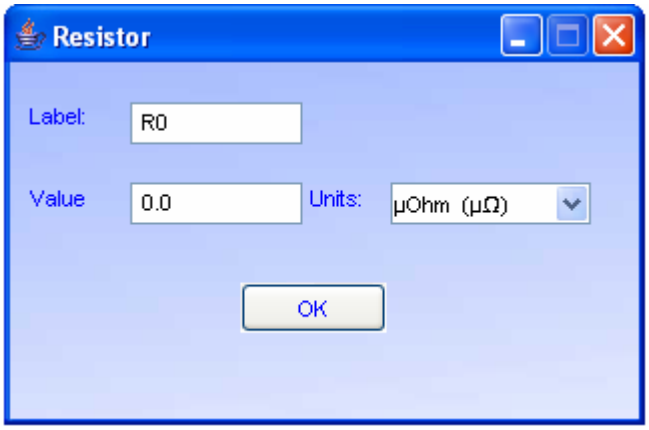


Figure 3-23: Resistor Input Window

ii Independent DC Source Input Window

The independence DC source input window is designed to edit the independent DC source (DCVS, ACVS) attributes. The window contains three fields: the label field, which represents the component name, the value and the unit fields, which represent the component value and unit respectively depending on the component type. Figure 3-24 represents an example of an independent DC source (DC Voltage Source) input window.

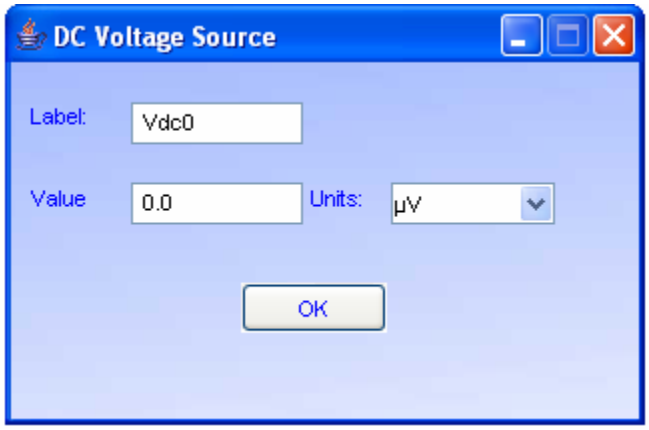


Figure 3-24: DC Voltage Source Input Window

iii Independent AC Source Input Window

The independent AC source input window is designed to edit the independent AC source (ACVS, ACCS) attributes. The window contains nine fields: the label, the peak, the frequency, the phase, the peak unit, the frequency unit, the phase unit and the waveforms (sin(wt)/cos(wt)), which represent the editable component attributes. Figure 3-25 represents an example of an independent AC source (ACVS) input window.

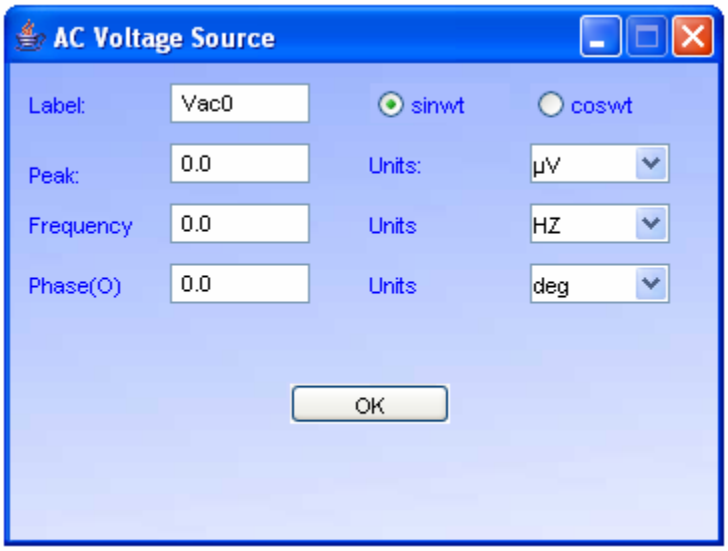


Figure 3-25: AC Voltage Source Input Window

iv Dependent Source Input Window

The dependence source input window is designed to edit the dependent source (VCVS, VCCS, CCVS and CCCS) attributes. The window contains four fields: the label, the control variable, the gain and the gain units, which represent the four component editable attributes. Figure 3-26 represents an example of a dependent source (VCVS) input window.

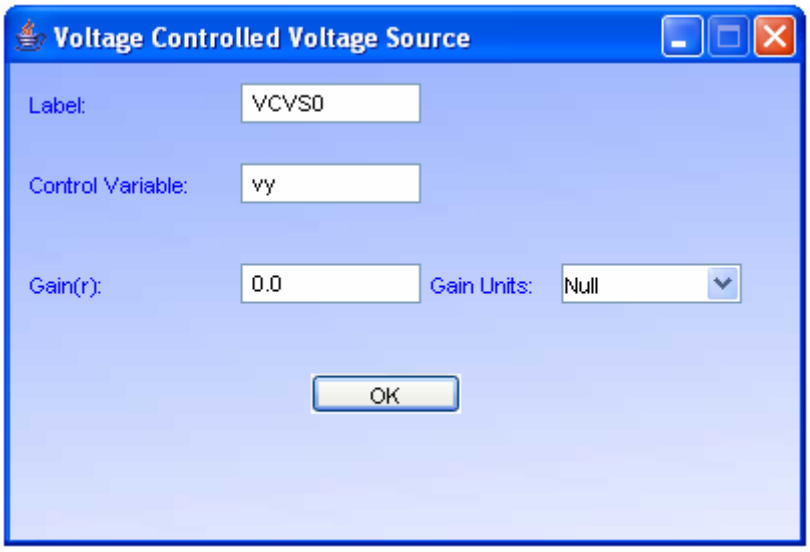


Figure 3-26: VCVS Input Window

v Current Marker Input Window

The current marker input window is designed to edit the current marker label attribute. Figure 3-27 represents the current marker input window.

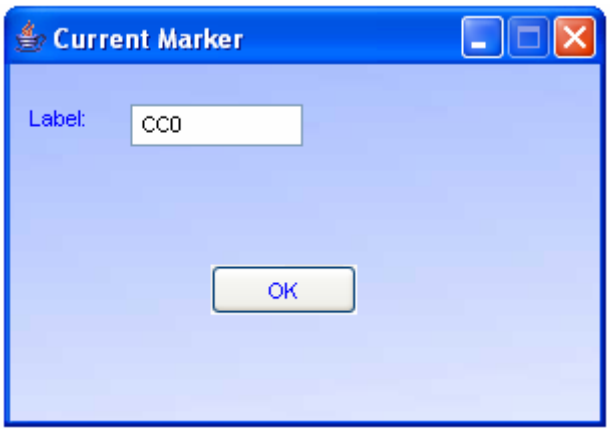


Figure 3-27: Current Marker Input Window

b) GUI Dialogs

i Canvas Properties Dialog

The canvas properties dialog is an input window by which the user can resize the drawing area and change the background and foreground color of the grid panel. Figure 3-28 represents the canvas properties dialog. The dialog will be displayed when “Canvas Colors” and “Canvas Size” actions are chosen from the canvas toolbar.

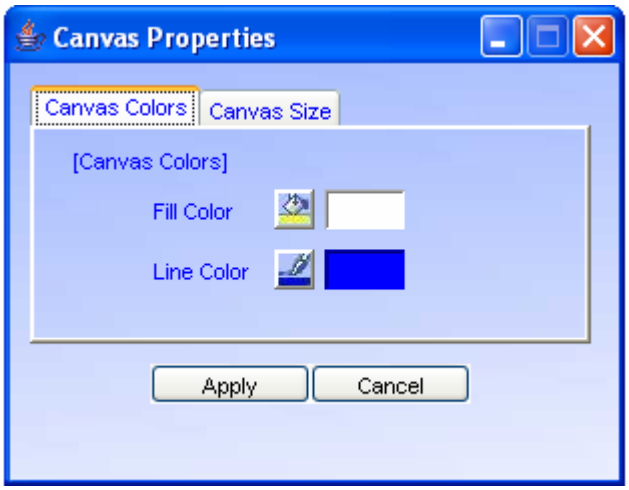


Figure 3-28: Canvas Properties Dialog

ii Display Properties Dialog

The display properties dialog is an input window by which the user can change the component view on the grid such as the component rotation, the name and value display format, and the background and foreground colors. The dialog will be displayed when “Display

Properties" action is chosen from either the component toolbar or from the popup menu. Figure 3-29 represents the canvas properties dialog.

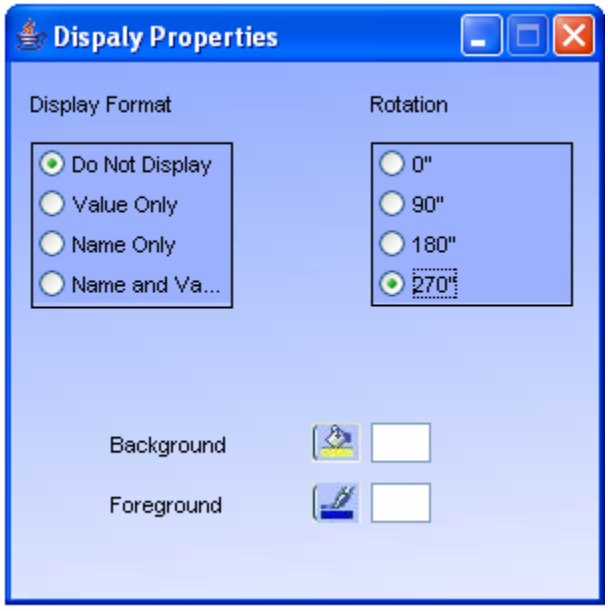


Figure 3-29: Display Properties Dialog

iii Insert Text Dialog

The insert text dialog is an input window by which the user can insert text on the drawing area. The dialog will be displayed when the "Insert Text" action is chosen from the drawing toolbar. Figure 3-30 represents the insert text dialog.

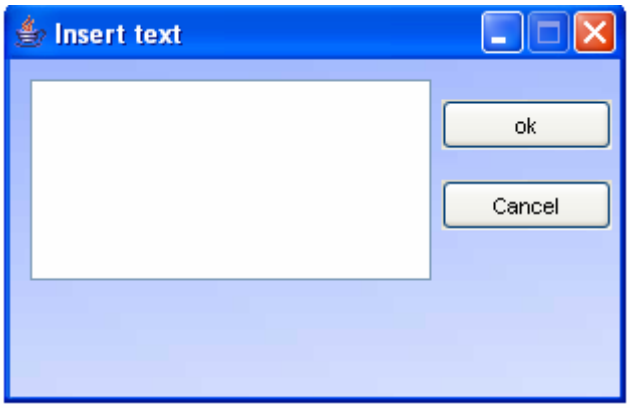


Figure 3-30: Insert Text Dialog

iv Color Chooser Dialog

The color chooser dialog is a user input window by which the user can change the color by choosing from the color panel. The grid canvas has four color choosers: The Canvas Color Chooser to change the background and foreground of the canvas, the Fill Color Chooser to change the fill color of the drawing shapes, the Line Color Chooser to change the line color of the drawing shapes and the Text Color Chooser to change the text and label colors on the canvas. The dialog will be displayed when “Fill Color,” “Line Color” and “Text Color” actions are chosen. Figure 3-31 represents the Canvas Background Color Chooser.

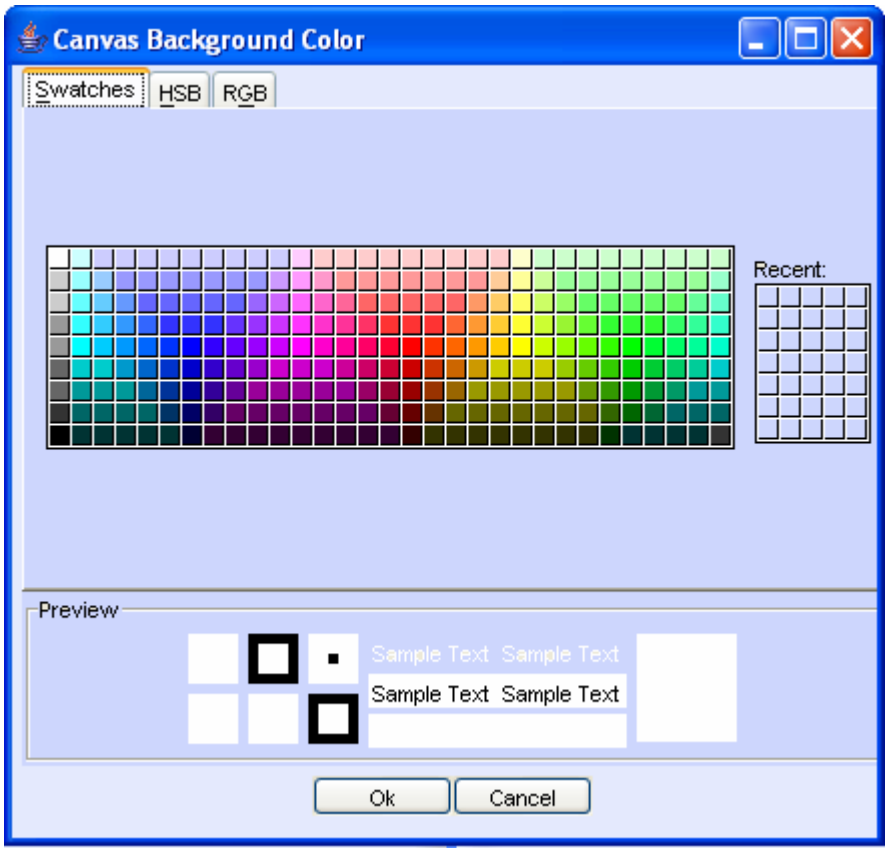


Figure 3-31: Canvas Background Color Dialog

v File Chooser Dialog

The file chooser dialog is an input window by which the user can explore the current directory and select the file name to save and load the schematic data. The GUI has two file chooser dialogs: the Open File Chooser Dialog and the Save File Chooser Dialog. The dialog will be displayed when “Save” and “Open” actions are chosen from either the standard toolbar or the file manager tools panel. Figure 3-32 represents the save file chooser dialog.

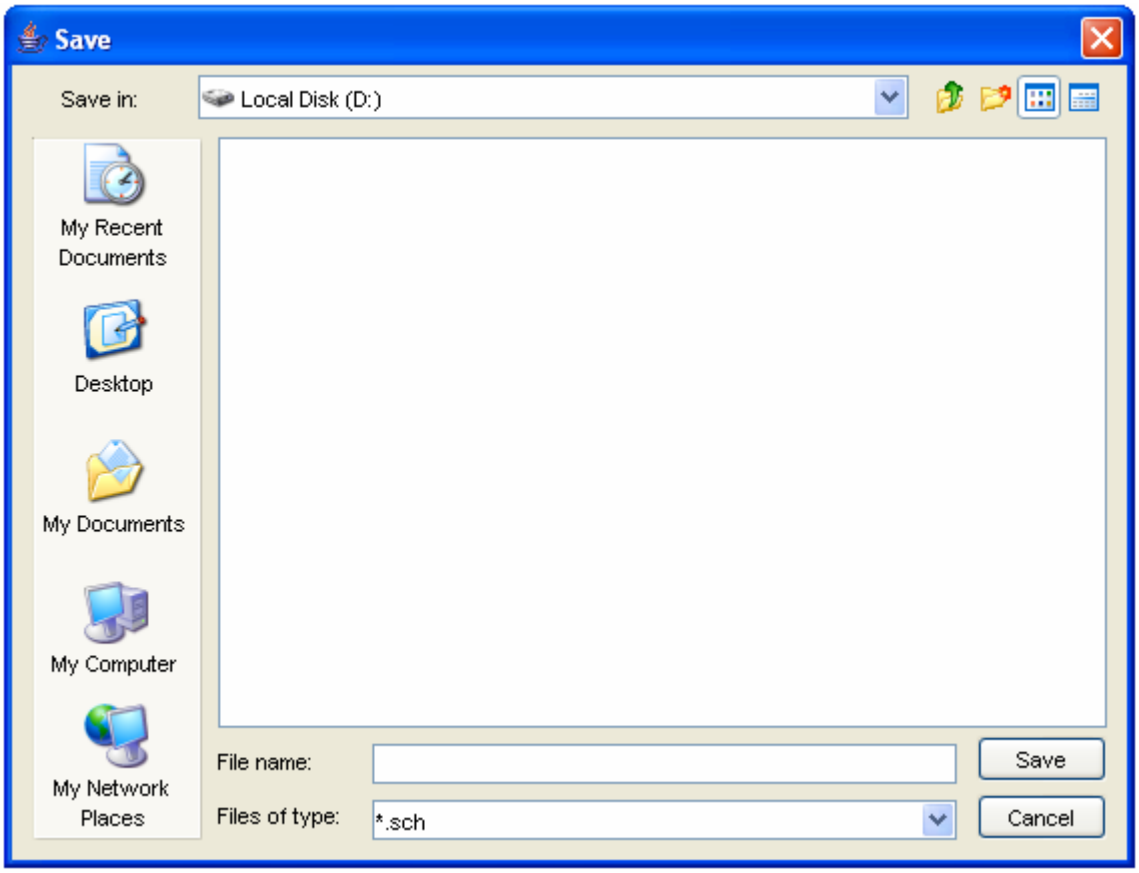


Figure 3-32: Save File Chooser Dialog

vi Server File Manager Dialog

The server file manager dialog is a user input window by which the user can manipulate the remote schematic files at the server. The dialog has the following actions: “Connect,” “Disconnect,” “Upload,” “Download,” “MK Directory,” “RM Directory,” “Delete” and “Exit.” The dialog will be displayed when “Server File manager” action is chosen from either the standard toolbar or the file manager tools panel. Figure 3-33 represents the server file manager dialog.

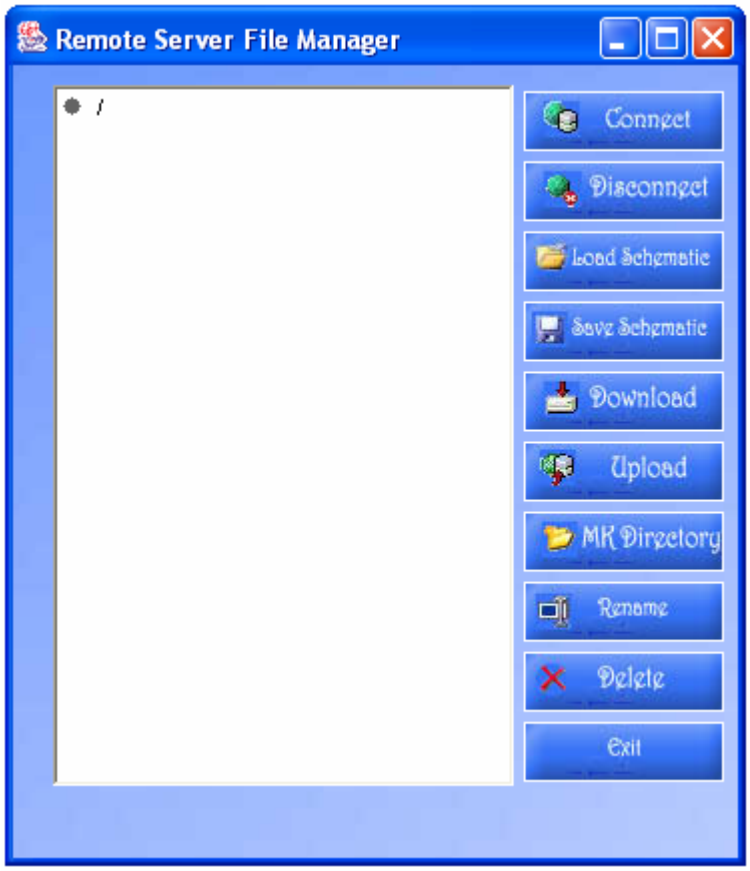


Figure 3-33: Server File Manager Dialog

vii Simulation Settings Dialog

The simulation settings dialog is an input window by which the user can edit the transient simulation settings. The dialog will be displayed when the “Simulation Settings” action is chosen from either the simulation toolbar or the simulation tools panel. Figure 3-34 represents the simulation settings dialog.

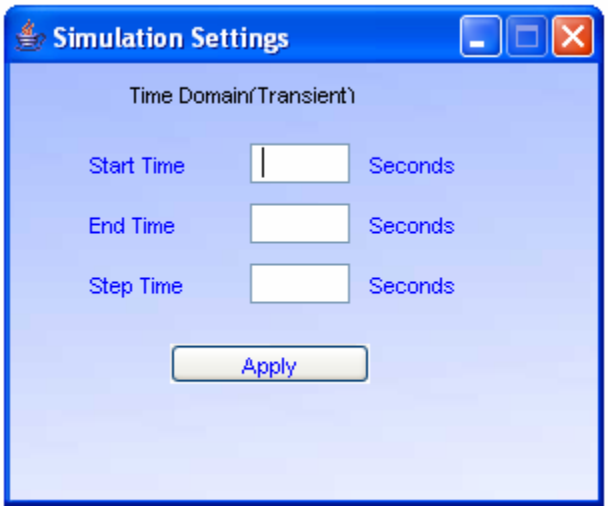


Figure 3-34: Simulation Settings Dialog

viii Netlist Display Dialog

The Netlist display dialog is an output window by which the user can display the generated schematic Netlist. The dialog will be displayed when the “Create Netlist” action is chosen from either the simulation toolbar or the simulation tools panel. Figure 3-35 represents the Netlist display dialog.

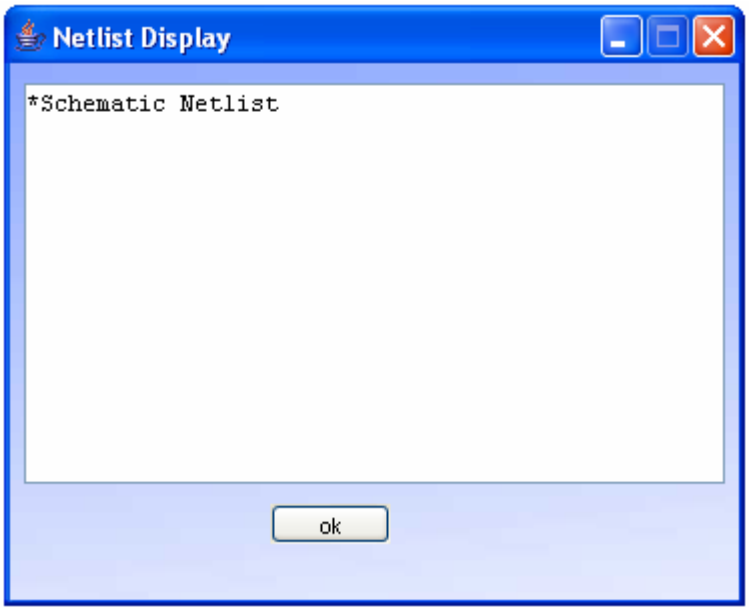


Figure 3-35: Netlist Display Dialog

ix Netlist Editor Dialog

The Netlist editor dialog is an input window by which the user can edit the Netlist. The dialog will be displayed when the “Edit Netlist” action is chosen from either the simulation toolbar or the simulation tools panel. Figure 3-36 represents the Netlist editor dialog.

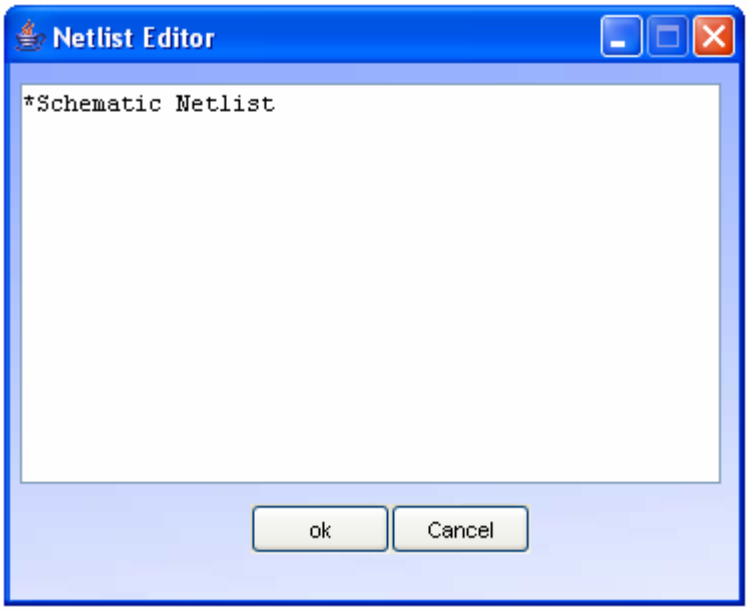


Figure 3-36: Netlist Editor Dialog

x CIRML Display Dialog

The CIRML display dialog is an output window by which the user can display the CIRML data format. The dialog will be displayed when the “Edit Netlist” action is chosen from the simulation toolbar. Figure 3-37 represents the CIRML display dialog.

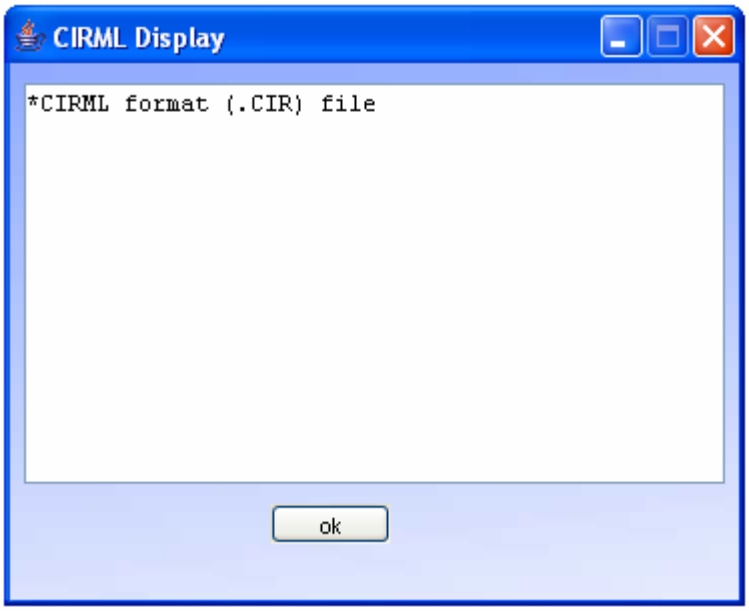


Figure 3-37: CIRML Display Dialog

xi Output Results Display Dialog

The output results display dialog is an output window by which the user can display the text simulation results. The dialog will be displayed when the “Run” and “View Simulation Result” actions are chosen from the simulation toolbar. Figure 3-38 represents the output results display dialog.

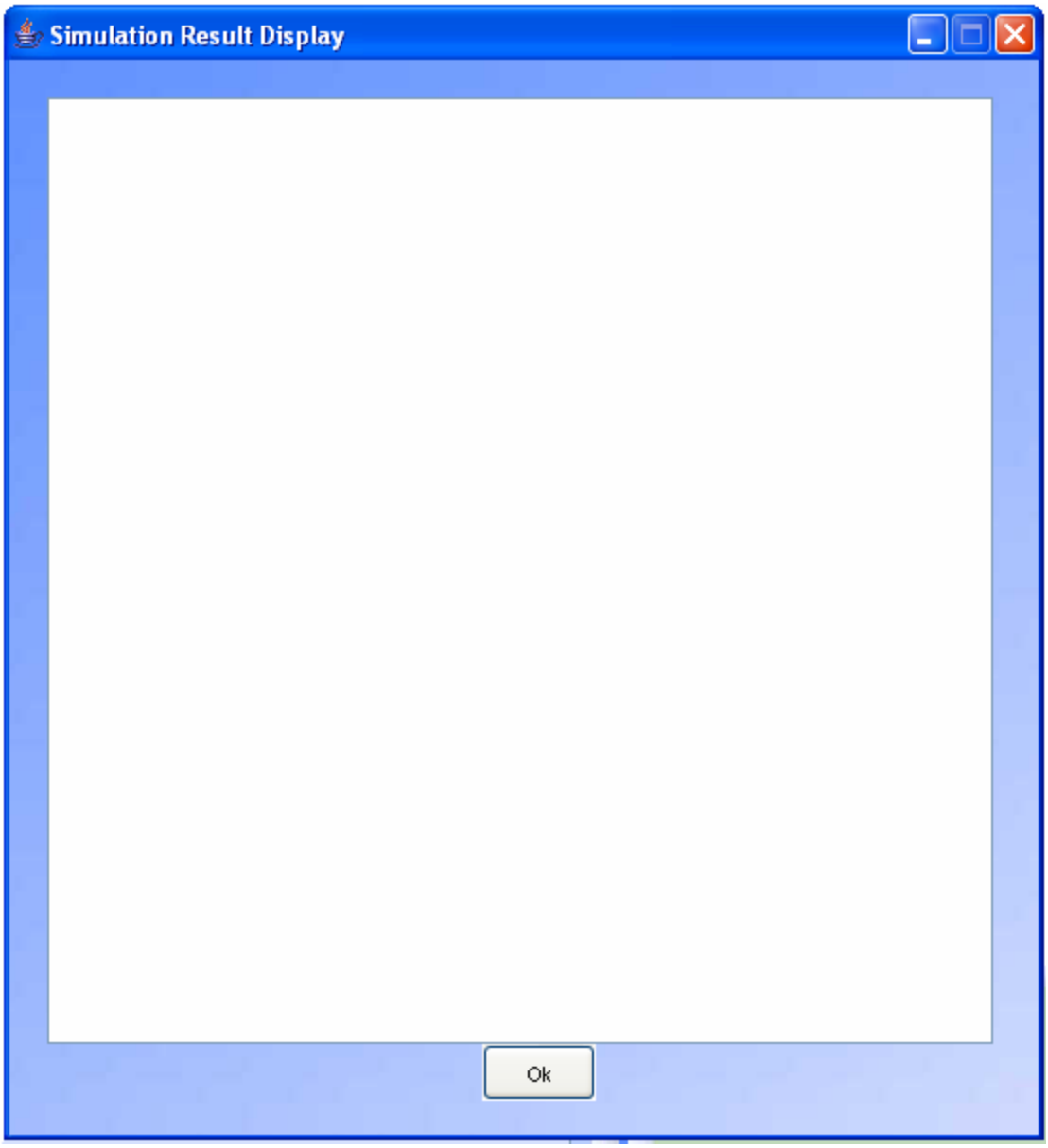


Figure 3-38: Output Result Display Dialog

xii Online Help Window

The online help window is an online helping tool that contains the system user manual and helping tutorial material for how to use the GUI. The dialog will be displayed when the

"Help" action is chosen from the standard toolbar or the "F1" key is pressed. Figure 3-39 represents the outline results display dialog.

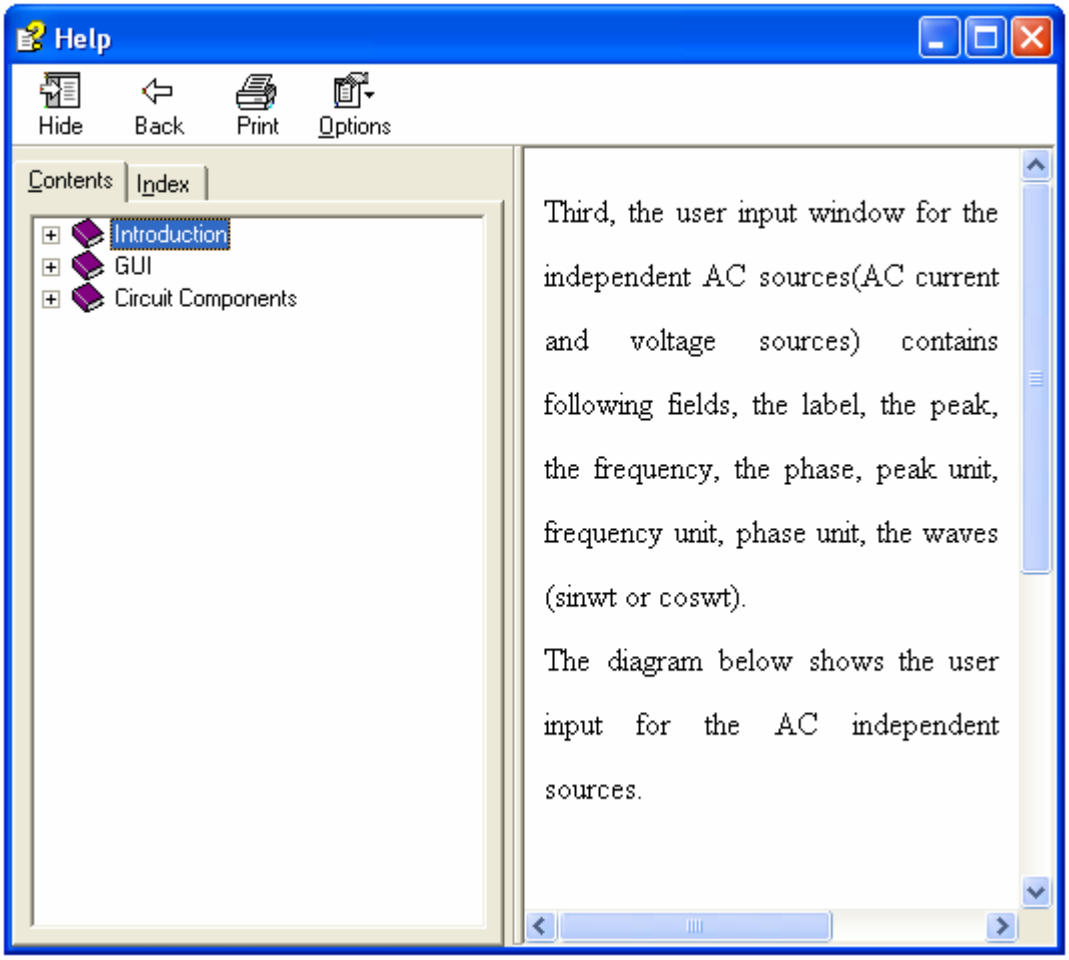


Figure 3-39: Online Help Topics Window

c) Drawing Area Popup Messages

i Welcome Message

The welcome message will be shown when the applet starts. Figure 3-40 represents the welcome message window.

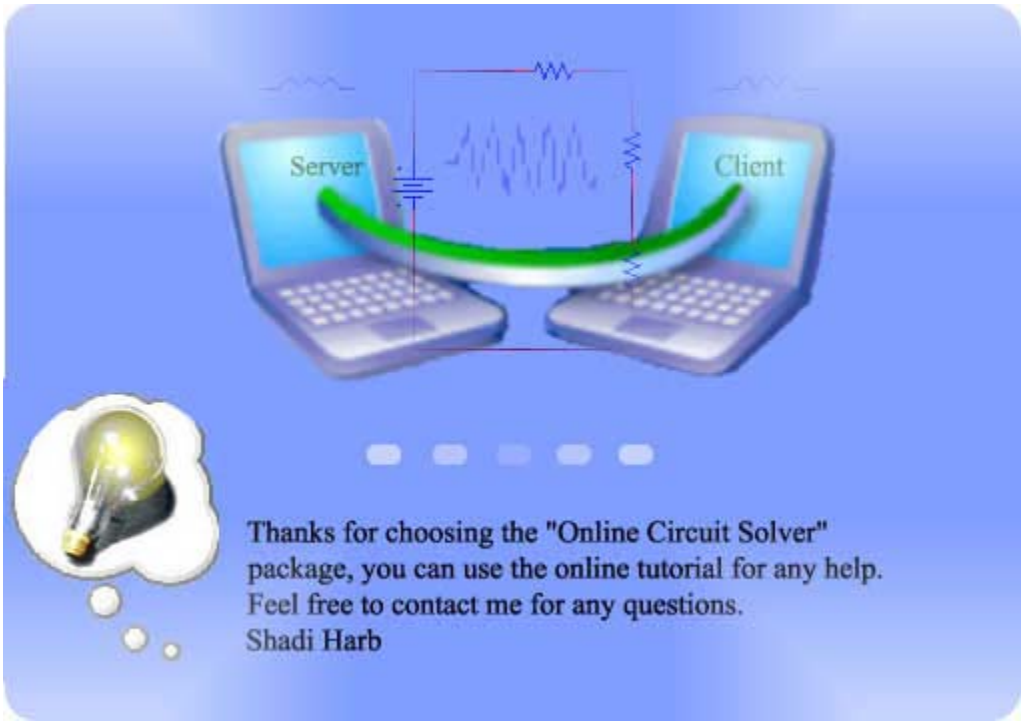


Figure 3-40: Welcome Message

ii Tip Message

The startup tip message also will be shown when the applet starts. Figure 3-41 represents the tip window.

Figure 3-41: Startup Tip Message

iii Save Schematic Page Message

The save schematic question message will be shown when a user deletes the current schematic page. Figure 3-42 represents the save question window.

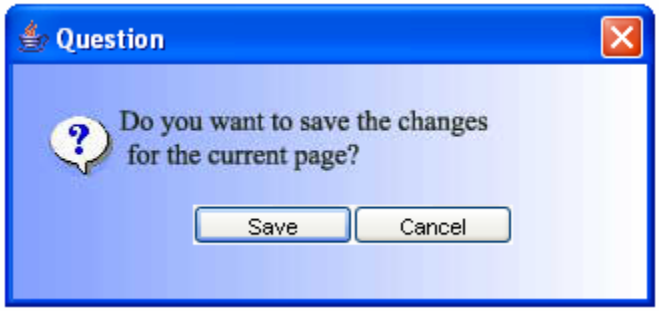


Figure 3-42: Save Schematic Page Question Message

iv Floating Node Message

The floating node alert message will be displayed when a floating node exists in the current schematic. Figure 3-43 represents the floating node alert window.

Figure 3-43: Floating Node Alert Message

v Ground Does Not Exist Message

The ground does not exist message will be displayed when the ground node is missing in the current schematic. Figure 3-44 represents the ground does not exist alert window.

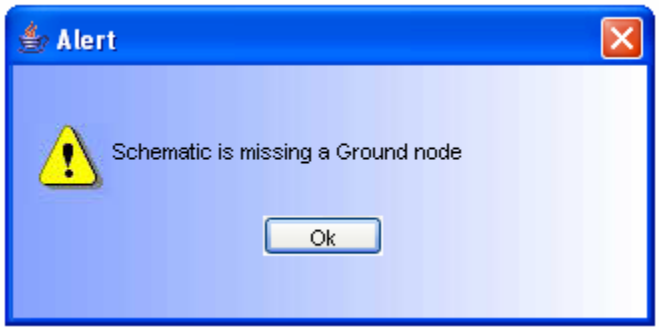


Figure 3-44: Ground Does Not Exist Alert Message

vi Dependent Controller Not Found Message

The dependent controller not found message will be displayed when a dependent controller is missing for a dependent source. Figure 3-45 represents the dependent controller not found alert window.

Figure 3-45: Dependent Controller Not Found Alert Message

vii Schematic Does Not Exist Message

The schematic does not exist error message will be displayed when the user tries to generate the Netlist of an empty schematic. Figure 3-46 represents the schematic does not exist error window.

Figure 3-46: Schematic Does Not Exist Error Message

viii Only Connection Lines Exist Message

The only connection lines exist error message will be displayed when the schematic contains only connection lines. Figure 3-47 represents the only connection lines exist error window.

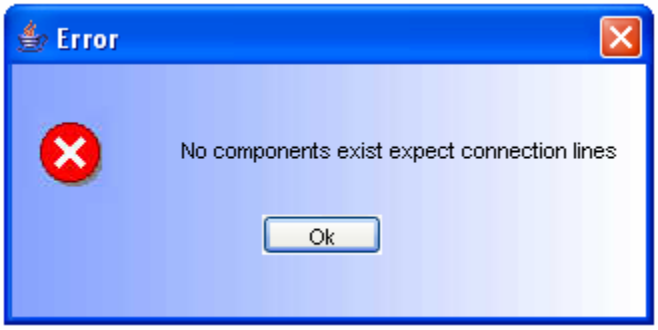


Figure 3-47: Only Connection Lines Exist Error Message

ix Short Circuit Message

The short circuit error message will be displayed when a short loop exists in the current schematic. Figure 3-48 represents the short circuit error window.

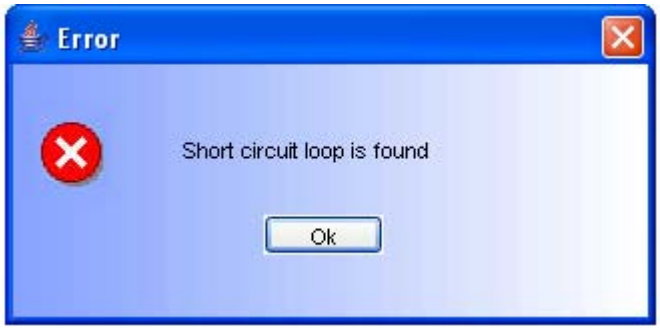


Figure 3-48: Short Circuit Error Message

3.3.2 GUI Manipulation

The GUI has following actions which manipulate drawing the circuit components on the grid area.

i) Placing Components on the Drawing Area

A component can be placed on the grid by two ways: The first is to click on the component image button in the elements panel and drop it on the grid by clicking the mouse again on the drawing area. The second way is to drag the component image from the elements panel and simply drop it on the drawing area after releasing the mouse. The same visited components can also be placed the same way from the visited elements panel.

ii) Displaying Components

A component's properties can be displayed such as the component attributes display format. The component rotation as well as component background and foreground colors can be edited two ways. The first is simply to select the component, right click on the drawing area and choose the "Display Properties" action from the displayed popup menu. The second way is to choose the same action from the component toolbar. The Display Properties dialog will be displayed when "Display Properties" is chosen.

iii) Editing components

A component can be edited by four ways. The first is simply to double-click on the component. The second is to right-click on the grid area after selecting the component. As a

result, the popup menu will be displayed, and the user can choose the “Properties” action from the menu to edit the component. The third option is to double-click on a component’s shown attributes (name or value). The forth way is to choose the “Edit Properties” action from the component toolbar. When the edit action is performed, an input user window will be displayed showing the component attributes, so the user can change them.

iv) Selecting Components

A component or a group of components can be selected two ways. The first way is to click on the component itself. In the case of a single component, the component and its attributes will be displayed in red color (Active State). The second way is to drag the mouse on the drawing area, which contains the component or the group of components. To select all components, use the “Select All” action at the standard toolbar or the “CTRL+A” key can be pressed to do the same action. When the component or groups of components are selected, they become active and displayed in red color, and they become ready to accept several actions.

v) Rotating Components

A component or a group of components can be rotated either to the right or to the left at right angles (90°, 180°, 270° or 360°) simply by selecting the component or the group of components. Then either right-click on the grid area and choose the “Rotate Right” and “Rotate Left” actions from the displayed popup menu, or choose the same actions from the component toolbar. The “CTRL+R” or “CTRL+L” keys also can be pressed to do the same actions.

vi) Flipping Components

A component or a group of components can be flipped either vertically or horizontally by simply selecting the component or the group of components and right-clicking on the grid area and choosing the “Flip Vertical” and “Flip Horizontal” actions from the displayed popup menu. Or, choose the same actions from the component toolbar. “CTRL+SHIFT+V” or “CTRL+SHIFT+H” keys can be pressed also to do the same actions.

vii) Resizing Components

Since component images can’t be resized in the current prototype, the user can resize the component virtually by drawing connection lines at the component end points. The user can simply select the component and start dragging at the end points, and a connection line should appear that starts from the end point that has been pressed.

viii) Cutting Components

A component or a group of components can be cut by simply selecting the component or the group of components, right-clicking on the grid area and choosing the “Cut” action from the popup menu. The “Cut” action button on the standard toolbar can be used as well, and the “CTRL+X” keys can be pressed also to do the same action.

ix) Copying Components

A component or a group of components can be copied by selecting the component or the group of components, right-clicking on the grid area and choosing the “Copy” action from the

popup menu. The “Copy” action button on the standard toolbar can be used as well, or the “CTRL+C” key can be pressed also to do the same action.

x) Pasting Components

A component or a group of components can be pasted by right-clicking on the grid area and choosing the “Paste” action from the popup menu. The “Paste” action button on the standard toolbar can be used as well and the “CTRL+V” keys can be pressed to do the same action.

xi) Deleting Components

A component or a group of components can be deleted by selecting the component or the group of components, right-clicking on the grid area and choosing the “Delete” action from the popup menu. The “Delete Component” action button on the standard toolbar can be used, and the “DEL” key can be pressed to do the same action.

xii) Grouping Components

Components can be grouped by selecting the components, right-clicking on the grid area and choosing the “Group” action from the popup menu. Or they can be grouped by choosing the same actions from the component toolbar. The “CTRL+G” keys can be pressed also to do the same action. By grouping components, they become one unit, and the user can such move them and rotate them as closed box components.

xiii) Ungrouping Components

Components can be ungrouped by simply selecting the components, right-clicking on the grid area and choosing the “Ungroup” action from the popup menu or by choosing the same actions from the component toolbar. The “CTRL+U” keys can be pressed also to do the same action. By ungrouping the set of components, each one becomes separate and receives all the single element actions.

xiv) Regrouping Components

Components can be regrouped by simply selecting the components, right-clicking on the grid area and choosing the “Regroup” action from the popup menu or by choosing the same action from the component toolbar. The “CTRL+O” keys can be pressed also to do the same action. By regrouping the set of components, all the subgroup components will become one unit.

xv) Aligning Components

Components can be aligned to top, bottom, right and left by simply selecting the components, right-clicking on the grid area and choosing from the displayed popup menu either the “Align Top,” “Align Bottom,” “Align Right” or “Align Left” actions. Or the user can choose the same actions from the component toolbar. “CTRL+SHIFT+T,” “CTRL+SHIFT+B,” “CTRL+SHIFT+R” and “CTRL+SHIFT+L” keys can be pressed also to do the same action respectively.

xvi) Undoing and Redoing Actions

The last grid actions that have occurred on the drawing area can be recovered by using the “Undo” and “Redo” actions. The undo action can be chosen by simply right-clicking on the grid area and choosing the “Undo” action from the displayed popup menu. The “Undo” and “Redo” actions at the standard toolbar can be used as well. “CTRL+Z” and “CTRL+D” keys can be pressed also to do the same action respectively.

xvii) Clearing All Components

All components can be deleted one of two ways. The first is to right-click on the drawing area and choose the “Clear All” action from the grid popup menu. The second way is select all components either by clicking on the “Select All” action at the standard toolbar or draging the mouse on the whole grid area and then either choosing the “Delete Component” action at the standard toolbar or pressing the “DEL” key to do the same action.

xviii) Connections Line Manipulation

A connection line can be drawn by simply clicking on the connection line image button at the connections elements panel. The starting point of the line is determined when the mouse is pressed on the grid. Then the mouse is dragged to where the desired end point is located. The connection line can be resized by selecting the line first and then dragging the line from the end points. The connection line can be moved by selecting the line and dragging at any point located on the line except the end points. Connection lines can be split at the intersection point to make it electrically connected.

xix) Drawing Area Manipulation

The drawing area can be zoomed in or zoomed out in two ways. The first is to right-click on the drawing area and to choose either the “Zoom In” or “Zoom Out” actions from the grid popup menu. The second is to use “Zoom In,” “Fit Zoom” and “Zoom Out” actions from the canvas toolbar. The drawing area can be scrolled horizontally and vertically either by choosing the horizontal and vertical scrolling bars of the grid or by selecting the “Scroll Hand” action from the canvas toolbar to do the same action. The “Go To” action that can be chosen from the canvas toolbar or from the displayed grid popup menu will move the current page position to the desired location by changing the x and y positions of the drawing area starting point. The drawing area can also be resized and the fill and line color of the grid area can be changed by choosing the “Canvas Colors” and “Canvas Size” actions from the canvas toolbar. The “Canvas Properties” dialog will be displayed to edit the canvas settings.

xx) Simulation Manipulation

The GUI gives the user several powerful tools to manipulate the simulation process of the schematic. Netlist can be edited manually using either the “Edit Netlist” action at the simulation toolbar or the “Edit Netlist” action button at the simulation tools panel. Netlist can be generated from the current drawing schematic by using either the “Create Netlist” action on the simulation toolbar or the “Create Netlist” action button on the simulation tools panel. The simulation can be run by using either the “Run” action at the simulation toolbar or the “Simulate” action at the simulation tools panel. Simulation settings can be edited using either the “Simulation Settings” action on the simulation toolbar or the “Simulation Settings” action button on the simulation tools panel. Nodes Net Names, Nodes Voltages and Nodes Currents can be shown by using the

"Display Netlist Nodes," "Display Voltages," or "Display Currents" actions on the simulation toolbar respectively. Voltage Marker, Voltage Differential Marker and Current Marker can be chosen from either the connections elements panel or the markers actions at the simulation toolbar. The simulation output result and the CIRML result can be display respectively using the "Display Output Result" and "Display CIRML Result" actions on the simulation toolbar.

xxi) Schematic Page Manipulation

Several actions can be applied on the schematic pages. A new page can be created by choosing either the "New Schematic Page" action on the standard toolbar or the "New Schematic" action button in the file manager tools panel. Current Schematic Page can be saved as a ".sch" file at the local client by choosing either the "Save Schematic Page" action on the standard toolbar or the "Save Schematic" option in the file manager tools panel. A schematic file can be loaded at the current schematic page by choosing either the "Load Schematic Page" action at the standard toolbar or the "Open Schematic" action button at the file manager tools panel. A current schematic can be deleted from the workplace by choosing either the "Delete Current Page" action at the standard toolbar or the "Delete Schematic" action button at the file manager tools panel. The Server File Manager can be accessed by choosing the "Server File Manager" action button at the file manager tools panel. It provides the user will all the file management tools to manipulate the remote schematic files at the server. Schematic pages can also be explored by choosing either the "Last Forward," "Forward," "Backward" or "Last "Backward" actions from the standard toolbar or the page number action from the schematic pages toolbar.

3.3.3 Component Types and Definitions

The current prototype version provides the user with the basic circuit elements, which can be classified by the following four basic groups:

i) Passive Elements Group

The passive elements group contains four circuit components, Resistor, Capacitor, Inductor and Transformer. Figure 3-49 shows the passive elements panel.

Figure 3-49: Passive Elements Panel

a) Resistor

The resistor component is a two nodes component symbol, which represents a resistor element with resistance. The resistance's value has the following units: µOhm, mOhm, Ohm, KOhm and MOhm. Resistor tolerance is not included in this version. The resistor symbol is shown in Figure 3-50.

Figure 3-50: Resistor Symbol

b) Capacitor

The capacitor component is a two nodes component symbol that represents a capacitor element. Its value has the following units: pF, µF, mF and F. The Capacitor symbol is shown in Figure 3-51.

Figure 3-51: Capacitor Symbol

c) Inductor

The inductor component is a two nodes component symbol, which represents an inductor element. Its value has the following units: µH, mH and H. The inductor symbol is shown in Figure 3-52.

Figure 3-52: Inductor Symbol

ii) Transformers Group

The transformers group contains four transformer types: Positive/Positive Transformer, Positive/Negative Transformer, Negative/Positive Transformer and Negative/Negative Transformer. Figure 3-53 shows the transformers panel.

Figure 3-53: Transformers Panel

a) Positive/Positive Transformer

The first type of transformer is represented by a four nodes component symbol with positive/positive polarities. Its attribute is defined by the turn ratio of the primary winding (n1) and the secondary winding (n2). Figure 3-54 represents the first type of transformer symbol.

Figure 3-54: Positive/Positive Transformer Symbol

b) Positive/Negative Transformer

The second type of transformer is represented by a four nodes component symbol with positive/negative polarities. Its attribute is defined by the turn ratio of the primary winding (n1) and the secondary winding (n2). Figure 3-55 represents the second type of transformer symbol.

Figure 3-55: Positive/Negative Transformer Symbol

c) Negative/Positive Transformer

The third type of transformer is represented by a four nodes component symbol with negative/positive polarities. Its attribute is defined by the turn ratio of the primary winding (n1) and the secondary winding (n2). Figure 3-56 represents the third type of transformer symbol.

Figure 3-56: Negative/Positive Transformer Symbol

d) Negative/Negative Transformer

The forth type of transformer is represented by a four nodes component symbol with negative/negative polarities. Its attribute is defined by the turn ratio of the primary winding (n1) and the secondary winding (n2). Figure 3-57 represents the forth type of transformer symbol.

Figure 3-57: Negative/Negative Transformer Symbol

iii) Independent Sources Group

The independent sources group contains four circuit elements: the DC Voltage source, the DC Current Source, the AC Voltage Source and the AC Current Source. Figure 3-58 shows the independent sources group panel.

Figure 3-58: Independent Sources Group Panel

a) DC Voltage Source

The DC voltage source is a two nodes component symbol that represents a DC voltage source element. Its voltage value is defined by the following units: µV, mV, V and KV. Figure 3-59 represents the DC voltage source symbol.

Figure 3-59: DC Voltage Source Symbol

b) DC Current Source

The DC current source is a two nodes component symbol that represents a DC current source element. Its current value is defined by the following units: µA, mA, A and KA. Figure 3-60 represents the DC current source symbol.

Figure 3-60: DC Current Source Symbol

c) AC Voltage Source

The AC voltage source is a two nodes component symbol that represents an AC voltage source element. Its voltage peak value is defined by the following units: µV, mV, V and KV. Figure 3-61 represents the AC voltage source symbol.

Figure 3-61: AC Voltage Source Symbol

d) AC Current Source

The AC current source is a two nodes component symbol that represents an AC voltage source element. Its voltage peak value is defined by the following units: µA, mA, A and KA. The frequency unit is defined by the following units: Hz, KHz and MHz, and the phase value is defined by “deg” and “rad” units. Figure 3-62 represents the AC current source symbol.

Figure 3-62: AC Current Source Symbol

iv) Dependent Sources Group

The dependent sources group contains four circuit elements: the Voltage Controlled Voltage Source, the Voltage Controlled Current Source, the Current Controlled Voltage Source and the Current Controlled Current Source. Figure 3-63 shows the dependent sources group panel.

Figure 3-63: Dependent Sources Panel

a) Voltage Controlled Voltage Source (VCVS)

The VCVS element represents the relationship between the controlled voltage source and its controlling voltage. The ratio between them represents the gain value without unit. Figure 3-64 represents the VCVS symbol.

Figure 3-64: Voltage Controlled Voltage Source Symbol

b) Voltage Controlled Current Source (VCCS)

The VCCS element represents the relationship between the controlled current source and its controlling voltage. The ratio between them represents the gain value, which has an admittance unit given by Siemen (µS, mS, S, KS and MS). Figure 3-65 represents the VCCS symbol.

Figure 3-65: Voltage Controlled Current Source Symbol

c) Current Controlled Voltage Source (CCVS)

The CCVS element represents the relationship between the controlled voltage source and its controlling current. The ratio between them represents the gain value, which has a resistance

unit given by Ohm (µOhm, mOhm, Ohm, KOhm and MOhm). Figure 3-66 represents the CCVS symbol.

Figure 3-66: Current Controlled Current Source Symbol

d) Current Controlled Current Source (CCCS)

The CCCS element represents the relationship between the controlled current source and its controlling current. The ratio between them represents the gain value without unit. Figure 3-67 represents the CCCS symbol.

Figure 3-67: Current Controlled Current Source Symbol

v) Connections Group

The connections group contains five elements: the connection line, the connection node, the ground, the voltage marker and the current marker. Figure 3-68 shows the connections group panel.

Figure 3-68: Connections Panel

a) Connection Line

The connection line can be drawn to connect between circuit components as a physical connection symbol. Figure 3-69 represents the connection line symbol.

Figure 3-69: Connection Line Symbol

b) Connection Node

The connection node can be used to make electrically connected points in the schematic circuit. Figure 3-70 represents the connection node symbol.

Figure 3-70: Connection Node Symbol

c) Ground

The ground represents the reference node for all components in the schematic. All voltages in the schematic refer to that reference node, which is considered a zero voltage node by default. Any schematic circuit can have more than one ground node. Figure 3-71 represents the ground symbol.

Figure 3-71: Ground Symbol

d) Voltages Marker

The voltage marker can specify the voltage difference between any node and the reference point. Figure 3-72 represents the voltage marker symbol.

Figure 3-72: Voltage Marker Symbol

e) Differential Voltages Nodes Marker

The voltage differential marker can specify the differential voltage between two nodes for VCVS and VCCS. The differential marker exists to determine the controlling voltage source. Figure 3-73 represents the differential voltage marker symbol.

Figure 3-73: Differential Voltage Marker Symbol

f) Current Marker

The current marker can specify the branch that carries the current with a specific label for CCVS and CCCS. The current branch exists to determine the controlling current source. Figure 3-74 represents the current marker symbol.

Figure 3-74: Current Marker Symbol

CHAPTER FOUR: SYSTEM DESIGN AND IMPLEMENTATION

4.1 System Platform Design

4.1.1 JAVA as a Developing Programming Language

JAVA is a viable and portable language that is commonly used in developing web-based applications with different platforms support, it provides Java Applets, which run within java-enabled browsers. The browser runs the applet by interpreting the “byte code” to the native machine code using an intermediate interface called Java Virtual Machine (JVM). The current prototype is developed by JAVA (JDK Ver. 1.4), and the front end Java Applet can be run by accessing the project homepage by any java-enabled browser.

4.1.2 JBUILDER as a Development System Environment

Borland JBUILDER V9.0 is used as a development environment for the project. It has a powerful compiler, debugger, applet viewer and good help and search tools. The development environment is supported by a java development kit (JDK Ver. 1.4). Although JBUILDER provides easy and simple auto generation code, all the code was written from scratch and none of these codes have been used to avoid the risk of using dependent environment code.

4.2 System Architecture Design

The online package is a Web-based client/server software that consists of two main packages: The Front-End Client Software Package and The Background Server Software Package. A detailed system architecture overview is shown in Figure 4-1.

4.2.1 Front-End Client Software Package (FECSP)

The client side is presented by two main applications: front end user interface (GUI) and client application. The GUI interface is designed by JAVA Applet. It can be interacted directly by using its buttons and panels. A circuit schematic can be built and saved either locally or remotely at the server and restored in the same way. A schematic Netlist can be generated and examined against various common errors. The client application runs two services: the circuit simulation and evaluation service and the file management service. The circuit simulation sends the saved data components from the local memory to the server to start the actual simulation, and eventually gets back the simulation results at the client for display. The file management service manipulates all the file management actions either locally at the client or remotely at the server.

4.2.2 Background Server Software Package (BSSP)

The server side is presented by two main applications, which are the server application and the third party simulation package. The server application runs two services: the circuit simulation and evaluation service and the file management service. The circuit simulation service consists of the Netlist parser and the third party interface. The Netlist parser converts the Netlist to CIRML format file (.cir) and the third party interface is responsible for automating the

simulation process and exchanging the information between the third party simulation package and the simulation service. The file management service is responsible for manipulating all the file management actions such as saving and loading schematic files at the server. The third party application (PSPICE), which is used as a real circuit simulation and analysis package, does the actual simulation and generates the simulation output files.

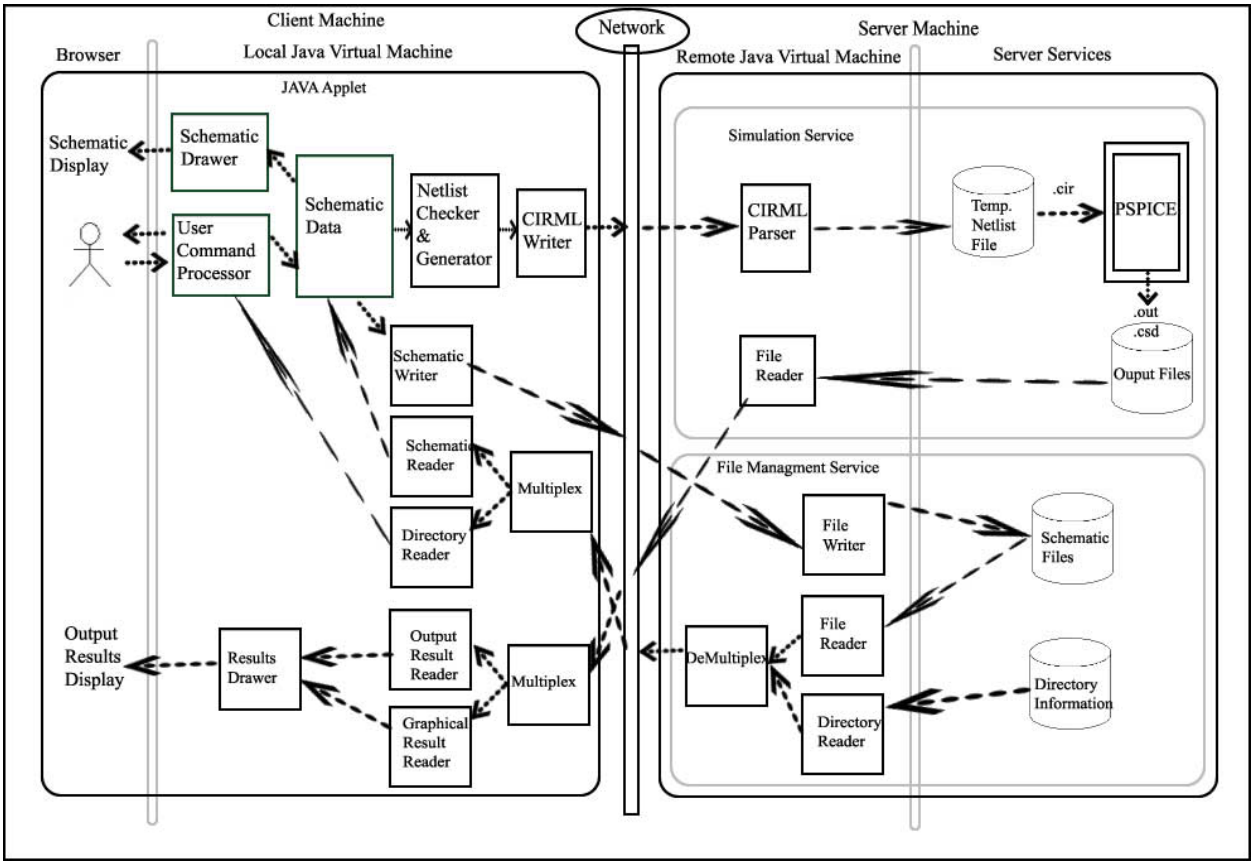


Figure 4-1: Detailed System Architecture Overview

4.3 Global Data Structure Improving

Many features and improvements have been incorporated into the existing data structure to accommodate new data and function implementations.

4.3.1 Problems in the Previous Design and Code

The previous work of the project subject is appreciated, including the general framework of the prototype that was partially implemented but with limited functions. The new implementation added new functionalities and features including solving many persisting code problems found in the previous work such as:

- The class hierarchy is too flat in the old design
- Polymorphism is not really utilized
- Redundant variables

4.3.2 Modified Global Data Structure

Compromising between keeping the existing code and refining the data structure is a critical issue and a big challenge in software coding. While refining the existing data structure facilitates implementing new features and functionalities, keeping the old structure makes the modifications and maintenance much easier. Therefore, a trade-off exists between replacing the old code with the new one and conforming to the previous code.

4.4 Module Design

4.4.1 Client Applet Design

Client Applet is the main part of the online package material. It is responsible for implementing all the functionalities and run time services supported by the front-end graphical user interface. The applet deals with all graphics (panels, images) and all information used in the GUI drawing. The applet also is responsible for keeping track of a considerable amount of data components structure and for manipulating all the component actions and services that facilitate drawing components on the drawing area in addition to all the attached services to run the circuit simulation and manage the schematic files.

4.4.2 Netlist Manipulation

Netlist generation is a very important step prior to the simulation phase. The Netlist holds all the circuit information (circuit topology, components attributes) used to initiate the simulation process.

i) Netlist Checker

Checking the Netlist against various common errors is an initial step in the Netlist generation process. The Netlist should be free from all expected errors so that it can be parsed correctly to the proper format for the simulation phase. The current prototype implements a basic Netlist checker, which checks the circuit against the following errors:

a) Schematic is empty

An error message is displayed when no component is found.

b) Short circuit loops

An error message is displayed when a short circuit loop is found in the schematic.

c) Connection lines are the only existing components

An error message is displayed when only connection lines are found in the schematic.

d) Node floating error

An alert message is displayed when a floating node is found in the schematic.

e) Ground does not exist

An alert message is displayed when ground is not found in the schematic.

f) Dependent controllers do not exist

An alert message is displayed when the dependent source controller is not found.

ii) Netlist Generation Concept

A schematic drawing displays a set of circuit components and their connection lines, which connect between other components. Nodes that are connected by one or more connections are considered electrically connected. To express connectivity, each node is assigned a net name, and all the nodes that are electrically connected belong to the same net name. If two nets are

electrically connected, they must be reduced to only one net name. The Netlist contains the components' information and their connectivity net names, which should be conformed to the following format:

<Component_Name> <Net_Name> + <Value><Unit>

iii) Netlist Generation Steps

Netlist generation has three main steps, which are the connection optimization, the net name assignment and the Netlist output. The connection optimization is the first important step of the Netlist generation and since the nodes of the connection line are electrically connected, they must be reduced to one net name. In this step, all the connection lines are removed and the components nodes are virtually moved to equivalent positions by which they are connected directly to the other components nodes without connection lines. The second step is the net name assignment, and all components are traversed to assign for each node a net name. If there are two nodes that are connected by a connection line, they should have the same net node. The last step is the Netlist output, which describes each component with its node's net name. The ground connection nodes should have the same net node name "N_0" since all ground nodes are electrically connected.

iv) Example of Netlist Generation Steps

Figure 4-2 shows a sample schematic circuit, and each connection line is marked with a number.

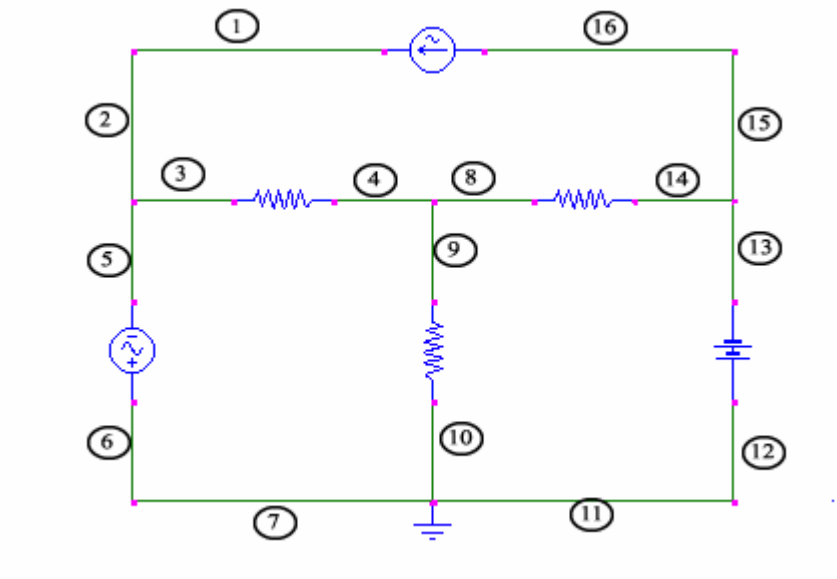


Figure 4-2: Sample Schematic for Netlist Generation Steps

After connection line 1 is removed, the circuit will look like Figure 4-3.

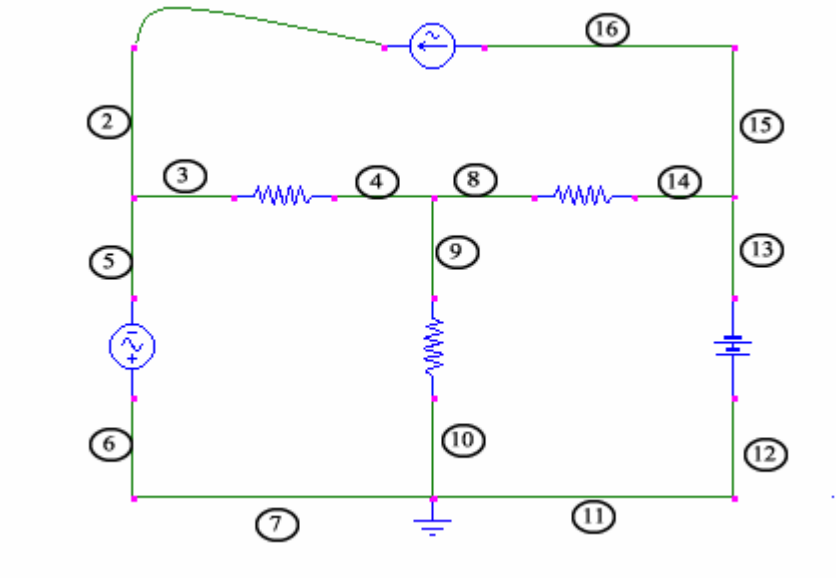


Figure 4-3: Connection Line Optimization, Step 1

Figure 4-4 shows the circuit after removing connection lines 1, 2 and 3.

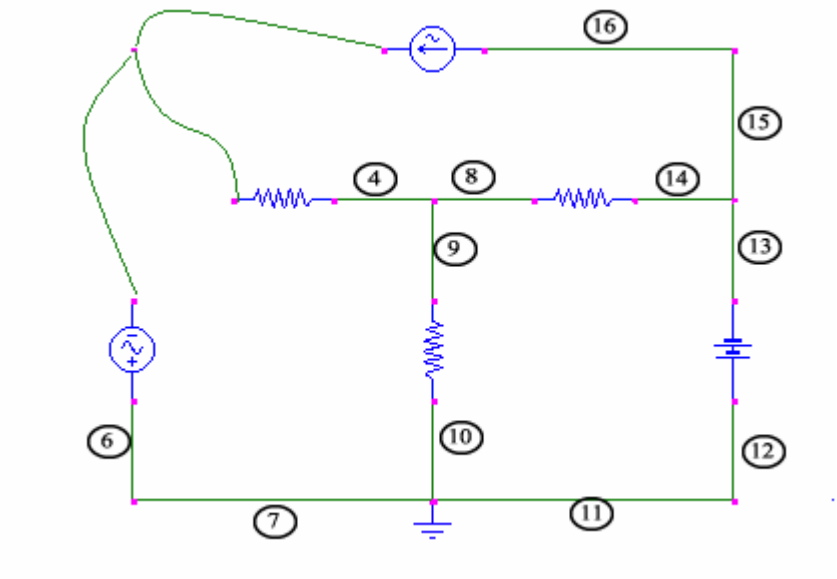


Figure 4-4: Connection Line Optimization, Step 2

After removing all the connection lines, the circuit will look like Figure 4-5.

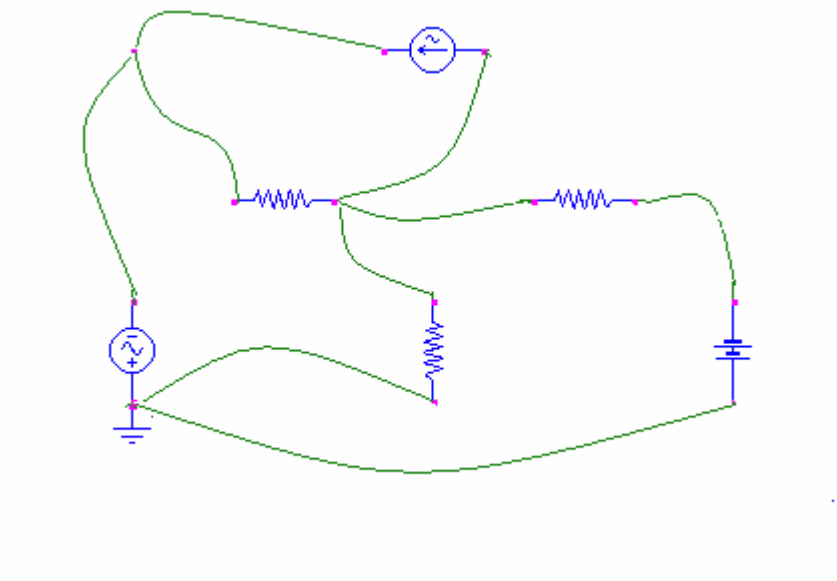


Figure 4-5: Connection Line Optimization, Final Step

The above figures of the Netlist generation demonstrate the connection line optimization step. Now, the next step is the net name assignment, and each node is denoted as a dot node in figure 4-6 and will be assigned a distinctive net name.

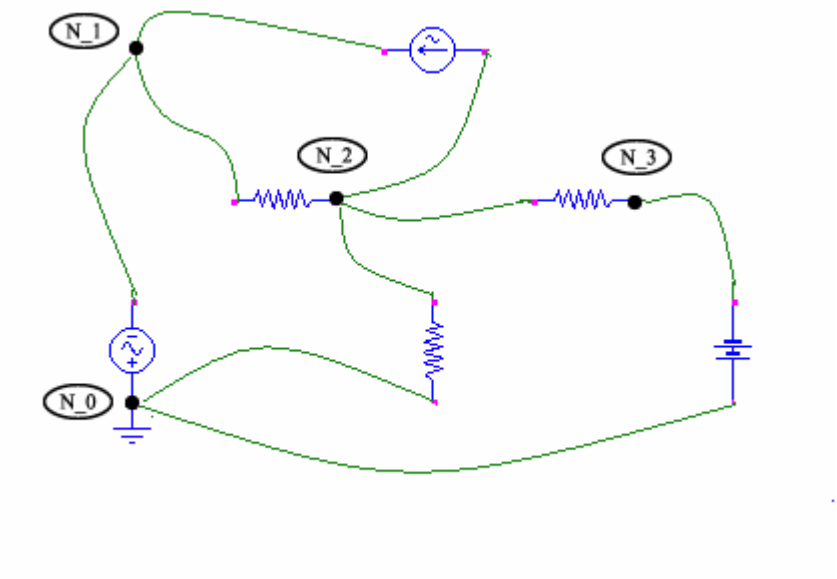


Figure 4-6: Net Name Assignment Step

The last step of the Netlist generation is the Netlist output. All components should be traversed to generate the Netlist. For example, the output Netlist for Figure 4-6 could be the following:

```
R_R1 N_2 N_0 100Ohm
R_R2 N_1 N_2 200Ohm
R_R3 N_2 N_3 40Ohm
Vdc_Vdc1 N_3 N_0 5V
Vac_Vac1 N_1 N_0 5V
Iac_Iac1 N_1 N_2 3mA
```

v) Netlist Implementation

Since the components node position is not changeable with software, an equivalent node position technique is used to implement the new node position. Each node has two positions: the actual position and the equivalent position. Before the connection optimization step, the actual

node position is the same as the equivalent one when the node position changes during the connection optimization step. The equivalent position is only updated by the new one. After the connection lines are removed, all components are traversed to assign each node a net name. If two nodes are connected by a connection line, they should have the same equivalent position and the same net name. Each distinguished equivalent position and net name is inserted into a net name vector, and any component node that has the same equivalent position entry in the vector will be assigned the associated net name.

vi) The Connection Optimization Algorithm

The connection optimization process can be described by the following Pseudo Code:

```
/** OptimizeNetwork is the main function that removes all the connection lines from the network */
OptimizeNetwork ()
{
For (all connection lines)
{
For (all other connection lines)
{
MergeNet (the current connection line, the other connection line)
}
For (all other components)
{
```

MergeNet (component, the current connection line)

}

}

}

//MergeNet is the function that updates the equivalent positions for all components nodes.

MergeNet (Component, Connection line)

{

Get the connection line's coordinate of node 1 and node 2

If the component's node 1 has the same coordinate with the connection's node 1

{Make the component's node1 as coordinate of connection's coordinate node 2}

}

4.4.3 System Main Services

Two main services are attached to the system, which are the simulation and evaluation service and the file management service. Both services are client/server applications. The main function of the two services is exchanging information between the server and the client. Socket interface is used to implement sending and receiving data between client and server. The client simulation service sends the Netlist information as a String object. The server simulation service gets the string and does the proper simulation processing. The two simulation services for the server and the client are connected by the same socket port. When the simulation is finished at the server, the server simulation service will send two object strings. The first object string is the

text output result (.out) and the second is the graphical output result (.csd), the client simulation service receives these string objects and does the output result processing. The file management service has two main functions, the local file management function and the remote file management function, the local file management function saves and restores the components data locally at the client as fertilizable object, some java policy actions should be updated to give the applet user privileges to access the local schematic files which described in Appendix A: User Manual. The remote file management function manipulates all the client/server interactions to access the remote schematic files at the server. The file management service uses a communication socket which is different from the simulation service one, it uses two bidirectional channels to transfer information, each channel works on different socket port number, the first channel which called “Command socket” is used to send the command information which determine the action that should be taken at the server file management service such as open file, save file and so an. The second channel which called “data Socket” sends and receives the actual binary data between client and server. The schematic data is sent and received as a data Object.

4.4.4 System Main Classes Overview

The system software has several main classes that can be considered the backbone of the system package such as mainApplet, ActiveState, Components, DrawTool and ServerApplication. Figure 4-7 shows an overview of hierarchal structure of all the system classes and their internal relationships.

- class java.lang.Object
  - class ECS.**ActiveState**
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Panel (implements javax.accessibility.Accessible)
        - class java.applet.Applet
          - class javax.swing.JApplet (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer)
            - class ECS.**mainApplet**
  - class ECS.**CompsStack**
  - class ECS.**Labels**
  - class ECS.**NeighbourComp**
  - class ECS.**NetNode**
  - class ECS.**Probe**
  - class ECS.**WorkPlace**

- class java.lang.Object
  - class ECS.CircuitComponents.**Components** (implements java.io.Serializable)
    - class ECS.CircuitComponents.**ImgComponents**
      - class ECS.CircuitComponents.**Node**
      - class ECS.CircuitComponents.**Pin1Comp**
      - class ECS.CircuitComponents.**Pin2Comp**
      - class ECS.CircuitComponents.**Pin4Comp**
  - class ECS.CircuitComponents.**CompPos**
  - class ECS.CircuitComponents.**Current**

- class java.lang.Object
  - class ECS.CircuitComponents.Components (implements java.io.Serializable)
    - class ECS.CircuitComponents.ImgComponents
      - class ECS.CircuitComponents.Pin1Components.**ConnectionNode**
      - class ECS.CircuitComponents.Pin1Components.**Ground**
      - class ECS.CircuitComponents.Pin1Comp
        - class ECS.CircuitComponents.Pin1Components.**NodeL**
        - class ECS.CircuitComponents.Pin1Components.**NodeR**

- class java.lang.Object
  - class ECS.CircuitComponents.Components (implements java.io.Serializable)
    - class ECS.CircuitComponents.Pin2Components.**ConnectionLine**
    - class ECS.CircuitComponents.ImgComponents
      - class ECS.CircuitComponents.Pin2Comp
        - class ECS.CircuitComponents.Pin2Components.**CurrentComp**
        - class ECS.CircuitComponents.Pin2Components.**DepComp**

- class java.lang.Object
  - class ECS.CircuitComponents.Components (implements java.io.Serializable)
    - class ECS.CircuitComponents.ImgComponents
      - class ECS.CircuitComponents.Pin4Comp
        - class ECS.CircuitComponents.Pin4Components.**NNTransformer**
        - class ECS.CircuitComponents.Pin4Components.**NPTransformer**
        - class ECS.CircuitComponents.Pin4Components.**PNTransformer**
        - class ECS.CircuitComponents.Pin4Components.**PPTransformer**
        - class ECS.CircuitComponents.Pin4Components.**Transformer**

- class java.lang.Object
  - class ECS.DrawingTools.**DrawTool** (implements java.io.Serializable)
    - class ECS.DrawingTools.**Circle**
    - class ECS.DrawingTools.**Line**
    - class ECS.DrawingTools.**Rectangle**
    - class ECS.DrawingTools.**TextBox**

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Window (implements javax.accessibility.Accessible)
        - class java.awt.Dialog
          - class javax.swing.JDialog (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - class ECS.GUIFrameWindows.**MakeDirectoryFrame**
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - class ECS.GUIFrameWindows.**CloseableFrame**
              - class ECS.GUIFrameWindows.**CanvasPropertiesFrame**
              - class ECS.GUIFrameWindows.**CurrentInputFrame**
              - class ECS.GUIFrameWindows.**DepSourcesInputFrame**
              - class ECS.GUIFrameWindows.**IndepACSourcesInputFrame**
              - class ECS.GUIFrameWindows.**IndepDCSourcesInputFrame**
              - class ECS.GUIFrameWindows.**InsertTextFrame**
              - class ECS.GUIFrameWindows.**NetlistEditorFrame**
              - class ECS.GUIFrameWindows.**OutputFrame**
              - class ECS.GUIFrameWindows.**PassiveInputFrame**
              - class ECS.GUIFrameWindows.**SimulationResultFrame**
              - class ECS.GUIFrameWindows.**SimulationSettingsFrame**
              - class ECS.GUIFrameWindows.**TransformerInputFrame**
            - class ECS.GUIFrameWindows.**RemoteFileManagerFrame**

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Window (implements javax.accessibility.Accessible)
        - class java.awt.Dialog
          - class javax.swing.JDialog (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - class ECS.GUIMessages.**AlertMessage**
            - class ECS.GUIMessages.**ErrorMessage**
            - class ECS.GUIMessages.**InfoMessage**
            - class ECS.GUIMessages.**QuestionMessage**
            - class ECS.GUIMessages.**WelcomeMessage**

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class javax.swing.JComponent (implements java.io.Serializable)
        - class javax.swing.JPanel (implements javax.accessibility.Accessible)
          - class ECS.GUIPanels.**GraphicalPanel**
          - class ECS.GUIPanels.**GraphicalResultPanel**
          - class ECS.GUIPanels.**GridCanvas**
          - class ECS.GUIPanels.**NetlistOutputPanel**
          - class ECS.GUIPanels.**ShowNetlistPanel**
          - class ECS.GUIPanels.**StatusBarPanel**
      - class java.awt.Window (implements javax.accessibility.Accessible)
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - class ECS.GUIPanels.**ColorPanel**

Figure 4-7: An Overview of Hierarchal Structure of all System Classes

i) mainApplet Class

The mainApplet class is the main applet source file, and it contains all the member data and member functions needed to build the GUI interface and implement the simulation and the file management services.

ii) ActiveState Class

The ActiveState Class is responsible to keep track of all the GUI states and events such as setting or resting certain flags based on specific GUI events. Also it saves all the graphical images needed to display the GUI graphics such as panel and button images.

iii) Components Class

The Components Class holds all the circuit components information such as a component name and its attributes. It has several member functions that manipulate drawing the circuit components on the grid area. All component types extend the base Components Class. The class abstract functions are implemented in different component types, and polymorphism technique is used to specify the proper object function at the runtime.

iv) DrawTool Class

The DrawTool Class holds all the drawing text and shapes (circles, lines, rectangles) information. It has several methods that manipulate drawing these components on the grid area.

v) ServerApplication Class

The ServerApplication class is considered the background server application, which implements two main services: the simulation service and the file management service. Java Socket programming is used to establish the connection between client and server. The server creates a new socket and waits for incoming connections. Once the client accepts the connection, transferring information will be started between client and server. Every connection has a specific socket port number. Multi-threading technique is utilized in the ServerApplication class to accommodate simultaneous client connections at the same time. The simulation and the file management service each have a main thread that runs separately from the main server application for effective time processing and to accommodate any new client connections.

CHAPTER FIVE: TESTING AND VERIFICATION

From the software engineering point of view, testing and verification is a continuous process in any software life cycle, and it is an integral part of the implementation and integration phase. The current prototype is fully tested by visual inspections and results comparisons. The "Black Box" testing technique has been used for acceptance testing. Testing is carried out by the following methods:

5.1 Functional Testing

Functional testing was performed on each phase of the development cycle. It tests the functionality of the system and it is compliant to the system requirements and specifications. At the integration and implementation phase, each class and function in the program is tested against expected errors. The testing results are provided by the functional feature tracking report.

5.2 Functional Feature Tracking Report

Feature ID Prefix Appreviations:

GUI: Graphical User Interface

SCH: Schematic Capture

FMS: File Management Service

SMS: Simulation Service

Table 1: GUI Test Plan

ID	NAME	DESCRIPTION	STATUS
GUI.001	GUI Frame Layout	The whole GUI frame, size and position, panels, borders, etc.	PASS
GUI.002	Drawing Tools Panel	Button images and their event handler	PASS
GUI.003	File Management Panel	Button images, and their event handler	PASS
GUI.004	Simulation Panel	Button images, and their even handler	PASS

Table 2: Schematic Capture Test Plan

ID	NAME	DESCRIPTION	STATUS
SCH.001	Drawing Grid Area	Draw the grid image in the drawing area	PASS

ID	NAME	DESCRIPTION	STATUS
SCH.002	Drawing Circuit Components	Draw all the circuit components in the drawing area	PASS
SCH.003	Drawing Paint Tools	Draw all the paint tools in the drawing area	PASS
SCH.004	Scroll Bars	Scroll the grid area horizontally and vertically	PASS
SCH.005	Component Drawing	Draw a component	PASS
SCH.006	Connection Line Drawing	Draw a connection line	PASS
SCH.007	Component Placing	Place a component	PASS
SCH.008	Connection Line Placing	Place a connection line	PASS
SCH.009	Component Moving	Drag a component to move it	PASS
SCH.010	Connection Line Moving	Drag a connection line to move it	PASS
SCH.011	Connection Line Resizing	Drag a connection line’s end point to resize it	PASS
SCH.012	Component Rotating	Rotate a component	PASS

ID	NAME	DESCRIPTION	STATUS
SCH.013	Dependent Sources Configuration	Edit Dependent Sources attributes and check if the controller will be placed after the input window is closed	PASS
SCH.014	Popup Menus	Display all Popup Menus for components and grid	PASS
SCH.015	Component Deleting	Delete a component or a group of components	PASS
SCH.016	Component Copying	Copy a Component or a group of Components	PASS
SCH.017	Component Cutting	Cut a Component or a group of Components	PASS
SCH.018	Component Pasting	Paste a Component or a group of Component	PASS
SCH.019	Undo Actions	Undo pervious Actions	PASS
SCH.020	Schematic Clearing	Delete All Components	PASS
SCH.021	Connection Line Splitting	Split A Line that is connected to another Line at the connection point	PASS

ID	NAME	DESCRIPTION	STATUS
SCH.022	Connection Line Selection	When the connection node is selected, the line itself should NOT be selected	PASS
SCH.023	0-Length Connection	0-Connection Line is not allowed	PASS

Table 3: File Management Service Test Plan

ID	NAME	DESCRIPTION	STATUS
FMS.001	Schematic Data to Bit Stream	Schematic Data can be converted to bit stream without losses	PASS
FMS.002	Bit Stream to Schematic Data	Bit stream can be converted back to Schematic Data	PASS
FMS.003	Save Schematic Locally	Schematic Data can be saved by serializing the Components Data	PASS
FMS.004	Load Schematic Locally	Schematic file can be loaded to the current schematic page	PASS

ID	NAME	DESCRIPTION	STATUS
FMS.005	Save Schematic Remotely	Schematic Data can be saved in the remote server via socket interface	PASS
FMS.006	Load Schematic Remotely	Schematic file can be loaded from the remote server via socket interface	PASS
FMS.007	Delete Schematic File Remotely	Schematic file can be deleted from the remote server via socket interface	PASS
FMS.008	Make Directory in the Server	A directory can be made in the remote server via socket interface	PASS
FMS.009	Remove Directory from the Server	A directory can be removed from the remote server via socket interface	PASS

ID	NAME	DESCRIPTION	STATUS
FMS.010	Multi-Client Connection	Multi -client connections can be accommodated at the same time	PASS

Table 4: Simulation Service Test Plan

ID	NAME	DESCRIPTION	STATUS
SMS.001	Netlist Generation	Generate the Netlist from the current schematic data	PASS
SMS.002	Netlist Case 1	Netlist should be correct for schematic containing all component and connection lines.	PASS
SMS.003	Netlist Case 2	Netlist should be correct for schematics containing all components but connection lines	PASS

ID	NAME	DESCRIPTION	STATUS
SMS.004	Netlist Case 3	Netlist should be correct for schematics containing all components, connection lines and one ground	PASS
SMS.005	Netlist Case 4	Netlist should be correct for schematics containing components connected directly without connection lines	PASS
SMS.006	Netlist Case 5	Netlist should be correct for schematics containing components connected directly to one or more grounds without connection line	PASS
SMS.007	Netlist Case 6	Netlist should be correct for schematic matched dependent sources.	PASS

ID	NAME	DESCRIPTION	STATUS
SMS.008	Short Circuit Loop	Netlist should be correct for schematics containing short circuit loop	PASS
SMS.009	Short Circuit loop of Connection Lines	Netlist should be correct for schematics containing short circuit only with connection lines	PASS
SMS.010	Short Circuit loop of Grounds	Netlist should be correct for schematics containing short circuit created by floating grounds	PASS
SMS.011	Netlist for More than One Component Type	Netlist should be correct for a type that has more than one component	PASS
SMS.012	Netlist After Deleting Components	Netlist should be correct after deleting components	PASS

ID	NAME	DESCRIPTION	STATUS
SMS.013	Netlist After Deleting and Adding Components	Netlist should be correct after deleting and then adding components	PASS
SMS.014	Netlist After Schematic is Moved or Rotated but not Reconnected	Netlist should be correct after moving, rotating and reconnecting	PASS
SMS.015	Schematic Loaded After from the Server	Netlist should be correct after schematic is loaded form server with all the previous cases	PASS
SMS.016	Netlist Generator Doesn't Affect the Current Schematic	Netlist generator should not affect the existing schematic	PASS
SMS.017	Ground Node Manipulation	All ground nodes must have the same net name	PASS
SMS.018	Dependent Sources Consistency Check	Every dependent source should have the corresponding controllers	PASS

ID	NAME	DESCRIPTION	STATUS
SMS.019	Netlist Sending and Receiving	Netlist can be sent and received via network using socket interface	PASS
SMS.020	PSPICE Invoking	Invoke PSPICE process in the server	PASS
SMS.021	Netlist Parser	Netlist parser converts the Netlist to CIRML format	PASS

5.3 Netlist Testing

Examining the Netlist is a very important step prior to the simulation phase. The Netlist holds the current circuit description (circuit topology, components attributes, etc.). The Netlist generator was tested against several circuit topologies, and several results showed its correctness as shown in the next section.

5.4 Netlist Testing Examples

The following two examples demonstrate two arbitrary circuit designs, and the generated Netlist is shown also. The output and the expected results validate the correctness of the Netlist generator.

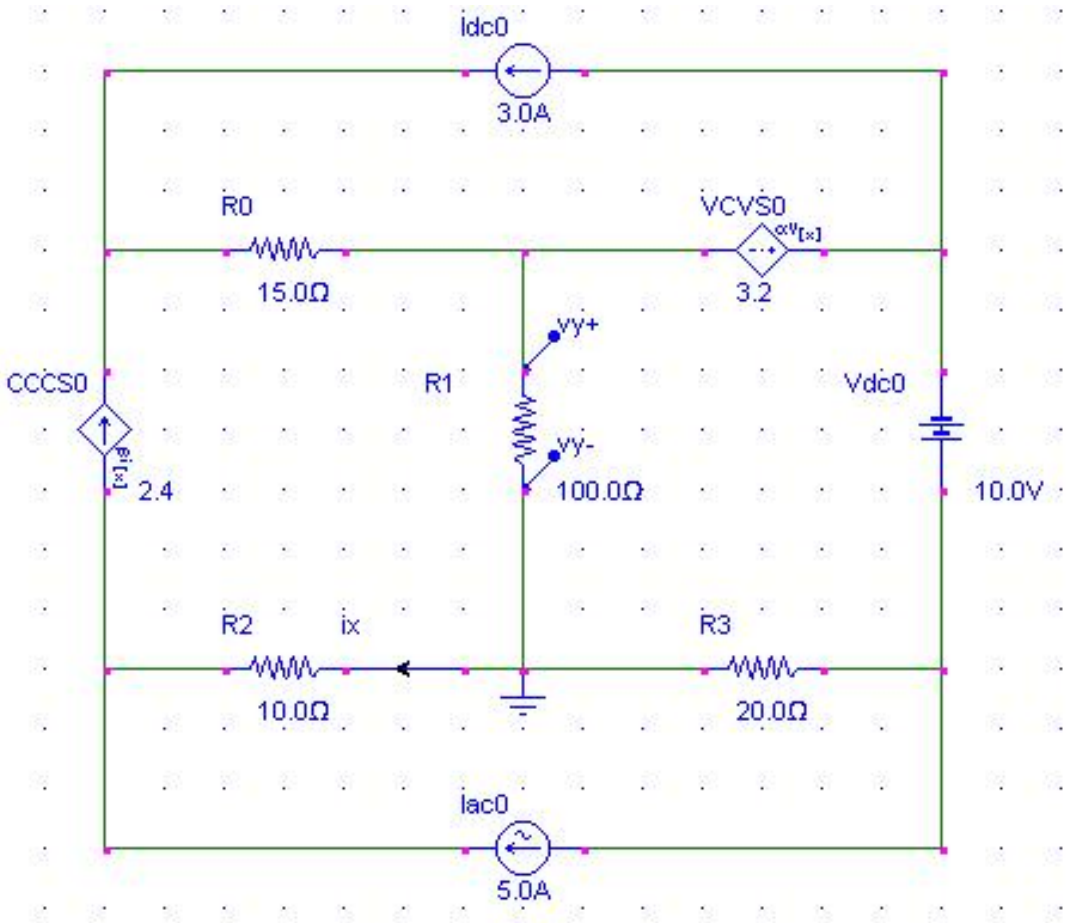


Figure 5-1: Example 1 Schematic

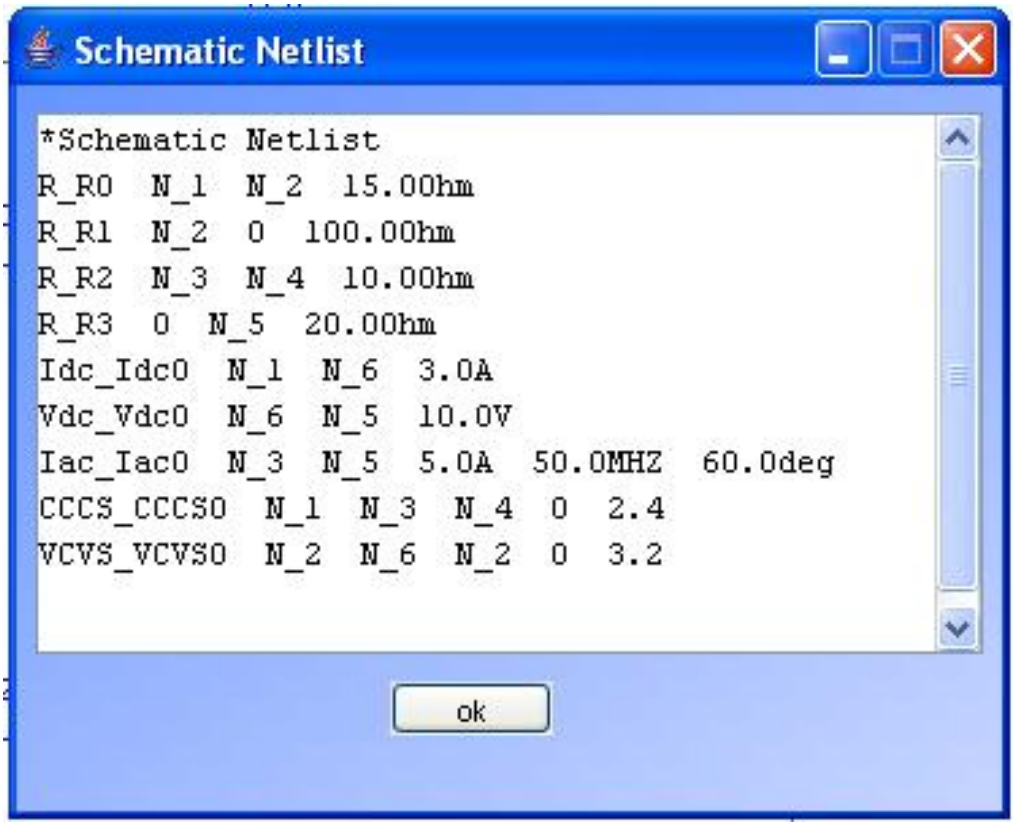


Figure 5-2: Example 1 Netlist Result

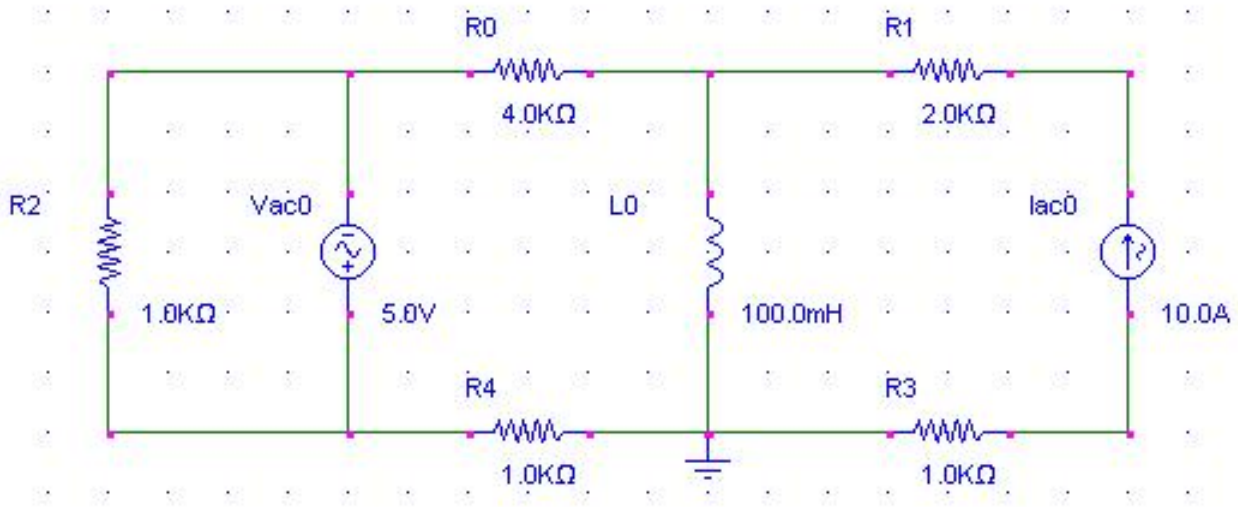


Figure 5-3: Example 2 Schematic

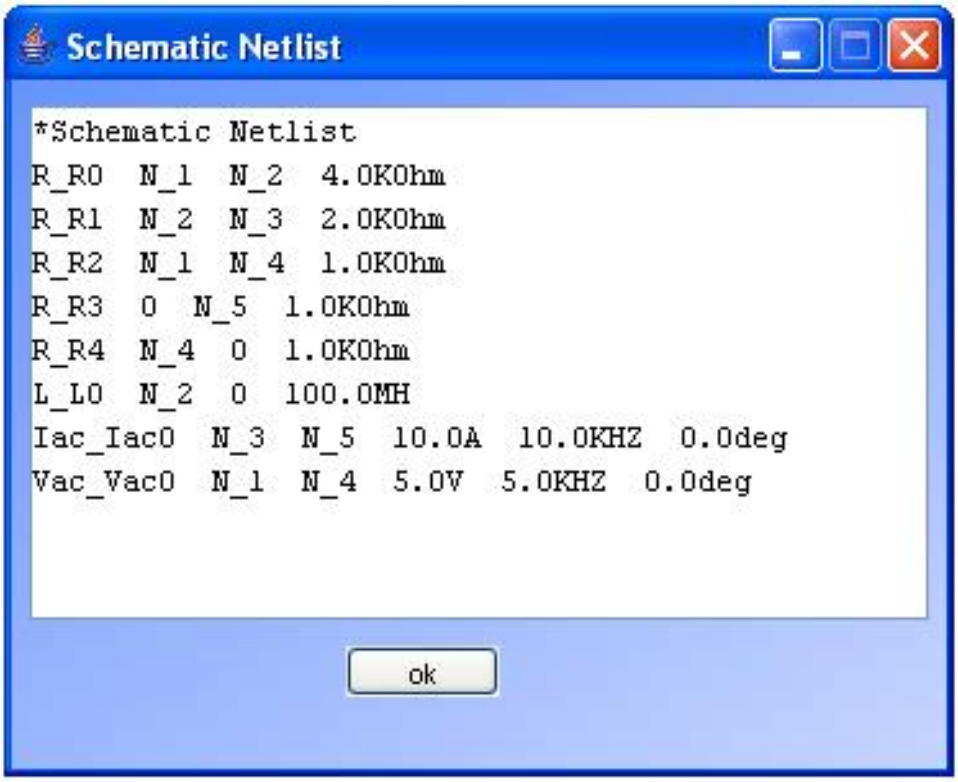


Figure 5-4: Example 2 Netlist Result

5.5 Browser Compatibility Testing

This test makes sure that the software applet works properly in different types of browsers over different platforms. The applets basically should work properly in the two main browsers: Netscape Communicator and Internet Explorer. The applet was tested as well as changes were made.

CHAPTER SIX: FUTURE WORK AND IMPROVMENTS

The current prototype package presents the first phase of the project “Teaching Electrical Circuit Analysis Using Web-Based Simulation.” Future improvements and modifications could be made on the current prototype to promote it to the final system, and such improvements could be summarized as follows:

6.1 GUI Schematic

More features and functionalities could be added to the current prototype, and such improvements could be summarized as follows:

6.1.1 Component Rotation

The current prototype gives the user the flexibility to rotate the components at right angles. Further modifications can be made to implement flexible component rotation at different angles.

6.1.2 Component Resizing

Actual component resizing is not implemented in the current prototype. A new drawing technique could be suggested to give more flexibility to display components other than using fixed images.

6.1.3 Component Display

The current prototype has a basic display technique to draw circuit components. More advanced display options can be added such as changing the color of the component and hiding and showing its attributes.

6.2 Data Structure and Components Library

6.2.1 Data Structure

Little risk has been taken to improve the existing data structure in the current prototype in order to maintain the back compatibility of the previous work. Radical and substantial changes could be made at the current data structure, which would give more flexibility to add new functionalities and to improve the maintainability of the current code.

6.2.2 Components Libraries

Most commercial software today defines its components in one or more predefined libraries rather that using hard-coded components description, which gives more flexibility to add, modify and delete components. Improvements could be made to implement a flexible and open component library system with more user privileges.

6.3 Simulation Analysis

6.3.1 Circuit Error Checker

More advanced circuit error checking techniques could be improved and added to avoid all the possible mistakes that the user might make in any circuit design.

6.3.2 Simulation Analysis

The current prototype supports only DC and AC circuit analysis. More circuit analysis types and simulation options could be added to the current simulation service. Also, a third party interface at the server should be improved to introduce fully PSPICE automation simulation service.

6.3.3 Circuit Simulation and Evaluation Service

Circuit simulation and evaluation service (CSAES) in the current prototype is client/server application. The actual simulation is carried out at the server with cooperation from the third party simulation package (PSPICE), which resides on the server. Radical changes on the simulation service will be made in the second phase of the project.The circuit simulation and evaluation service will be a self-evaluation service, and it will do the actual circuit analysis at the client without automating the simulation process to a third party software.

6.4 User File Managments System

A more advanced file management system could be built to provide more user privileges such as user quotas, user account and user security. These options should be considered the top priorities in any file management system.

CHAPTER SEVEN: CONCLUSION

The current prototype was implemented successfully in JAVA, which shows the feasibility of implementing a Web-based circuit simulation package with fully functional GUI interface and which gives the user the capability to build and simulate any basic circuit design. Two main client/server services are attached to the system: the Circuit Simulation and Evaluation Service and the File Management Service. The simulation service is responsible for generating the Netlist and parsing it to the CIRML format and running the simulation by sending the components data to the server and eventually geting the results back to the client for display after doing the actual simulation at the server. The file management service deals with all the file management manipulations to save and restore the schematic files either locally at the client or remotely at the server. DC and AC simulation analysis are introduced in this version. The current prototype can be considered as the first phase towards implementing the final Web-based simulation system.

APPENDIX A: USER MANUAL

1.1 System Requirments

On the Server side, PSPICE as a third party simulation package should be mounted, and the windows version is used.

On the client side, any Web browser such as Internet Explorer or Netscape Communicator running on PC with a java run time enabled (JRE) version 1.1 or above could be used to start the GUI applet.

1.2 Setting up the Software

The software package with all java classes and images should be installed in one directory and Web shared using a Web server. PSPICE should also be installed in the same directory. The location of PSPICE was hard coded, so if the PSPICE location is changed, the code should be updated correspondingly at the ServerApplication.java. The ServerApplication.java should be run at the server by typing the “java ServerApplication” DOS command at the current software directory. The server application is listening now at the background for any client requests.

Now the application is ready to use, and the front-end interface can be accessed from the Internet by connecting to the remote server through a Web browser. Since the application is mounted on a Web server, under wwwroot, it is very important that all software packages with their associated files should be located under the same directory.

1.3 How to run the Software

The front-end interface will be accessed eventually from the Internet by connecting to the remote Web server using a java enabled Web browser.

Once the mainApplet.java is loaded at the Web browser, all the GUI graphics are shown, and the user can now draw the schematic circuit and the Netlist (.net) and the CIRML format (.cir) are created. After that, the circuit information will be sent to the server to be processed by the server application and the third party simulation package (PSPICE). When simulation is done, the simulation results are written into ".out" and ".csd" files, which are sent back to the client for display.

APPENDIX B: HELP TOPICS

1.1 Introduction

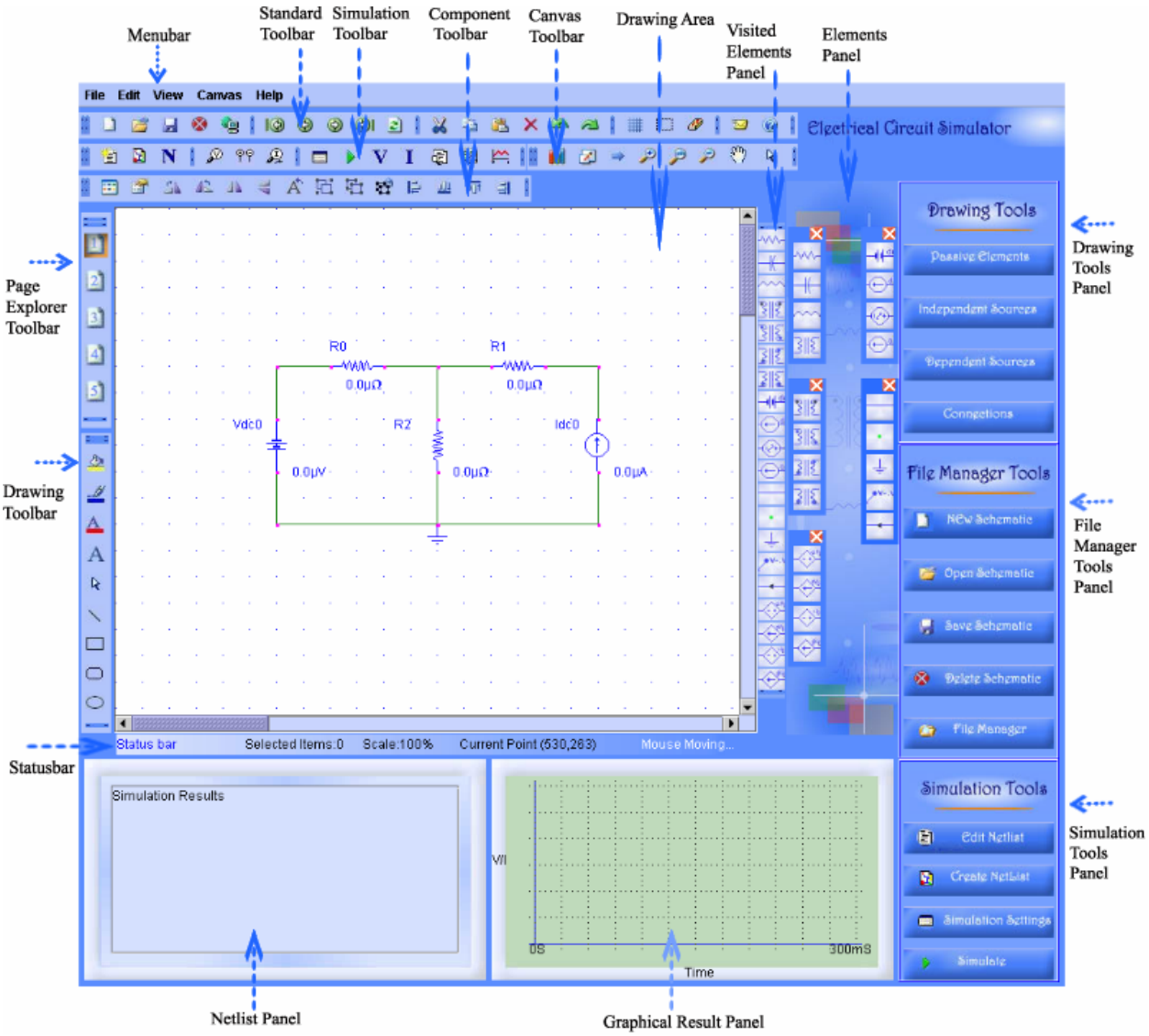
WELCOME TO THE

"WEB-BASED CIRCUIT DESIGN SIMULATION PACKAGE FOR SOLVING ELECTRICAL ENGINEERING CIRCUITS"

This online help topic guides the user through the process of generating and simulating a basic circuit. The evaluation and simulation service will be conducted by the PSPICE application tool.

1.2 GUI Overview

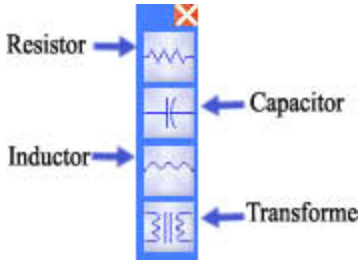
The GUI shows all the required graphics to build a schematic circuit such as images, buttons and panels as shown in the figure below.



1.3 List of Circuit Components

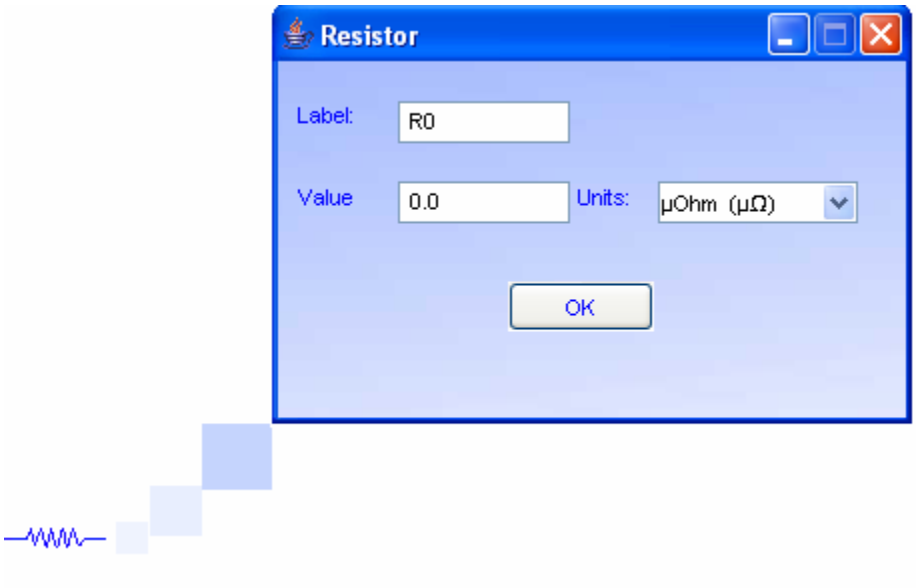
1.3.1 Passive Elements Group

The passive elements group contains four circuit components: Resistor, Capacitor, Inductor and Transformer. The passive elements panel is shown in the figure below.



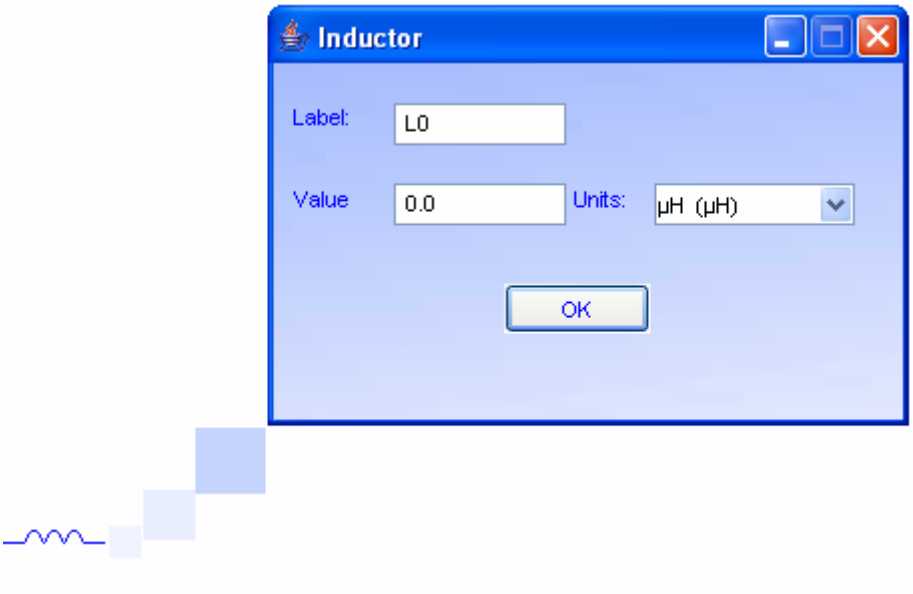
i) Resistor

This is a resistor, and the default value is µOhm. The value must be entered in the value field. The default label can be changed. The resistor symbol and its attributes are shown in the figure below.



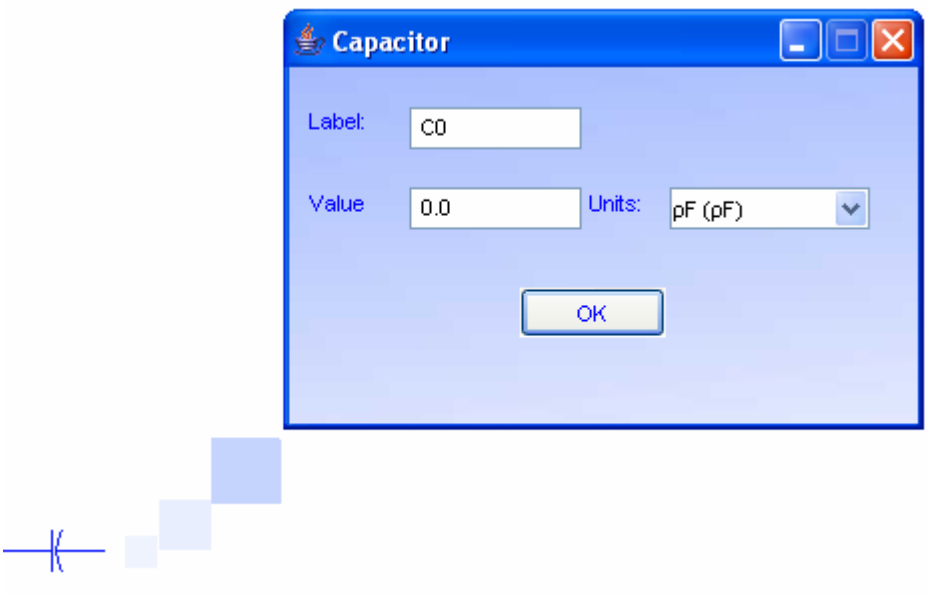
ii) Inductor

This is an inductor, and the default value is µH. The default label can be changed. The inductor symbol and its attributes are shown in the figure below.



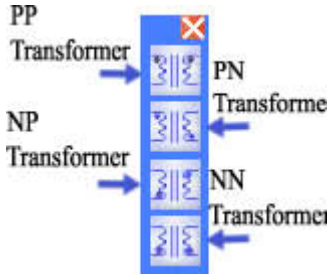
iii) Capacitor

This is a capacitor, and the default value is µF. The value must be entered in the value field. The default label can be changed. The capacitor symbol and its attributes are shown in the figure below.



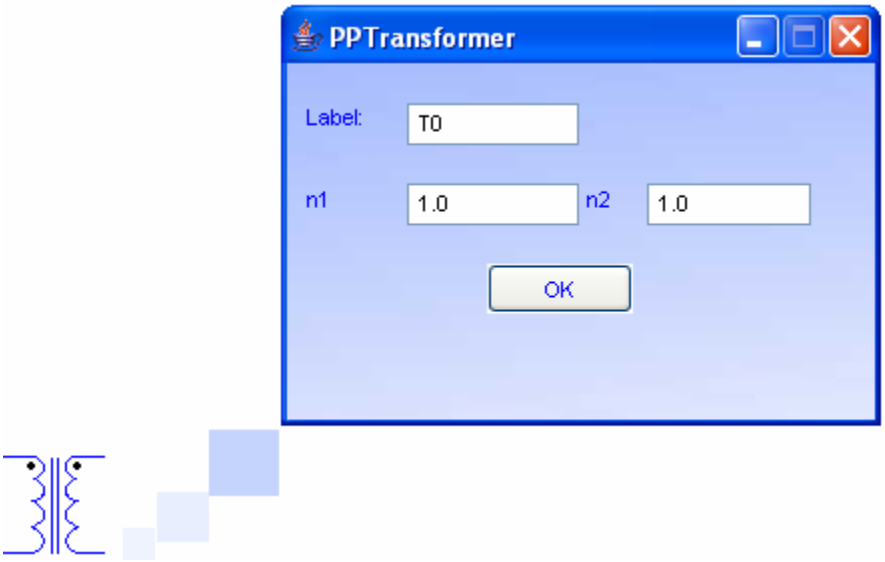
iv) Transformers Group

The transformers group contains four transformer types: Positive/Positive Transformer, Positive/Negative Transformer, Negative/Positive Transformer and Negative/Negative Transformer. The transformers panel is shown in the figure below.



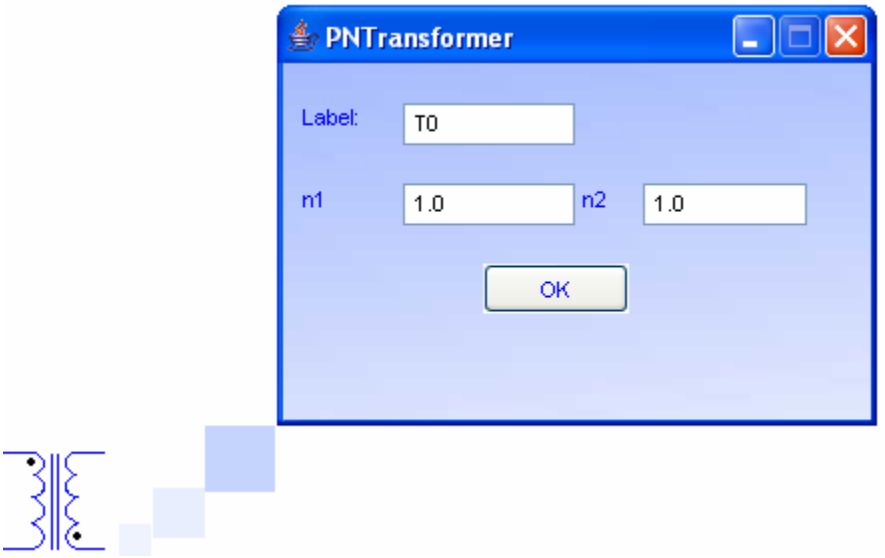
a) Positive/Positive Transformer

This is a Positive/Positive Transformer. The wending ratio and the label can be changed. The transformer symbol and its attributes are shown in the figure below.



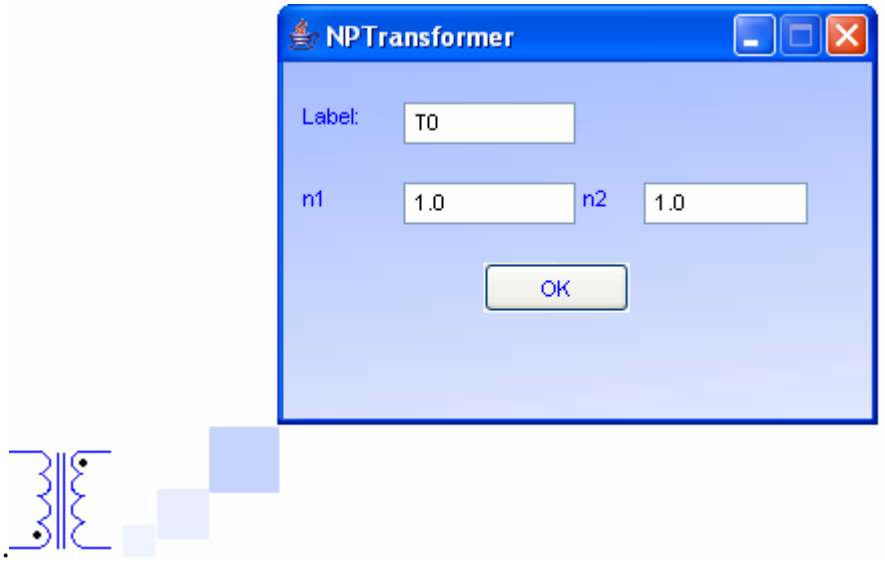
b) Positive/Negative Transformer

This is a Positive/Negative Transformer. The wending ratio and the label can be changed. The transformer symbol and its attributes are shown in the figure below.



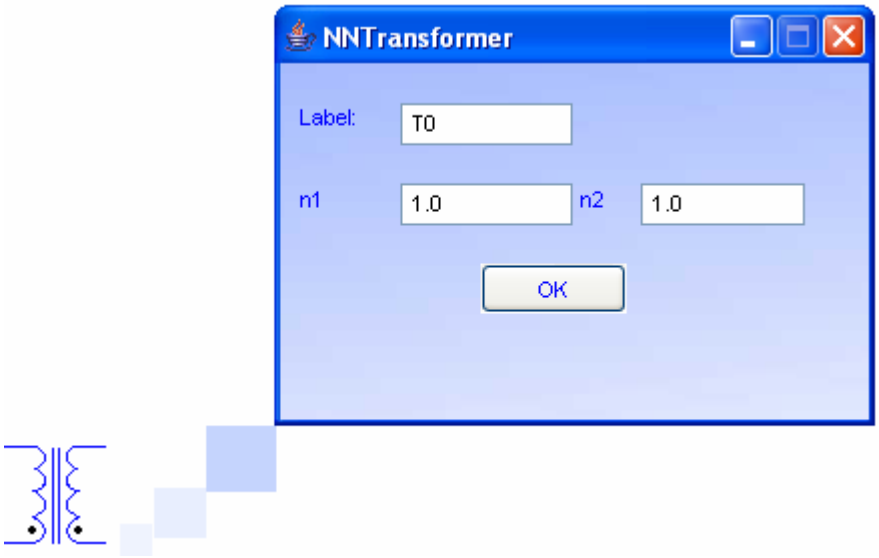
c) Negative/Positive Transformer

This is a Negative/Positive Transformer. The wending ratio and the label can be changed. The transformer symbol and its attributes are shown in the figure below.



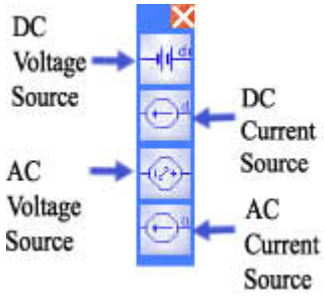
d) Negative/Negative Transformer

This is a Negative/Negative Transformer. The wending ratio and the label can be changed. The transformer symbol and its attributes are shown in the figure below.



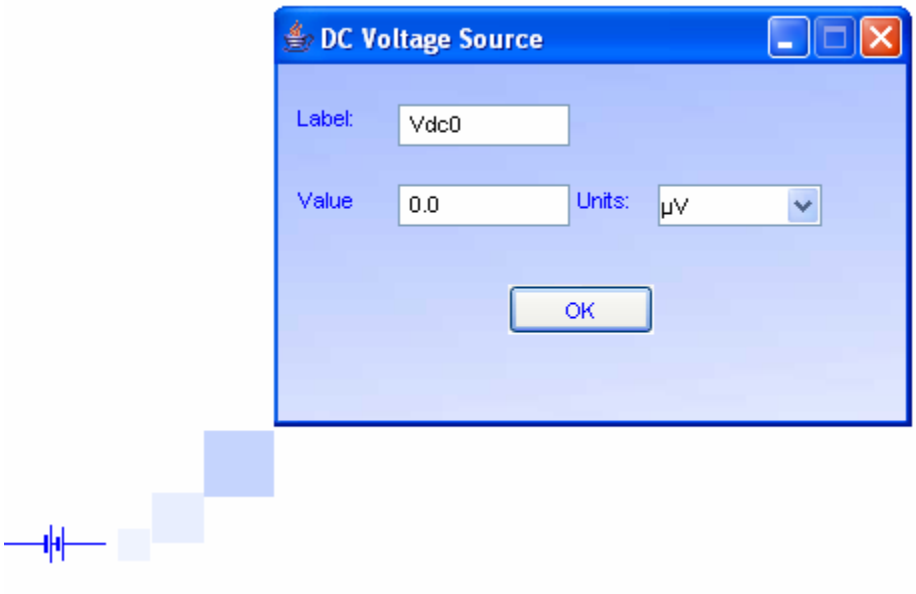
1.3.2 Independent Sources Group

The independent sources group contains four circuit elements: the DC Voltage source, the DC Current Source, the AC Voltage Source and the AC Current Source. The independent sources group panel is shown in the figure below.



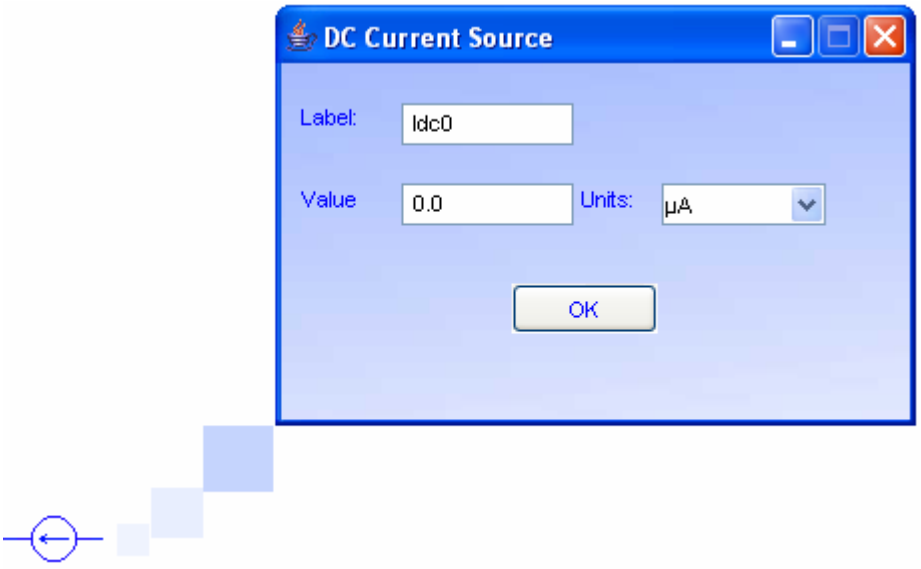
i) DC Voltage Source

This is a DC Voltage Source. The default value is V. The value must be entered in the value field. The default label can be changed. The DC Voltage Symbol and its attributes are shown in the figure below.



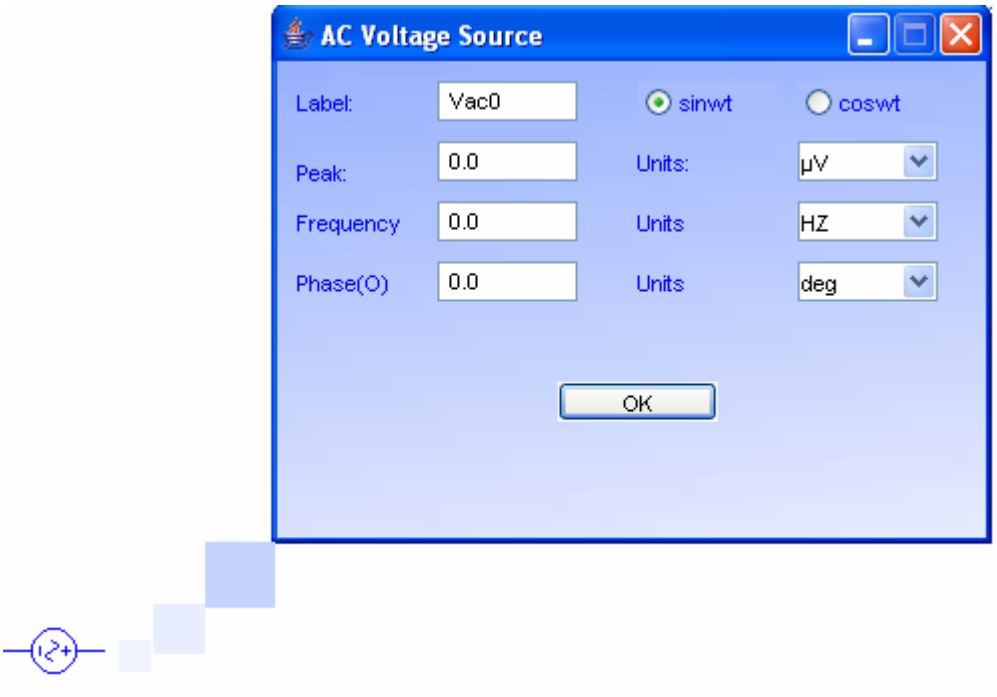
ii) DC Current Source

This is a DC Current Source. The default value is A. The value must be entered in the value field. The default label can be changed. The DC Current Source symbol and its attributes are shown in the figure below.



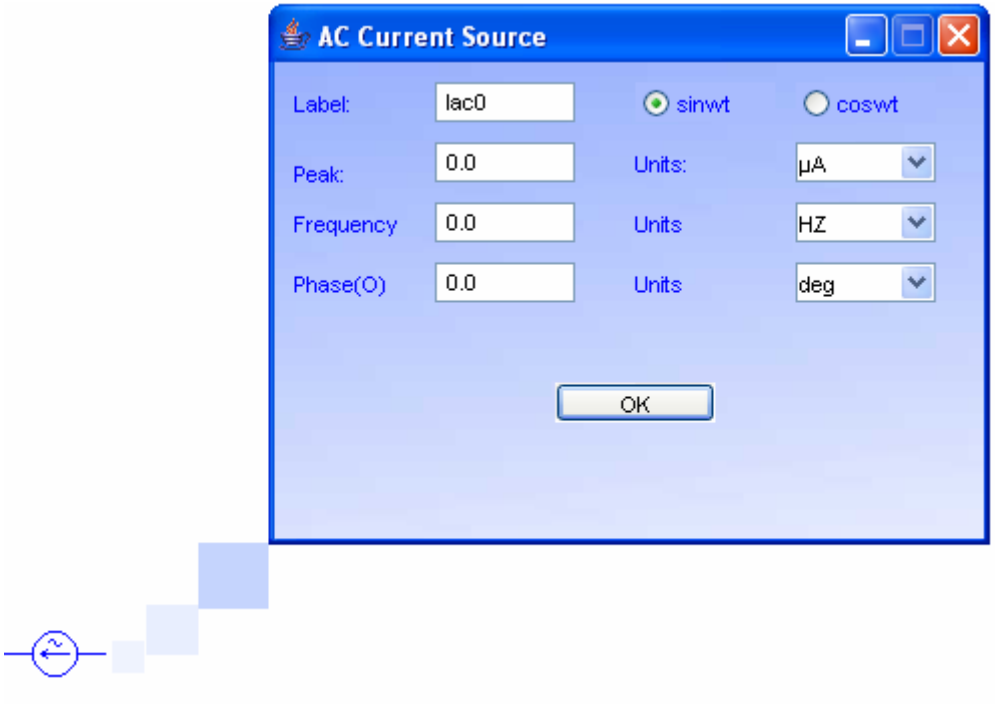
iii) AC Voltage Source

This is an AC Voltage Source. The default value is V. The peak, frequency and phase values have to be entered. The default label can be changed. The AC Voltage Source symbol and its attributes are shown in the figure below.



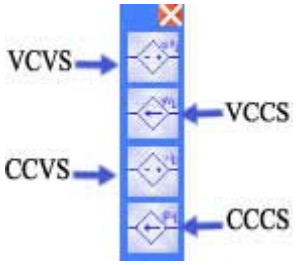
iv) AC Current Source

This is an AC Voltage Source. The default value is A. The peak, frequency and phase values have to be entered. The default label can be changed. The AC current source symbol and its attributes are shown in the figure below.



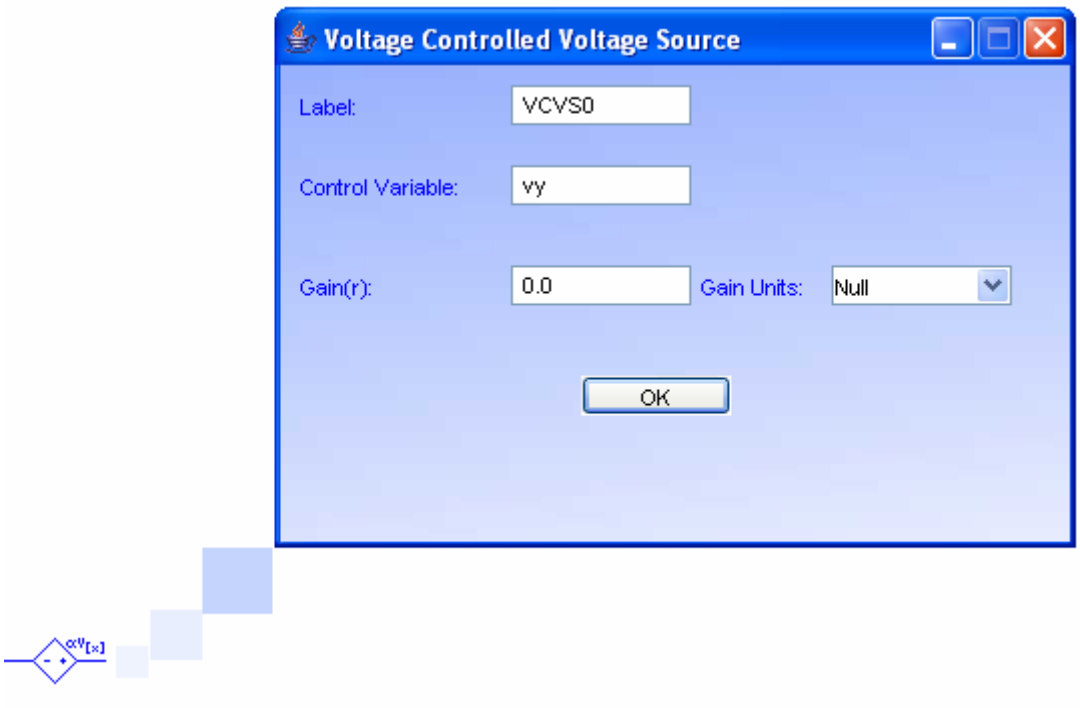
1.3.3 Dependent Sources Group

The dependent sources group contains four circuit elements: the Voltage Controlled Voltage Source, the Voltage Controlled Current Source, the Current Controlled Voltage Source and the Current Controlled Current Source. The independent sources group panel is shown in the figure below.



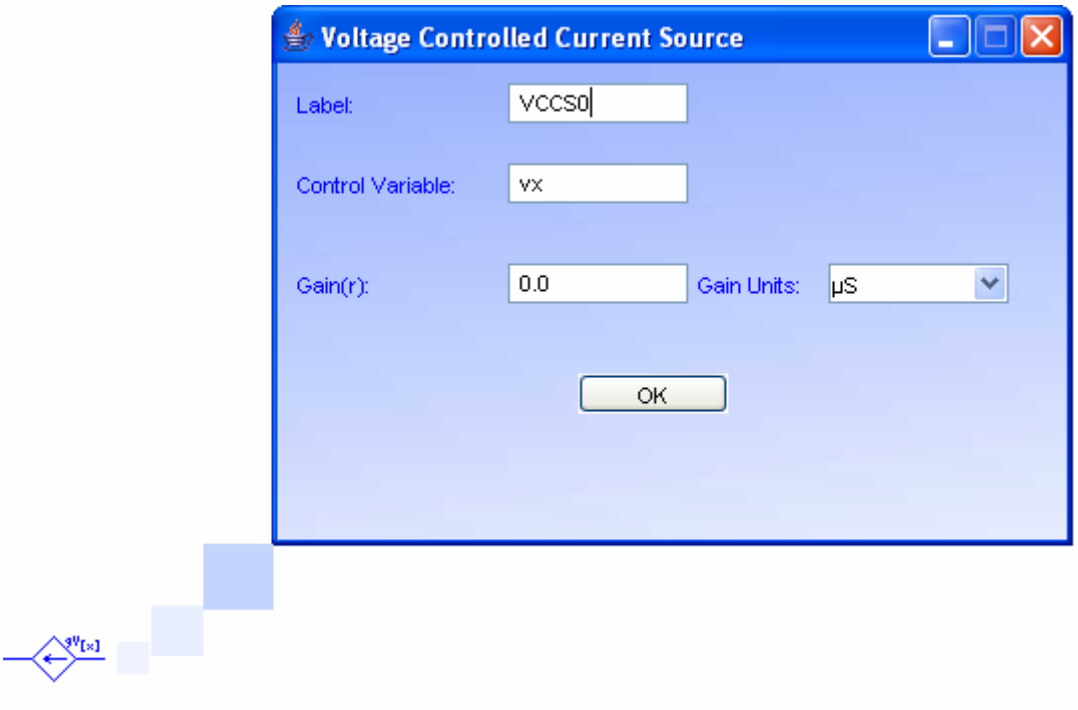
i) Voltage Controlled Voltage Source (VCVS)

This is a VCVS. The control variable and the gain have to be entered. The gain doesn't have a unit. The VCVS symbol and its attributes are shown in the figure below.



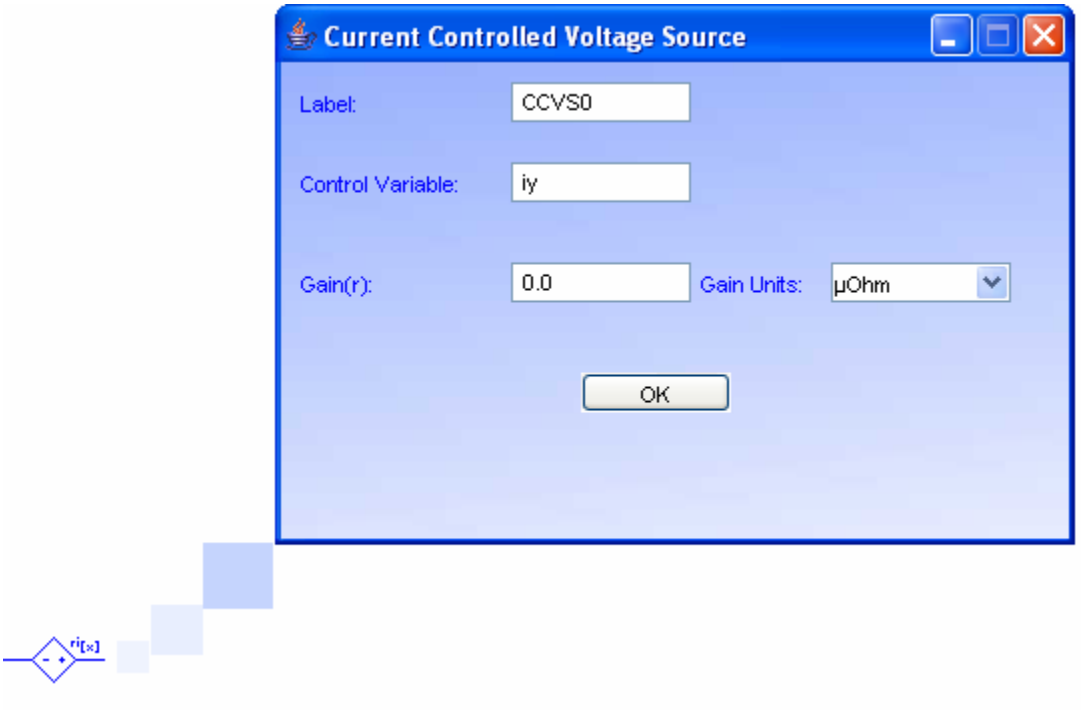
ii) Voltage Controlled Current Source (VCCS)

This is a VCCS. The control variable and the gain have to be entered. The gain has a Siemen unit. The VCCS symbol and its attributes are shown in the figure below.



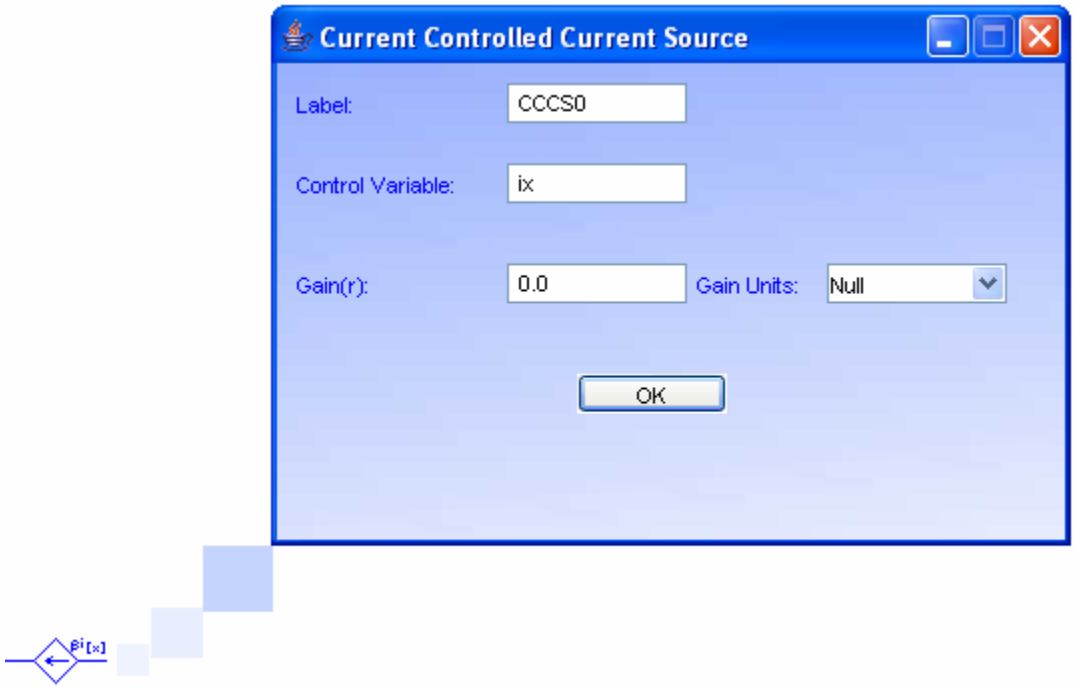
iii) Current Controlled Voltage Source (CCVS)

This is a CCVS. The control variable and the gain have to be entered. The gain has an Ohm unit. The CCVS and its attributes are shown in the figure below.



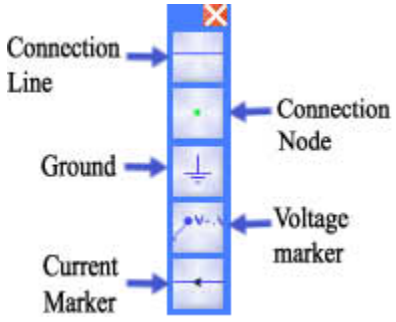
iv) Current Controlled Current Source (CCCS)

This is a CCCS. The control variable and the gain have to be entered. The gain doesn't have a unit. The CCCS and its attributes are shown in the figure below.



1.3.4 Connections Group

The connections group contains five elements: the connection line, the connection node, the ground, the voltage marker and the current marker. The connections group panel is shown in the figure below.

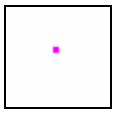


i) Connection Line

The connection line can be drawn to connect circuit components as a physical connection symbol. The connection line symbol is shown in the figure below.

ii) Connection Node

The connection node can be used to make electrically connected points in the schematic circuit. The connection node symbol is shown in the figure below.



iii) Ground

The ground represents the reference node for all components in the schematic. All voltages in the schematic refer to that reference node, which is considered by default as a zero voltage node. Any schematic circuit can have more than one ground node. The ground symbol is shown in the figure below.

iv) Voltages Marker

The voltage marker can specify the voltage difference between any node and the reference point. The voltage marker symbol is shown in the figure below.

v) Differential Voltages Nodes Marker

The voltage differential marker can specify the differential voltage between two nodes for VCVS and VCCS. The differential marker should be exist to determine the controlling voltage source. The differential voltage nodes marker symbol is shown in the figure below.

vi) Current Marker

The current marker can specify the branch that carries the current with a specific label for CCVS and CCCS. The current branch should be exist to determine the controlling current source. The current marker symbol is shown in the figure below.

1.4 How To …

1.4.1 Components Manipulation

i) Placing Components

How to Place Components

To place a component:

1. From the elements panel, click on the component image button.
2. Drop the component image by clicking on the drawing area.

or

1. From the elements panel, drag the component image to the drawing area.

2. Drop the component by releasing the mouse on the drawing area.

Tip: The same visited components can also be placed by the same way from the visited elements panel.

ii) Selecting Components

How to Select Components

To Select a Single Component:

Click on the component image at the drawing area. The component and its attributes will be displayed in red color (Active State).

or

Drag the mouse on the drawing area that contains the component.

To Select a Group of Components:

Drag the mouse on the drawing area that contains the group of components.

To Select All Components:

Click on "Select All" icon  at the standard toolbar, and all components will become active and will be displayed in the color red.

Note: "**CTRL+A"** can be used also to select all components.

iii) Editing Components Attributes

How to Edit Components Attributes

To Edit Component Attributes:

Double click on the component itself.

or

1. Select the Component.
2. Right click on the drawing area, and a popup menu will be displayed.
3. Choose the edit properties action from the popup menu.

or

Double Click on the shown component's attribute (name and value).

or

Click on the Edit Properties icon  from the Component Toolbar.

Tip: When an editing action is performed, an input user window will be displayed showing the component attributes so the user can change them.

Note: The "F2" key can be pressed also to do the same action.

iv) Changing Components Display

How to Change Components Display

To Change Component Display:

1. Select the component.
2. Right click on the drawing area, and a popup menu will be displayed.
3. Choose display properties action from the popup menu.

or

Click on the Display Properties icon  from the Component Toolbar.

Tip: When display properties action is performed, an input user window will be displayed showing the component display attributes so the user can change them.

Note: The "F3" key can be pressed also to do the same action.

v) Rotating Components

How to Rotate Components

To Rotate a Component or Group of Components:

1. Select the component of the group of components.
2. Right click on the drawing area, and a popup menu will be displayed.
3. Choose either the “Rotate Right” or “Rotate Left” action.

or

Click on either Rotate right  or Rotate Left  icon from the Component Toolbar.

Note: **“CTRL+R**” can be pressed to rotate the component to the right by 90^{o}.
“CTRL+L” can be pressed to rotate the component to the left by 90^{o}.

vi) Flipping Components

How to Flip Components

To Flip a Component or Group of Components Either Vertically or Horizontally:

1. Select the component of the group of components.
2. Right click on the drawing area, and a popup menu will be displayed.
3. Choose either the “Flip Vertical” or “Flip Horizontal” action.

or

Click on either Flip Vertical  or Flip Horizontal  icon from the Component Toolbar.

Note: **“CTRL+SHIFT+V**” can be pressed to flip the component vertically.
“CTRL+SHIFT+H” can be pressed to flip the component horizontally.

vii) Aligning Components

How to Align Components

To Align a Group of Components:

1. Select the group of components.
2. Right click on the drawing area, and a popup menu will be displayed.
3. Choose either Align Top, Align Left, Align Bottom or Align Right actions.

or

Click on Align Right  , Align Left  , Align Top  and Align Bottom  icon from the Component Toolbar

Note: **"CTRL+SHIFT+R**" can be pressed to align components to the right
"CTRL+SHIFT+L" can be pressed to align components to the left
"CTRL+SHIFT+T" can be pressed to align components to the top
"CTRL+SHIFT+B" can be pressed to align components to the bottom

viii) Resizing Component

How to Resize Components

To Resize a Component by Drawing a Connection Line at the End Points:

1. Select the component.
2. Start dragging at one of the end points, and a connection line should appear that starts from the end point that has been pressed.

ix) Cutting Component

How to Cut Components

To Cut a Component or a Group of Components:

1. Select the component or the group of components.
2. Right click on the drawing area. A popup menu will be displayed.
3. Choose the Cut action from the displayed popup menu.

or

1. Select the component or the group of components
2. Click on the Cut action  from the standard toolbar

Note: CTRL+X" can be pressed also to do the same action.

x) Copying Component

How to Copy Components

To Copy a Component or a Group of Components:

1. Select the component or the group of components.
2. Right click on the drawing area. A popup menu will be displayed.
3. Choose the Copy action from the displayed popup menu.

or

1. Select the component or the group of components.
2. Click on the Copy action  from the standard toolbar.

Note: "CTRL+C" can be pressed also to do the same action.

xi) Pasting Component

How to Paste Components

To Paste a Component or a Group of Components:

1. Right click on the drawing area. A popup menu will be displayed.
2. Choose the Paste action from the displayed popup menu.

or

Click on the Paste action  from the standard toolbar.

Note: "CTRL+V" can be pressed also to do the same action.

xii) Deleting Components

How to Delete Components

To Delete a Component or a Group of Components:

1. Select the component or the group of components.
2. Right click on the drawing area. A popup menu will be displayed.

3. Choose the Delete action from the displayed popup menu.

Or

1. Select the component or the group of components.
2. Click on the Delete action  from the standard toolbar.

Note: The "DEL" key can be pressed also to do the same action.

xiii) Grouping Components

How to Group Components

To Select a Group of Components:

1. Select the components.
2. Right click on the drawing area.
3. Click on the Group action from the displayed popup menu.

or

1. Select the components.
2. Click on the Group  icon from the Component Toolbar.

Note: "CTRL+G" can be pressed also to do the same action.

xiv) Ungrouping Components

How to Ungroup Components

To Ungroup Components:

1. Select the components.
2. Right click on the drawing area.
3. Choose the Ungroup action from the displayed popup menu.

or

1. Select the components.

2. Click on the Regroup  icon from the Component Toolbar.

Note: "CTRL+U" can be pressed also to do the same action.

.

xv) Regrouping Components

How to Regroup Components

To Regroup Components:

1. Select the components.
2. Right click on the drawing area.
3. Choose the "Regroup" action from the displayed popup menu.

or

1. Select the components.
2. Click on the Regroup  icon from the Component Toolbar.

Note: "CTRL+P" can be pressed also to do the same action.

xvi) Undoing/Redoing Actions

How to Undo/Redo Actions

To Undo/Redo GRID actions:

1. Right click on the drawing area.
2. Choose the Undo or Redo action from the displayed popup menu.

Or

Click on the Undo  or Redo  icon from the standard toolbar.

Note: "CTRL+Z" can be pressed also to do undo actions.
"CTRL+W" can be pressed also to do redo actions.

xvii) Clearing all Components

How to Clear All Components

To Clear all Components:

1. Right click on the drawing area, and a popup menu will be displayed.
2. Choose the Clear All action from the displayed popup menu.

or

1. Select all components.
2. Click on the Delete  icon from the standard toolbar or press DEL key.

1.4.2 Connection Line Manipulation

i) Drawing a Connection Line

How to Draw a Connection Line

To Draw a Connection Line:

1. Click on the connection line image button at the connections elements panel.
2. Press on the drawing area and start dragging to draw the line.

Tip: The start point is determined whenever you press the mouse, and the end point is determined when you release it.

ii) Moving a Connection Line

How to Move a Connection Line

To Move a Connection Line:

1. Select the line.
2. Press on any end point of the line and start dragging.

iii) Splitting a Connection Line

How to Split a Connection Line

To Split the Line:

1. Select the line.
2. Move it to the position where the other line's end point intersects with the first line.

.

1.4.3 Drawing Area Manipulation

i) Zooming In/Out on the Canvas

How to Zoom In/Out on the Canvas

To Zoom In or Zoom Out on the Drawing Area:

1. Right click on the drawing area.
2. Choose either Zoom in or Zoom out actions from the displayed popup menu.

or

Click on Zoom In or Zoom Out icon from the Standard Toolbar.

Note: Fit Zoom from the standard toolbar can be used to get the actual canvas size

ii) Resizing the Canvas

How to Resize the Canvas

To Resize the Drawing Area:

Click on the Canvas Size  icon from the canvas toolbar. A canvas properties window will be displayed.

iii) Changing the Canvas Colors

How to Change the Canvas Colors

To Change the Drawing Area Colors:

Click on the Canvas Colors  icon from the canvas toolbar. A canvas properties window will be displayed.

Note: The"F4" key can be pressed also to do the same action.

1.4.4 Simulation Manipulation

i) Editing Netlist

How to Edit the Netlist

To Edit the Netlist:

Click on the Edit Netlist  icon from the simulation toolbar.

or

Click on the Edit Netlist action button from the simulation tools panel.

Note: The "F6" key can be pressed also to do the same action.

ii) Creating Netlist

How to Create Netlist

To Create the Netlist:

Click on the Create Netlist  icon from the simulation toolbar.

or

Click on the Create Netlist action button from the simulation tools panel.

Note: The "F5" key can be pressed also to do the same action.

iii) Editing Simulation Settings

How to Edit Simulation Settings

To Edit the Simulation Settings:

Click on the Simulation Settings icon  from the simulation toolbar.

or

Click on the Simulation Settings action button from the simulation tools panel.

Note: The "F7" key can be pressed also to do the same action.

iv) Running Simulation

How to Run Simulation

To Run the Simulation:

Click on the Run  icon from the simulation toolbar.

or

Click on the Simulate action from the simulation tools panel.

Note: The "F9" key can be pressed also to do the same action.

v) Showing Nodes Net Names

How to Show the Nodes Net Names

To Show the Nodes Net Names:

Click on the Show Nodes  icon from the simulation toolbar.

vi) Showing Nodes Voltages

How to Show Nodes Voltages

To Show the Nodes Voltages:

Click on the Show Voltages  icon from the simulation toolbar.

vii) Showing Nodes Currents

How to Show Nodes Currents

To Show the Nodes Currents:

Click on the Show Currents  icon from the simulation toolbar.

viii) Placing Markers

How to Place Markers

To Place Markers:

Click on the circuit markers (Voltage  , Current  and Voltage Differential Marker ) from the simulation toolbar.

ix) Showing CIRML Output

How to Show the CIRML Output

To Show the CIRML Output:

Click on the Display CIRML  icon from the simulation toolbar.

Note: The "F8" key can be pressed also to do the same action.

x) Viewing Simulation Results

How to View the Simulation Results

To View the Simulation Output Results:

Click on the Simulation Output Result  icon from the simulation toolbar.

Note: The "F10" key can be pressed also to do the same action.

1.4.5 Schematic Pages Manipulation

i) Creating a New Page

How to Create a New Page

To Create a New Page:

Click on the New Schematic Page  icon from the standard toolbar.

or

Click on the New Schematic action button from the file manager tools. panel

Note: "CTRL+N" can be pressed also to do the same action.

ii) Saving a Schematic Page

How to Save a Schematic Page

To Save a Schematic Page:

Click on Save Schematic Page  icon from the standard toolbar.

or

Click on Save Schematic action button at the file manager tools panel.

Note: "CTRL+S" can be pressed also to do the same action.

iii) Loading a Schematic Page

How to Load a Schematic Page

To Load a Schematic Page:

Click on Load Schematic Page  icon from the standard toolbar.

or

Click on Open Schematic action button at the file manager tools panel.

Note: "CTRL+O" can be pressed also to do the same action

iv) Deleting a Schematic Page

How to Delete a Schematic Page

To Delete a Schematic Page:

Click on the Delete Current Schematic  icon from the standard toolbar

or

Click on the Delete Schematic action button from the file manager tools panel

Note: "ALT+F4" can be pressed also to do the same action.

v) Exploring Schematic Pages

How to Explore the Schematic Pages

To Explore the Schematic Pages:

Click on Last Backward  , Backward  , Forward  or Last Forward  icons from the standard toolbar.

or

Click on the page number action from the schematic pages toolbar.

vi) Connecting to the Remote File Manager Server

How to Connect to The Remote File Manager Server

To Connect to the Remote File Manager Server:

Click on the Server File Manager  icon from the standard toolbar.

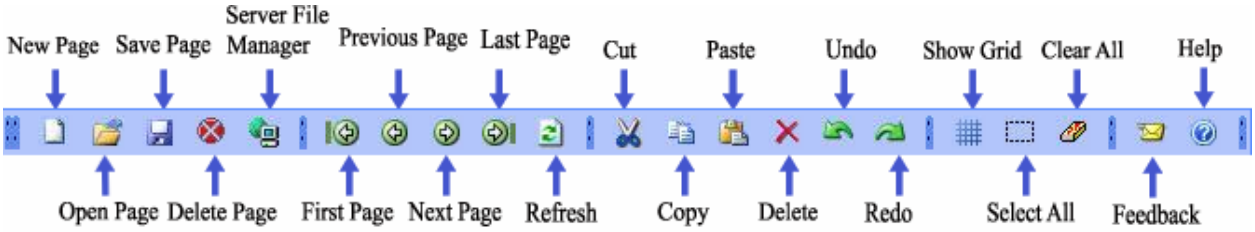
Or

Click on the Server File Manager action button from the file manager tools panel.

1.5 Quick Reference

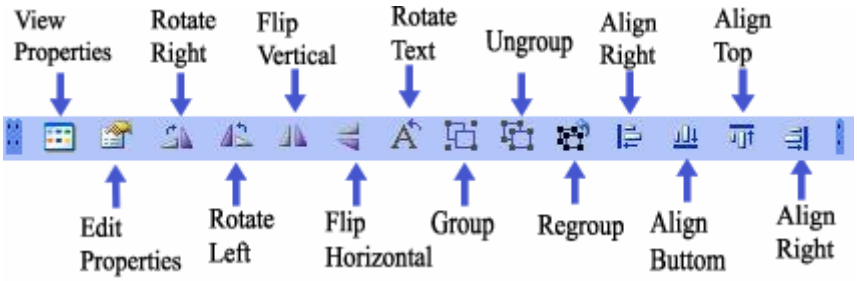
1.5.1 Toolbars

i) Standard Toolbar



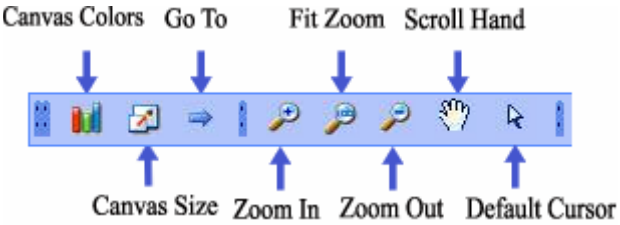
Icon	Action Description
	Create a new schematic page
	Open a schematic page
	Save a schematic page
	Close a schematic page
	Server File Manager
	Go to the first schematic page
	Go the previous schematic page
	Go to the next schematic page
	Go to the last schematic page
	Refresh the current schematic page
	Cut component
	Copy component
	Paste component
	Delete component
	Undo action
	Redo action
	Show/Hide GRID
	Select All Components
	Clear All Components
	Email us
	Help Topics

ii) Component Toolbar



Attributes	Description
	Edit the view component properties
	Edit the circuit component attributes
	Rotate the component to the right
	Rotate the component to the left
	Flip the component vertically
	Flip the component horizontally
	Rotate text
	Group components
	Ungroup components
	Regroup set of components
	Align components to the left
	Align components to the bottom
	Align components to the top
	Align components to the right

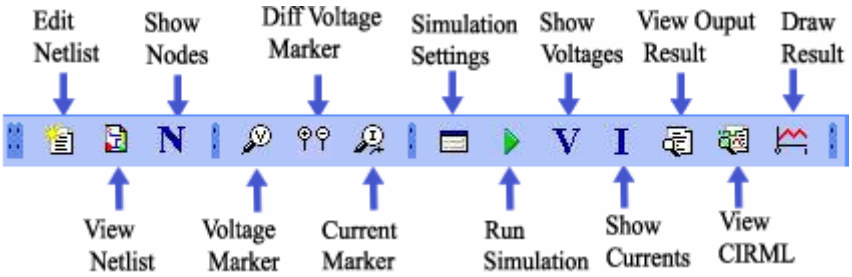
iii) Canvas Toolbar



Attributes	Description
	Change the background and foreground colors of the canvas
	Change the width and height of the canvas
	Go to a certain position in the canvas
	Zoom in the on canvas area

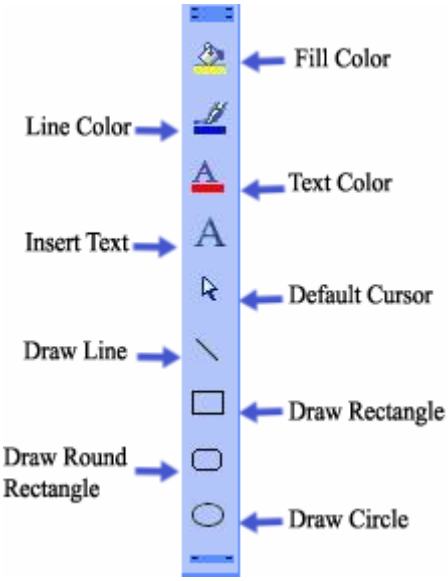
	Go to the actual zoom size
	Zoom out on the canvas area
	Scroll the canvas either vertically or horizontally
	Change the cursor to the default shape

iv) Simulation Toolbar



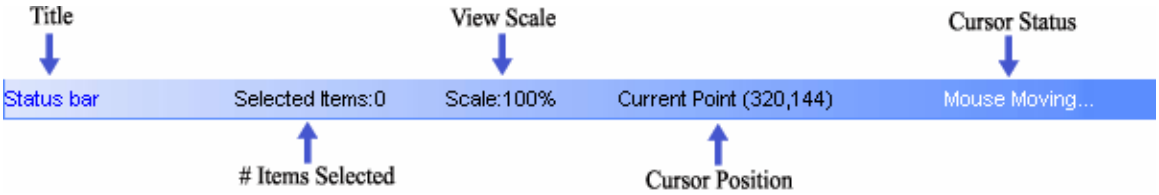
Attributes	Description
	Edit Schematic
	View Schematic
	Show Nodes Net Names
	Voltage Marker
	Differential Voltage Marker
	Current Marker
	Simulation Settings
	Run Simulation
	Show Nodes Voltages
	Show Nodes Currents
	View Simulation Results
	View CIRML Output
	Draw Result

v) Painting Toolbar



Attributes	Description
	Change the fill color of the drawing shape
	Change the line color of the drawing shape
	Change the text color of the drawing area
	Insert text at the drawing area
	Change to the default cursor
	Draw a line
	Draw a rectangle
	Draw a round rectangle
	Draw a circle

1.5.2 GUI Status Bar



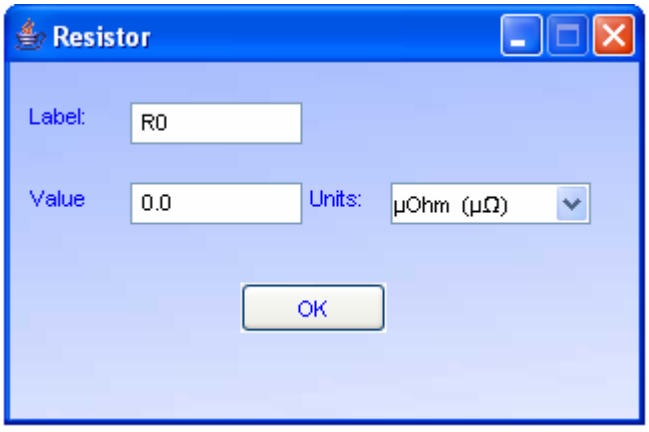
The GUI Status Bar displays the following GUI events:

Status	Description
Selected Items	It displays the number of active components
Scale	It displays the drawing area scale
Current Point	It displays the X and Y positions of the cursor
Mouse Events	It display the current mouse event such as pressing, dragging, clicking and releasing

1.5.3 GUI Dialog Windows

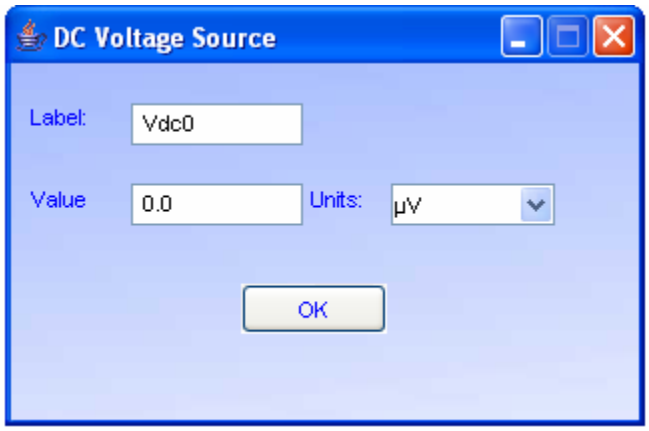
i) Passive Elements Input Window

The passive elements input window, as shown in the figure below, will be displayed when a passive component is edited such as Resistor, Capacitor and Inductor.



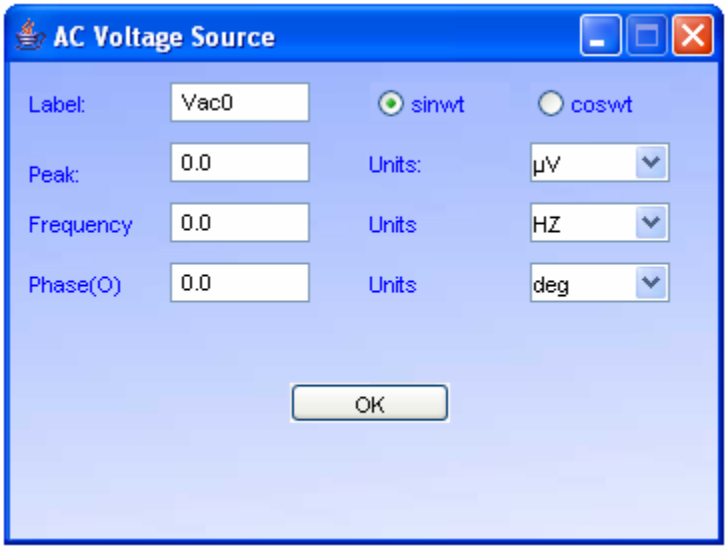
ii) Independent DC Sources Input Window

The Independent DC sources input window, as shown in the figure below, will be displayed when a DC source is edited such as DC Voltage source and DC Current Source.



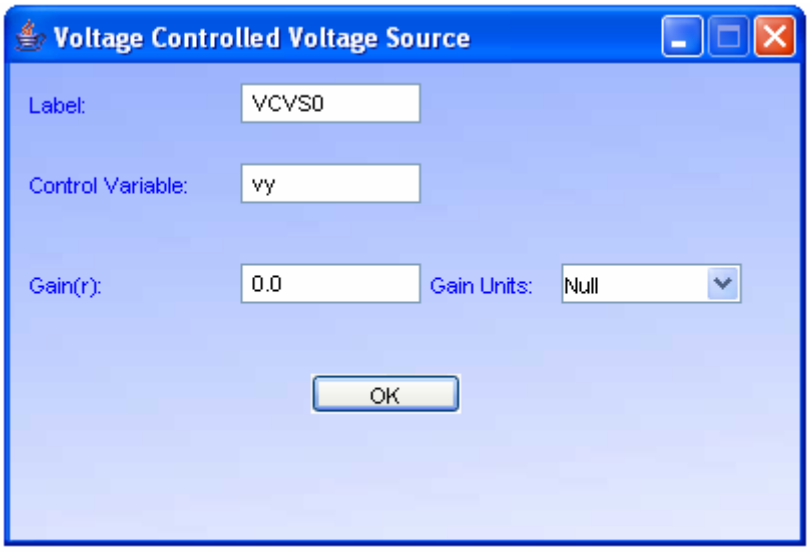
iii) Independent AC Sources Input Window

The Independent AC Sources Input Window, as shown in the figure below, will be displayed when an AC source is edited such as AC Voltage Source and AC Current Source.



iv) Dependent Sources Input Window

The Dependent Sources Input Window, as shown in the figure below, will be displayed when a dependent source is edited such as Voltage Controlled Voltage Source, Voltage Controlled Current Source, Current Controlled Voltage Source and Current Controlled Current Source.

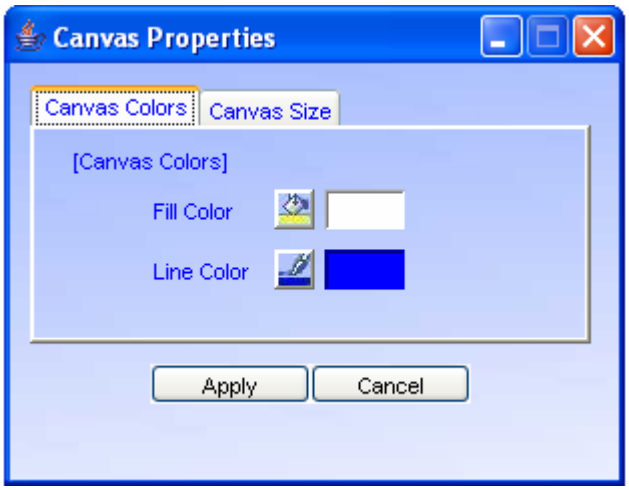


v) Current Marker Input Window

The Current Marker Input Window, as shown in the figure below, will be displayed when a current marker is edited.

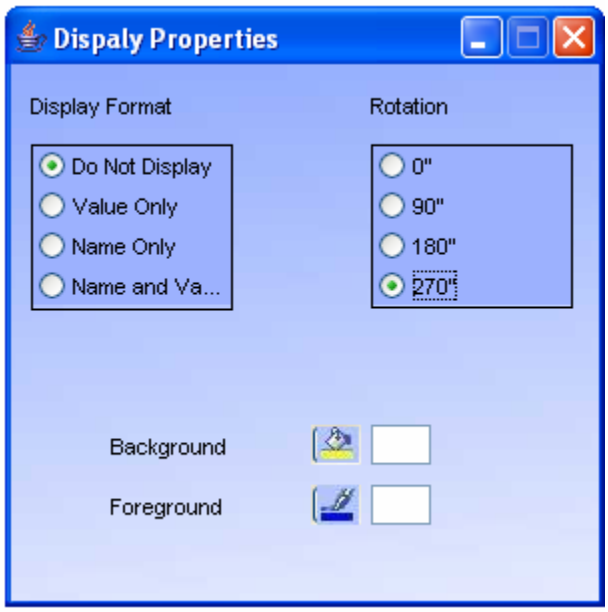
vi) Canvas Properties Dialog

The Canvas Properties Dialog, as shown in the figure below, will be displayed when you choose the canvas colors and the canvas size actions.



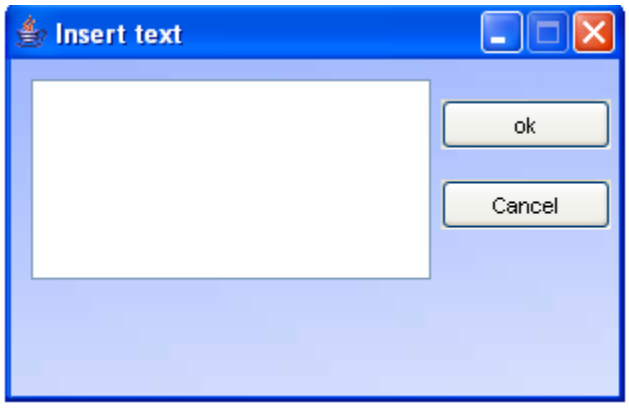
vii) Display Properties Dialog

The Display Properties Dialog, as shown in the figure below, will be displayed when you choose the display properties action.



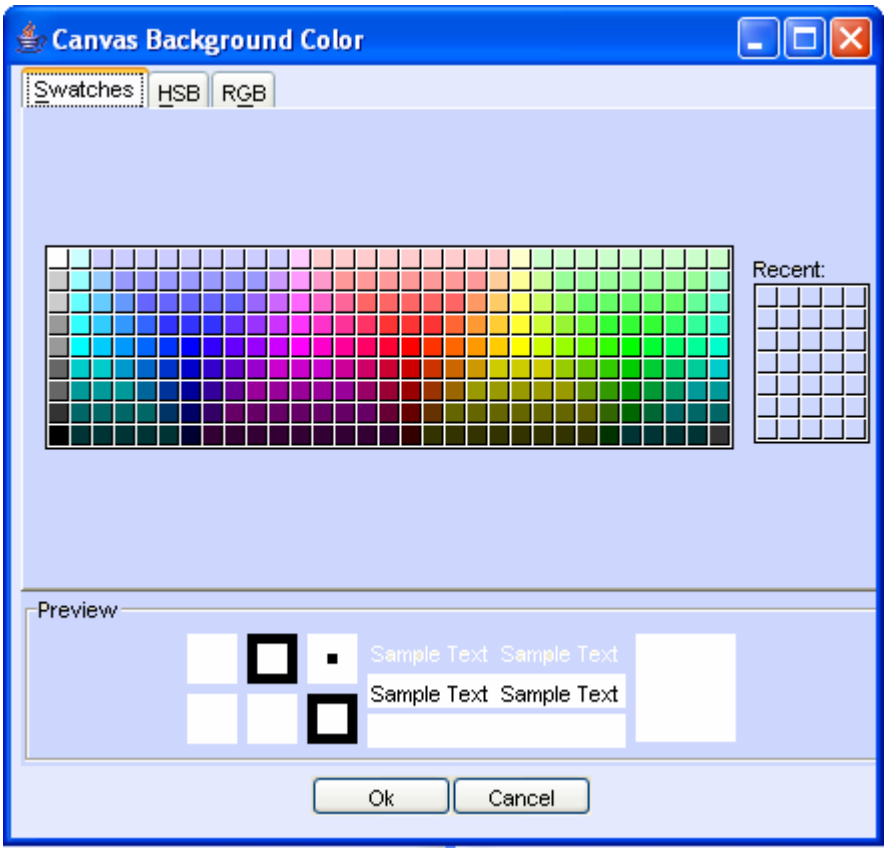
viii) Insert Text Dialog

The Insert Text Dialog, as shown in the figure below, will be displayed when you choose the insert text action.



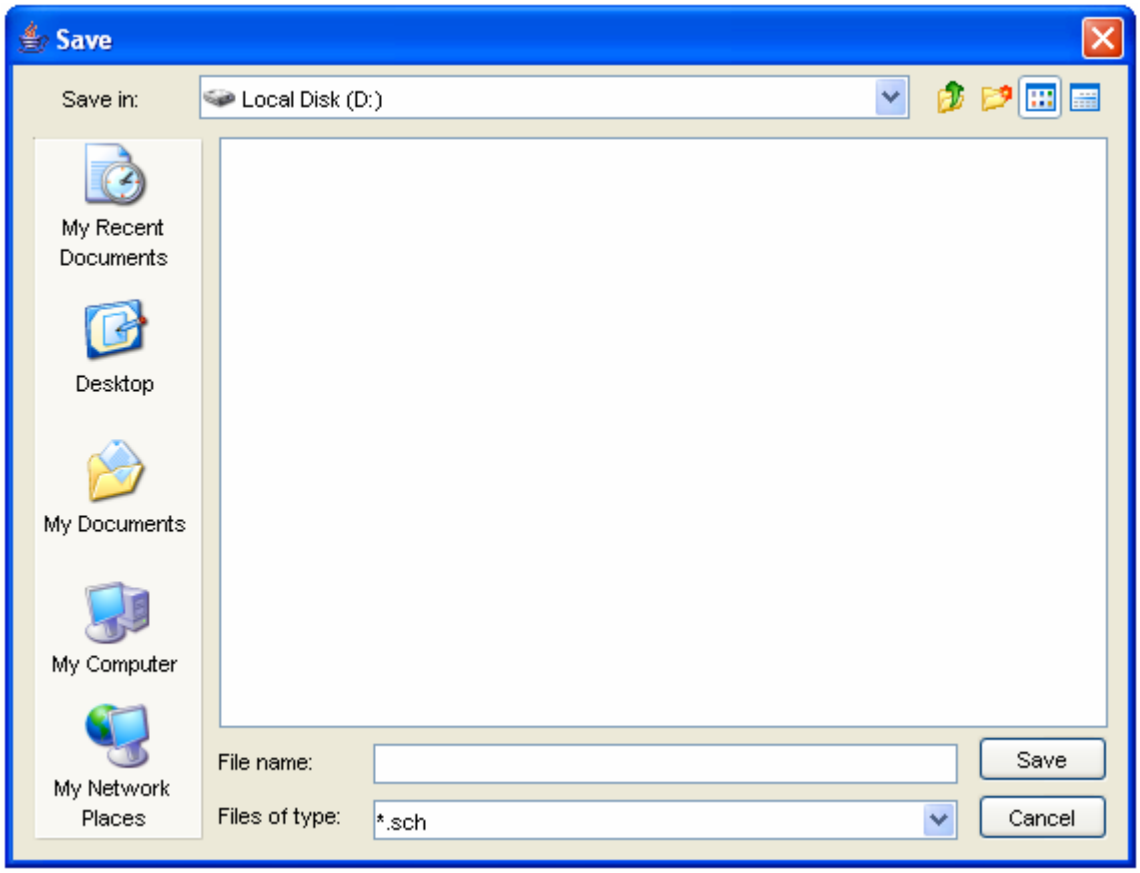
ix) Color Chooser Dialog

The Color Chooser Dialog, as shown in the figure below, will be displayed when you choose the Fill Color, Line Color, Text Color, Canvas Background Color and the Canvas Foreground Color actions.



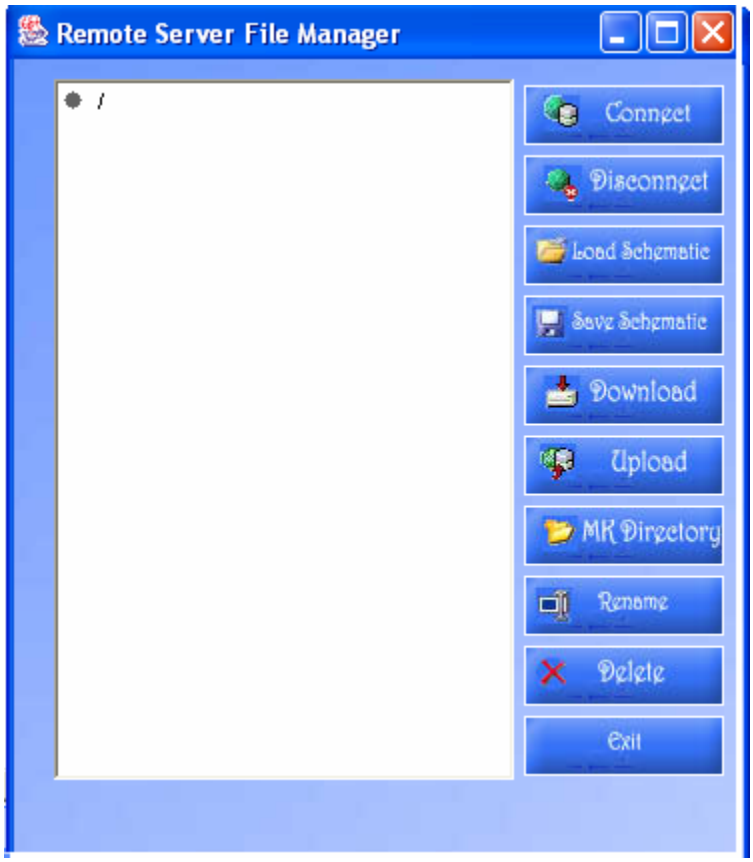
x) File Chooser Dialog

The File Chooser Dialog, as shown in the figure below, will be displayed when you choose the open and save actions.



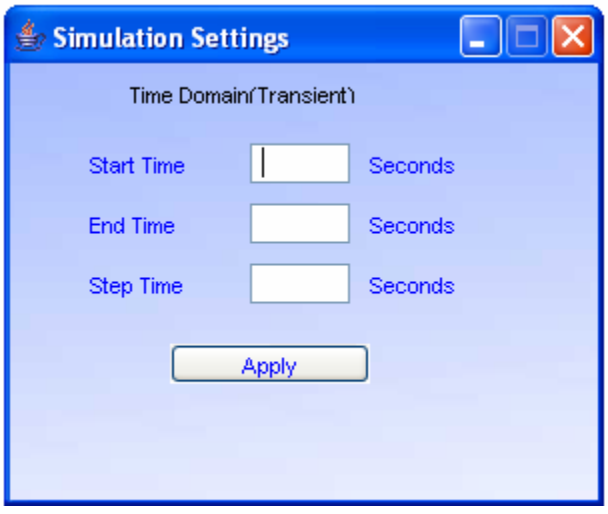
xi) Server File Manager Dialog

The Server File Manager Dialog, as shown in the figure below, will be displayed when you choose the server file manage action.



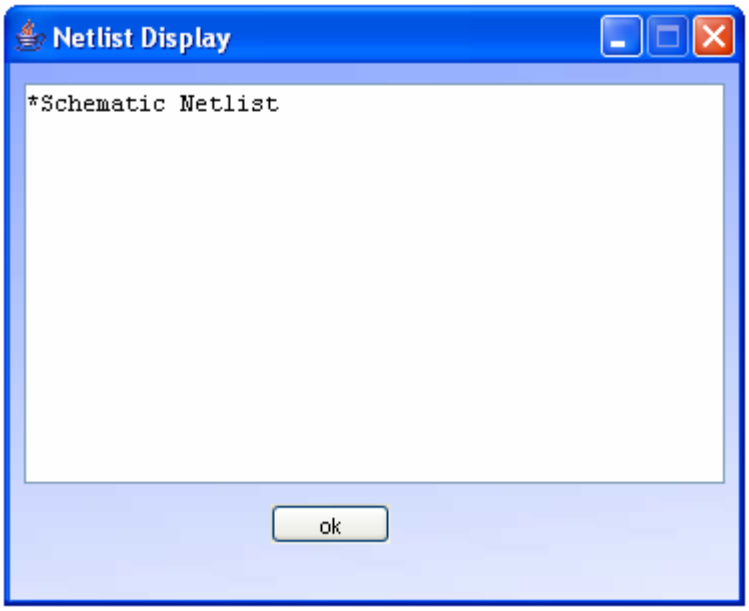
xii) Simulation Settings Dialog

The Simulation Settings Dialog, as shown in the figure below, will be displayed when you choose the simulation settings action.



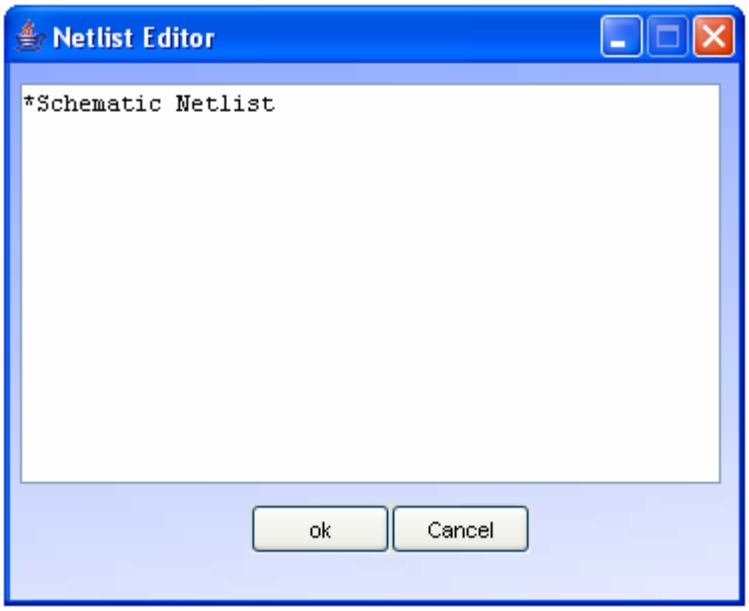
xiii) Netlist Display Dialog

The Netlist Display Dialog, as shown in the figure below, will be displayed when you choose the view Netlist action.



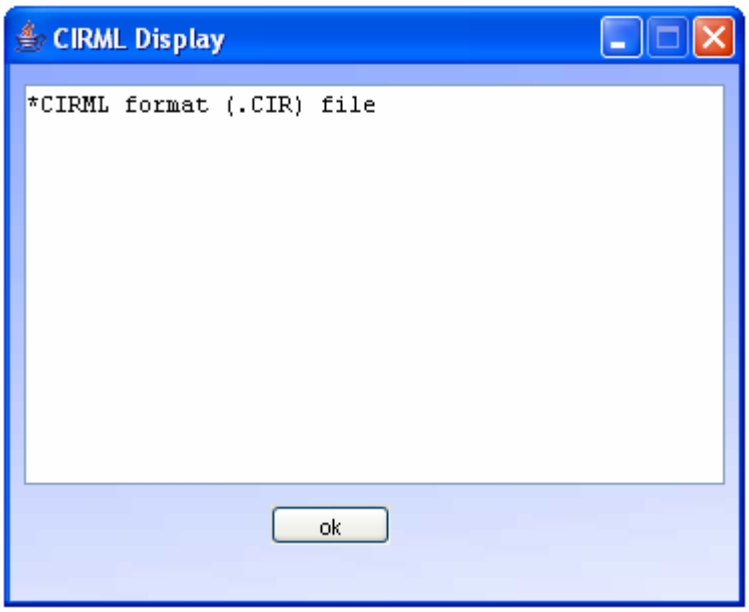
xiv) Netlist Editor Dialog

The Netlist Editor Dialog, as shown in the figure below, will be displayed when you choose the edit Netlist action.



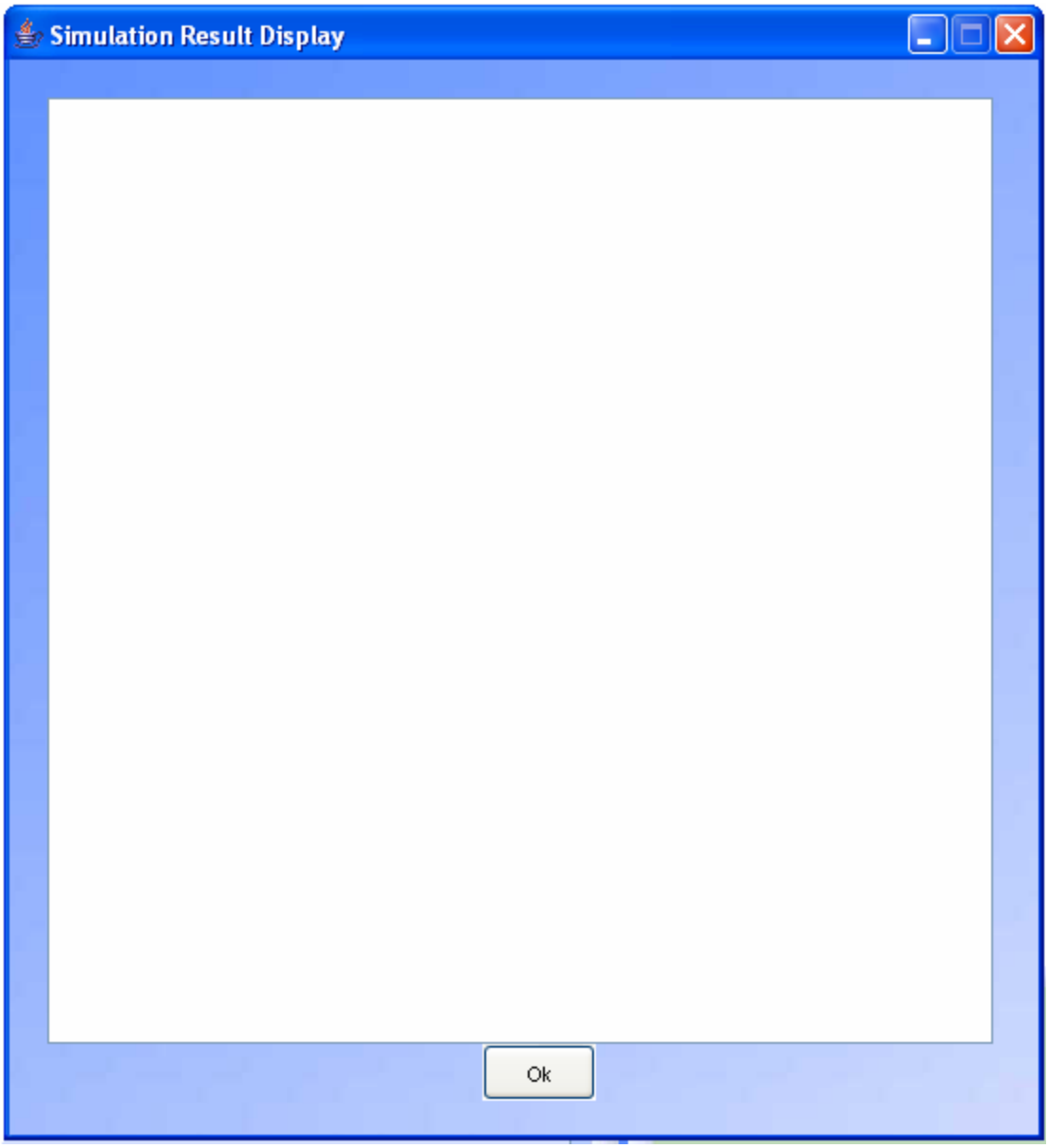
xv) CIRML Display Dialog

The CIRML Display Dialog, as shown in the figure below, will be displayed when you choose the view CIRML action.



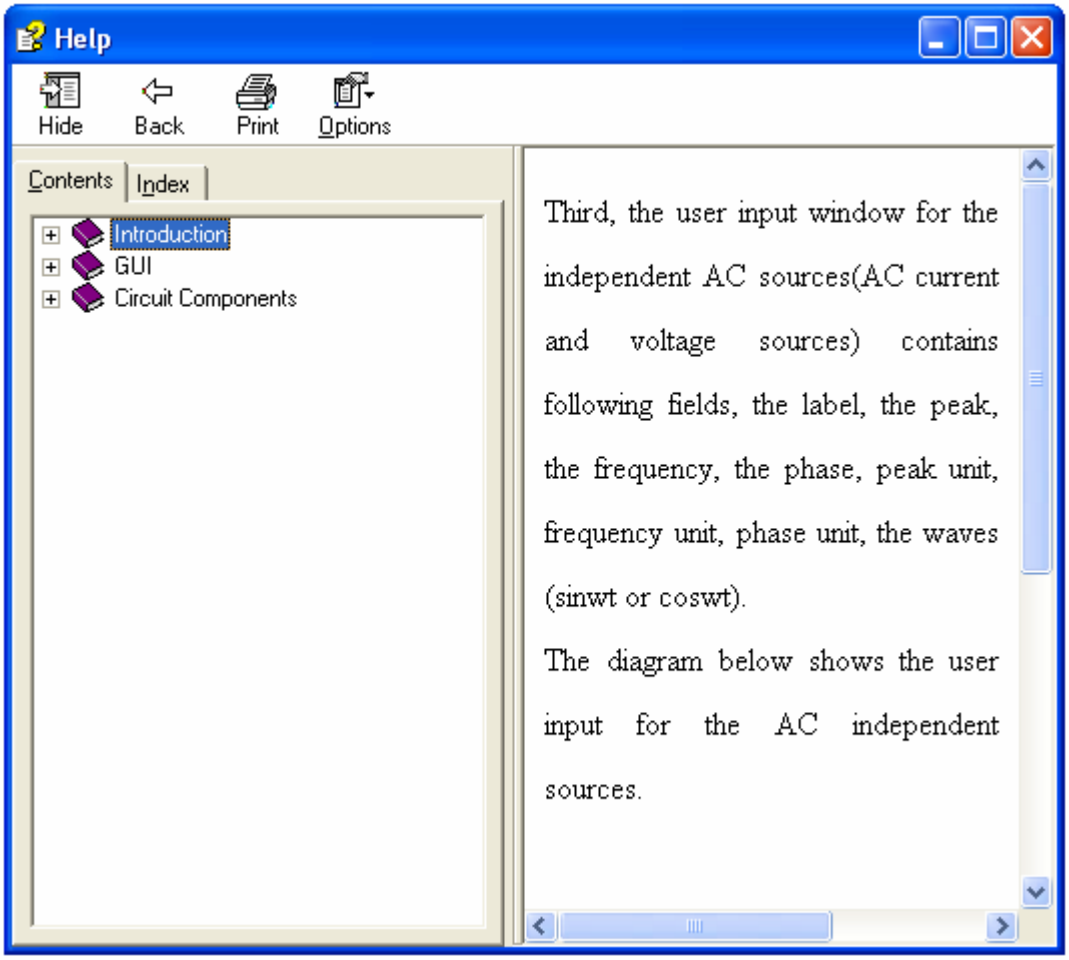
xvi) Simulation Results Display Dialog

The Simulation Results Display Dialog, as shown in the figure below, will be displayed when you choose the view simulation results action.



xvii) Online Help Window

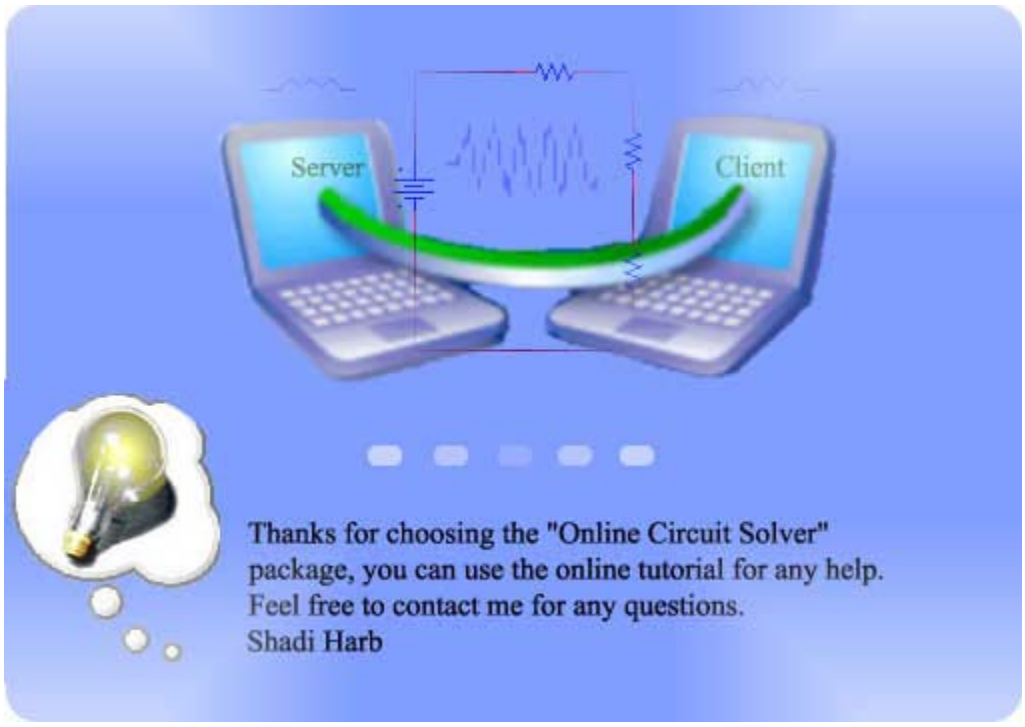
The Online Help Window, as shown in the figure below, will be displayed when you choose the help action.



1.5.4 Popup Messages

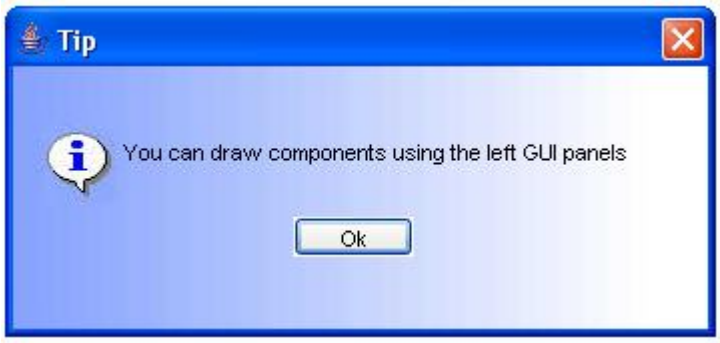
i) Welcome Message

The Welcome Message, as shown in the figure below, will be displayed when the applet starts.



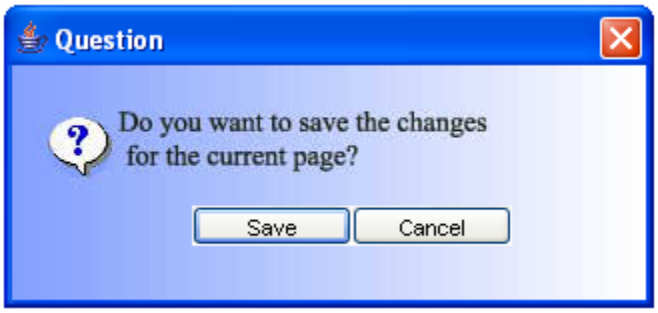
ii) Tip Message

The Tip Message, as shown in the figure below, will be displayed when the applet starts.



iii) Save Schematic Page Message

The Save Schematic Page Message, as shown in the figure below, will be displayed when you choose the delete schematic page action.

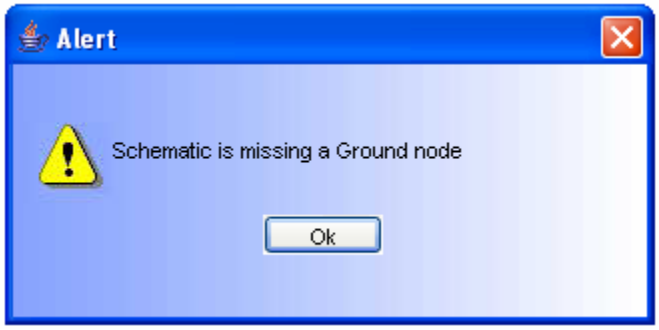


iv) Floating Node Message

The Floating Node Message, as shown in the figure below, will be displayed when a floating node is found in the current schematic.

v) Ground Does Not Exist Message

The Ground Does Not Exist Message, as shown in the figure below, will be displayed when a ground is missing in the current schematic.



vi) Dependent Controller Not Found Message

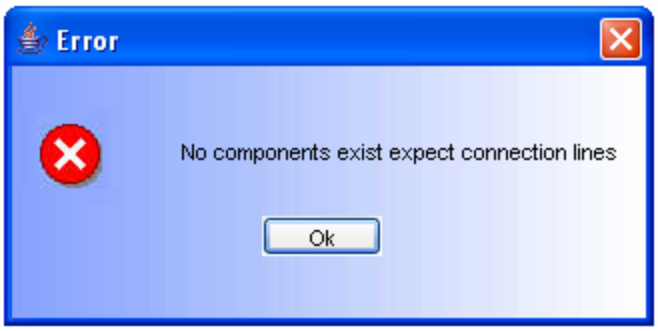
The Dependent Controller Not Found Message, as shown in the figure below, will be displayed when a dependent controller is not found in the current schematic.

vii) Schematic Does Not Exist Message

The Schematic Does Not Exist Message, as shown in the figure below, will be displayed when a schematic is empty.

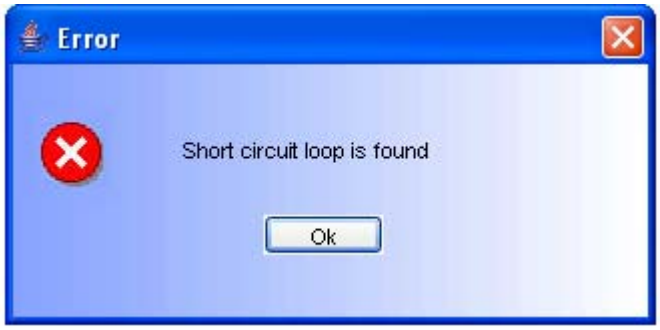
viii) Only Connection Lines Exist Message

The Only Connection Lines Exist Message, as shown in the figure below, will be displayed when no components exist except connection lines.



ix) Short Circuit Message

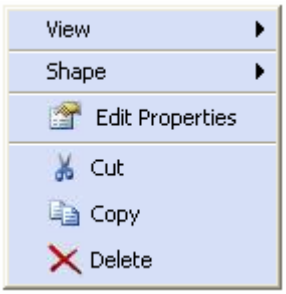
The Short Circuit Message, as shown in the figure below, will be displayed when a short circuit loop is found.



1.5.5 Popup Menus

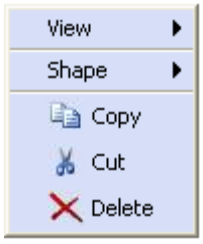
i) Single Element Popup Menu

The Single Element Popup Menu will be displayed when you select a single component and right click on the drawing area as shown in the figure below.



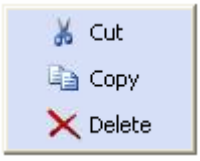
ii) Multiple Elements Popup Menu

The Multiple Elements Popup Menu will be displayed when you select a group of components and right click on the drawing area as shown in the figure below.



iii) Connection Line Popup Menu

The Connection Line Popup Menu will be displayed when you select a connection line and right click on the drawing area as shown in the figure below.



iv) Grid Area Popup Menu

Grid Area Popup Menu will be displayed when you right click on the drawing area as shown in the figure below.

1.5.6 Keyboard Shortcuts

Key	Description
Ctrl+N	Create New Page
Ctrl+O	Open Page
Ctrl+S	Save Page
Ctrl+Z	Undo
Ctrl+W	Redo
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl-V	Paste
Ctrl+A	Select All
Ctrl+D	Duplicate
Ctrl+R	Rotate Right
Ctrl+L	Rotate Left
Ctrl+H	Flip Horizontal
Ctrl+F	Flip Vertical
Shift+I	Zoom In
Shift+O	Zoom Out
Shift+F	Fit Zoom
Shift+G	Group
Shift+U	Ungroup
Shift+R	Regroup
Ctrl+Shift+T	Align Top
Ctrl+Shift+B	Align Bottom
Ctrl+Shift+R	Align Right
Ctrl+Shift+L	Align Left
Alt+F4	Delete Current Page
DEL	Delete Component
F1	Show Help Dialog

F2	Edit Component Attributes
F3	Change Component Display
F4	Edit Canvas Properties
F5	Create Netlist
F6	Edit Netlist
F7	Edit Simulation Parameters
F8	View CIRML Output
F9	Run Simulation
F10	View Simulation Results

1.5.7 Components Attributes Description

Attributes	Description
Label	To edit the name of the component
Value	To edit the value of the component
Unit	To edit the unit of the value
Peak	To edit the peak value of the waveform in the AC sources
Peak Unit	To edit the unit of the peak value unit of the waveform in the AC sources
Frequency	To edit the frequency value of the waveform in the AC sources
Frequency Unit	To edit the frequency value unit of the waveform in the AC sources
Phase	To edit the phase angle of the waveform in the AC sources
Phase Unit	To edit the phase angle unit of the waveform in the AC sources
Sin(wt)/cos(wt)	To edit the shape of the waveform in the AC sources
Control Variable	To edit the control variable associated with the dependent sources
Gain	To edit the ratio between the value of the controlling and the controlled sources
Gain Unit	To edit the ratio unit between the controlling and the controlled sources

1.6 Hints and Tips

1) Make sure that the circuit has a ground.
2) Every circuit component should have a numeric value.
3) A connection line can be drawn by selecting a component and starting to drag at one of the end points.

.

APPENDIX C: SOURCE CODE

mainApplet.java

```
package ECS;

////////////////////////////////////////////////////////////////////////////
/**
 * This is the main applet class in the Electrical Circuit Solver
 * which initializes the applet and displays the GUI graphics and take care
 * of all the GUI eventhandling.
 * @author Shadi Harb
 * @version 0.5 December 2004
 */

////////////////////////////////////////////////////////////////////////////
import java.io.*;
import java.lang.reflect.*;
import java.net.*;
import java.util.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.FileFilter;

import ECS.CircuitComponents.*;
import ECS.CircuitComponents.Pin1Components.*;
import ECS.CircuitComponents.Pin2Components.*;
import ECS.CircuitComponents.Pin2Components.ConnectionLine;
import ECS.DrawingTools.*;
import ECS.GUIFrameWindows.*;
import ECS.GUIMessages.*;
import ECS.GUPanels.*;

/*////////////////////////////////////////////////////////////////////////////
//                          MAIN APPLET                                     //
////////////////////////////////////////////////////////////////////////////*/
public class mainApplet
   extends JApplet {
//*************************************//
//   VARIABLE INITILAZATION   //
//*************************************//
  public static int componentsCount = ActiveState.CURRENTMARK + 1;
```

```
public static Components[][] allComps;
public static Components[][] deletedComps;
public static DrawTool[] drawComps;
public CompsStack undoStack;
public static Vector txtLabels = new Vector();
public static Vector netListNodes = new Vector();
public static int[] clickedPos = new int[2];
public static int[] startPoint = new int[2];
public static Color canvasBackgroundColor = Color.WHITE;
public static Color canvasForegroundColor = Color.BLUE;
public static Color drawingFillColor = Color.WHITE;
public static Color drawingLineColor = Color.BLUE;
public static Color drawingTextColor = Color.BLUE;
public static int canvasWidth = 2500;
public static int canvasHeight = 2500;
public double[] index;
public String netListString = "", cirMLString = "";
public String pageTitle = "";

private Components[][][] recordComps;
private Components[][][] groupedComps;
private Components[] activeComps;
private Components currentComp;
private Components activeComponent;
public static WorkPlace page1, page2, page3, page4, page5, copiedPage;
private String circuitType = null;
private String graphicalResultStr = "";
public int[] empty = new int[componentsCount];
private int y = 6;
private ActiveState activestate = new ActiveState();
private boolean activecomp;
private boolean NVComponent = false;
private int numbOfGraphs;
private int[] xyPressed = new int[2];
private int[] xyReleased = new int[2];
private int[] xyold = new int[2];
private int[] xynew = new int[2];
private int[] xyClicked = new int[2];
private double minVI = 100, maxVI = 0;
private int XAxiaStep;
private int[] type = {
   -1, -1};
private int undoNumb = 0;
private String probesString = "", netlistPanelString = "";
private Vector probesVector = new Vector();
```

```
private String outputResult = "";
private boolean start = false;
private Vector btns = new Vector();

String[] NetNames = new String[1];
Point[] NetPoints = new Point[1];
private boolean load = false, save = false;
private int result;
private int[] oldPos = new int[2];
private boolean moved = false;
private boolean rotateGroup = false;
Font myFont = new Font("Serif", Font.PLAIN, 13);
/*********************************************
 Initilization Components and Buttons
 *********************************************/
private JPanel drawAreaPanel, toolsPanel, simulationPanel,
    panelR
    , panelLU, panelLD, panelDR, panelDL, passivePanel, independentPanel,
    dependentPanel,
    connectionsPanel, transformersPanel, menuPanel, fileManagerPanel,
    elementsToolPanel, statusPanel, visitedElementsPanel;
private JScrollPane mainScroll, pagesExplorerScrollBar;
public static GridCanvas gridCanvas;
private JButton netListBtn, simulateBtn, passiveBtn, independentBtn,
    dependentBtn
    , connectionsBtn, saveBtn, loadBtn, clearBtn, resistorBtn, capacitorBtn,
    inductorBtn, transformerBtn, dcVSBtn, dcCSBtn, acVSBtn, acCSBtn, VCVSBtn,
    VCCSBtn,
    CCVSBtn, CCCSBtn, connectionBtn, nodeBtn, groundBtn, diffVoltageMBtn,
    currentMBtn,
    voltageMBtn, transformerABtn, transformerBBtn, transformerCBtn,
    transformerDBtn,
    fileManagerBtn, zoomInBtn, zoomOutBtn, saveSchematicBtn, openSchematicBtn,
    newSchematicBtn,
    helpBtn, newBtn, cutBtn, copyBtn, pasteBtn, undoBtn, runBtn, handBtn,
    dCursorBtn, deletePageBtn,
    fitZoomBtn, insertTextBtn, fillColorBtn, lineColorBtn,
    createNetlistBtn,
    circleBtn, rectangleBtn, dftCursorBtn, simSettingsBtn,
    simulationSettingsBtn, lineBtn, new1Btn,
    new2Btn, new3Btn, new4Btn, new5Btn, VResistorBtn, VCapacitorBtn,
    VInductorBtn,
    VPPTransformerBtn, VPNTransformerBtn, VNPTransformerBtn,
    VNNTransformerBtn,
    VVCVSBtn, VVCCSBtn,
```

```
    VCCVSBtn, VCCCSBtn, VDCVoltageBtn, VDCCurrentBtn, VACVoltageBtn,
    VACCurrentBtn,
    VConnLineBtn, VConnNodeBtn, VGroundBtn, VCurrentMarkerBtn,
    VVoltageMarkerBtn, VVoltageDiffMarkerBtn,
    textColorBtn, roundRectangleBtn, arcBtn, deleteBtn, selectAllCompsBtn,
    deleteSchematicBtn, editNetlistBtn, showVoltagesBtn, showCurrentsBtn,
    showHideNodesBtn, showOutputResultBtn, showCIRMLBtn, voltageMarkerBtn,
    voltageDifferentialMarkerBtn, currentMarkerBtn, netlistEditingBtn,
    editComponentBtn, displayPropertiesBtn, rotateRightBtn, rotateLeftBtn,
    groupComponentsBtn, ungroupComponentsBtn, forwardBtn, backwardBtn,
    refreshBtn,
    connectFMBtn, disconnectFMBtn, uploadFMBtn, downloadFMBtn,
    mkDirectoryFMBtn,
    rmDirectoryFMBtn, renameFMBtn, deleteFMBtn, redoBtn, flipVerticalBtn,
    flipHorizontalBtn, rotateTextBtn, canvasColorsBtn, canvasSizeBtn,
    serverFileManagerBtn, alignRightBtn, alignLeftBtn, alignTopBtn,
    alignBottomBtn,
    drawResultBtn, lastForwardBtn, lastBackwardBtn, gridBtn, clearAllCompsBtn,
    contactUsBtn, regroupComponentsBtn, goToBtn, arrowUpBtn, arrowDownBtn;
private JLabel toolLabel, circuitLabel, simulationLabel, title;
private JToolBar passiveToolbar, independentToolbar, dependentToolbar,
    connectionsToolbar, standardToolbar, upperToolbar2, transformersToolbar,
    pagesExplorerToolbar, paintingToolbar, simulationToolbar,
    componentToolbar,
    serverFileManagerToolbar, canvasToolbar;
private JPopupMenu singleElementPopmenu = new JPopupMenu();
private JPopupMenu gridPopupMenu = new JPopupMenu();
private JPopupMenu connectionLinePopupMenu = new JPopupMenu();
private JPopupMenu multiElementsPopmenu = new JPopupMenu();
private JPopupMenu closeMenu = new JPopupMenu();
private JMenu singleElementShapePopMenu = new JMenu("       Shape");
private JMenu singleElementViewPopMenu = new JMenu("       View");
private JMenu multiElementsShapePopMenu = new JMenu("       Shape");
private JMenu multiElementsViewPopMenu = new JMenu("       View");
private JMenuItem enterValue; // = new JMenuItem("Enter Value");
private JMenuItem singleElementRotateRight; // = new JMenuItem();
private JMenuItem singleElementRotateLeft; // = new JMenuItem();
private JMenuItem singleElementDelete; // = new JMenuItem("Delete Del");
private JMenuItem clearAll; // = new JMenuItem("Clear All");
private JMenuItem multiElementsRotateLeft;
private JMenuItem multiElementsRotateRight; // = new JMenuItem();
private JMenuItem multiElementsDelete; // = new JMenuItem("Delete Del");
private JMenuItem deleteLine; // = new JMenuItem("Delete");
private JMenuItem copyLine; // = new JMenuItem("Copy");
private JMenuItem cutLine; // = new JMenuItem("Cut");
```

```
private JMenuItem undo = new JMenuItem("Undo");
private JMenuItem menuItem10 = new JMenuItem();
private JMenuItem singleElementCopy; // = new JMenuItem("Copy");
private JMenuItem multiElementsCopy; // = new JMenuItem("Copy");
private JMenuItem paste = new JMenuItem("Paste");
private JMenuItem singleElementCut; // = new JMenuItem("Cut");
private JMenuItem multiElementsCut; // = new JMenuItem("Cut");
private JMenuItem multiElementsGroup; // = new JMenuItem("Group");
private JMenuItem multiElementsUngroup; // = new JMenuItem("Ungroup");
private JMenuItem goToLine, goToPage, zoomIn, zoomOut;
private NetlistOutputPanel netlistPanel;
private GraphicalResultPanel graphicalResult;
private GraphicalPanel elementsPanel;
private SimulationResultFrame outputResultFrame; //=new SimulationResultFrame();
private Cursor zoomInCursor, zoomOutCursor;
private JMenuBar menuBar;
private JMenu fileMenu, editMenu, viewMenu, optionsMenu, simulationMenu,
   helpMenu,
   toolsBarMenu, canvasPropertiesMenu, zoomingMenu, markersMenu,
   navigateMenu, drawMenu,
   componentsMenu, rotateFlipMenu, groupMenu, alignMenu, analysisMenu,
   goToPageMenu;

private JMenuItem newMenuItem, openMenuItem, saveMenuItem, closePageMenuItem,
   cutMenuItem, copyMenuItem, pasteMenuItem, deleteMenuItem,
   selectAllMenuItem, undoMenuItem, redoMenuItem
   , zoomInMenuItem, zoomOutMenuItem, fitZoomMenuItem,
   scrollHandMenuItem, defaultCursorMenuItem, helpMenuItem,
   contactUsMenuItem,
   standardBarMenuItem, visitedElementsBarMenuItem, statusBarMenuItem,
   pageExplorerBarMenuItem, simulationBarMenuItem, acknowledgmentMenuItem,
   exitMenuItem,
   displayProperties, multiElementsRegroup, singleElementDisplayProperties,
   singleElementZoomIn, singleElementZoomOut, singleElementGoTo
   , singleElementFlipVertical, singleElementFlipHorizontal,
   multiElementsDisplayProperties,
   multiElementsZoomIn, multiElementsZoomOut, multiElementsGoTo,
   multiElementsFlipVertical, multiElementsFlipHorizontal,
   multiElementsAlignRight, multiElementsAlignLeft, multiElementsAlignTop,
   multiElementsAlignBottom,
   editNetlist, viewNetlist, showNodes, voltageMarker, diffVoltageMarker,
   currentMarker, editSimulation,
   run, showVoltages, showCurrents, viewCIRML, viewSimulationResult,
   drawResult, canvasColorsMenuItem,
   canvasSizeMenuItem, voltageMarkerMenuItem, diffVoltageMarkerMenuItem,
```

```
    currentMarkerMenuItem, hideAllMarkersMenuItem,
    showAllMarkersMenuItem, lastPreviousMenuItem, previousMenuItem,
    nextMenuItem, lastNextMenuItem, goToMenuItem,
    editPageInfoMenuItem, drawTextMenuItem, drawLineMenuItem,
    drawCircleMenuItem, drawRectangleMenuItem,
    drawRoundRectangleMenuItem, displayPropertiesMenuItem,
    editPropertiesMenuItem, rotateRightMenuItem,
    rotateLeftMenuItem, flipVerticalMenuItem, flipHorizontalMenuItem,
    rotateTextMenuItem, alignRightMenuItem,
    alignLeftMenuItem, alignTopMenuItem, alignBottomMenuItem, groupMenuItem,
    regroupMenuItem, ungroupMenuItem,
    editNetlistMenuItem, createNetlistMenuItem, showNodesNetNamesMenuItem,
    showNodesVoltagesMenuItem,
    showNodesCurrentsMenuItem, simulationSettingsMenuItem, runMenuItem,
    viewCIRMLMenuItem, viewSimulationResultMenuItem,
    drawResultMenuItem, showGridMenuItem, showConnectionPointsMenuItem,
    removeConnectionLinesMenuItem, clearAllMenuItem,
    copyPageMenuItem, pastePageMenuItem, selectPageMenuItem, saveAsMenuItem,
    findMenuItem, replaceMenuItem, duplicateMenuItem, page1MenuItem,
    page2MenuItem, page3MenuItem, page4MenuItem, page5MenuItem,
    canvasBarMenuItem, componentBarMenuItem, paintingBarMenuItem
    , redrawMenuItem, refreshMenuItem, fillColorMenuItem, lineColorMenuItem,
    textColorMenuItem, statusBarSettingsMenuItem;

public String startTime = "0", endTime = "200m", stepTime = "10m";
public final MouseListener btnsMouseListener = new java.awt.event.
    MouseAdapter() {
  public void mouseEntered(MouseEvent me) {
    btns_mouseEntered(me);
  }

  public void mouseExited(MouseEvent me) {
    btns_mouseExited(me);
  }

};

public final ActionListener btnsActionListener = new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    btnsActionPerformed(ae);
  }
};

public final ActionListener menusActionListener = new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
```

```
    menusActionPerformed(ae);
  }
};

public final MouseListener canvasMouseListener = new MouseAdapter() {
  public void mouseClicked(MouseEvent me) {
    gridCanvas_mouseClicked(me);
  }

  public void mousePressed(MouseEvent me) {
    gridCanvas_mousePressed(me);
  }

  public void mouseReleased(MouseEvent me) {
    gridCanvas_mouseReleased(me);
  }

  public void mouseExited(MouseEvent me) {
    gridCanvas_mouseExited(me);
  }
};

public final MouseMotionListener canvasMouseMotionListener = new
    MouseMotionAdapter() {
  public void mouseMoved(MouseEvent me) {
    gridCanvas_mouseMoved(me);
  }

  public void mouseDragged(MouseEvent me) {
    gridCanvas_mouseDragged(me);
  }
};

public final KeyListener canvasKeyListener = new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    gridCanvas_keyPressed(ke);
  }

  public void keyReleased(KeyEvent ke) {
    gridCanvas_keyReleased(ke);
  }

};
public final MouseListener mainScrollMouseListener = new MouseAdapter() {
  public void mouseReleased(MouseEvent me) {
```

```
    mainScroll_mouseReleased(me);
   }

  };

  public final KeyListener mainScrollKeyListener = new KeyAdapter() {
   public void keyPressed(KeyEvent ke) {
    mainScroll_keyPressed(ke);
   }
  };
  public final MouseMotionListener mainScrollMouseMotionListener = new
     MouseMotionAdapter() {
   public void mouseDragged(MouseEvent me) {
    mainScroll_mouseDragged(me);
   }
  };

//*************************************************************************/
/
  public void init() {

   allComps = new Components[componentsCount][];
   deletedComps = new Components[componentsCount][];
   recordComps = new Components[10][componentsCount][];
   groupedComps = new Components[1][componentsCount][];
   drawComps = new DrawTool[1];
   page1 = new WorkPlace(canvasWidth, canvasHeight, canvasBackgroundColor,
                 canvasForegroundColor, "page1", pageTitle);
   page2 = new WorkPlace(canvasWidth, canvasHeight, canvasBackgroundColor,
                 canvasForegroundColor, "page2", pageTitle);
   page3 = new WorkPlace(canvasWidth, canvasHeight, canvasBackgroundColor,
                 canvasForegroundColor, "page3", pageTitle);
   page4 = new WorkPlace(canvasWidth, canvasHeight, canvasBackgroundColor,
                 canvasForegroundColor, "page4", pageTitle);
   page5 = new WorkPlace(canvasWidth, canvasHeight, canvasBackgroundColor,
                 canvasForegroundColor, "page5", pageTitle);
   undoStack = new CompsStack();
   ActiveState.origComps = new Components[componentsCount][];
   ActiveState.copiedComps = new Components[componentsCount][];
   ActiveState.splitedLines = ActiveState.createNewElements(ActiveState.
     CONNECTOR);
   ActiveState.deletedLines = ActiveState.createNewElements(ActiveState.
     CONNECTOR);
   for (int i = 0; i < componentsCount; i++) {
    allComps[i] = ActiveState.createNewElements(i);
```

```
  deletedComps[i] = ActiveState.createNewElements(i);
  ActiveState.origComps[i] = ActiveState.createNewElements(i);
  ActiveState.copiedComps[i] = ActiveState.createNewElements(i);
  empty[i] = 0;

}

for (int j = 0; j < 10; j++) {

  for (int type = 0; type < componentsCount; type++) {
    recordComps[j][type] = ActiveState.createNewElements(type);
  }

}

Image zoominImg = getImage(getDocumentBase(), "ECS/images/zoomingIn.jpg");
Image zoomoutImg = getImage(getDocumentBase(), "ECS/images/zoomingOut.jpg");
Toolkit tk = getToolkit();
Point hotPoint = new Point(0, 0);
zoomInCursor = tk.createCustomCursor(zoominImg, hotPoint, "ZoomIn Cursor");
zoomOutCursor = tk.createCustomCursor(zoomoutImg, hotPoint,
                              "ZoomOut Cursor");

/**************************************
       Applet Area Setup
 **************************************/
this.getContentPane().setSize(900, 500);
this.getContentPane().setLayout(null);
this.getContentPane().setBackground(new Color(91, 141, 255));
/**************************************
       Load Components
 **************************************/

uploadComponentsImgs();
/**************************************
      Drawing Area Setup
 **************************************/
drawGridArea();

/**************************************
      Drawing NetList Panel
 **************************************/
drawNetListPanel();

drawGraphicalPanel();
```

```
/**************************************
      Tools Area Setup
 **************************************/
drawToolsPanel();
drawPassiveElements();
drawIndependentSources();
drawDependentSources();
drawConnections();
drawTransformers();
drawFileManagerPanel();
buildPopupMenus();
/**************************************
      Simulation Area Setup
 **************************************/
drawSimulationPanel();

/**************************************
      Draw Toolbars
 **************************************/

drawStandrardToolbar();
drawSimulationToolbar();
drawComponentToolbar();
drawVisitedElementsPanel();
drawpagesExplorerToolbar();
drawPaintingBar();
drawCanvasToolbar();
//drawServerFileManagerToolbar();
/**************************************
      Draw Panels
 **************************************/
drawPanelR();
drawPanelLD();
//drawPanelDL();
//drawPanelDR();
drawStatusBar();
/**************************************
      Build The Menubar
 **************************************/
drawMenubar();

this.getContentPane().addMouseMotionListener(new MouseMotionAdapter() {

  public void mouseDragged(MouseEvent me) {
```

```
      applet_mouseDragged(me);
    }
  });

  gridCanvas.addMouseListener(canvasMouseListener);
  gridCanvas.addMouseMotionListener(canvasMouseMotionListener);
  gridCanvas.addKeyListener(canvasKeyListener);

//*************************************************************************************/
/
//                    ADDING LISTINERS                          //
//*************************************************************************************/
/

  //************GRIDCANVAS LISTENERS*******//

//.....................................................................//
  /*KeyAdapter servicesKeyAdapter = new KeyAdapter()
   {
        public void keyTyped(KeyEvent e)
        {
         System.out.println("127..... pressed....");
   if(e.getKeyCode()==127)
   {
   ActiveState.multiActive=false;
   deleteComponent(true);
   gridCanvas.gridImage.clearRect(0,0,650,550);
   gridCanvas.drawGridLayout(startPoint);
   drawAllComps();
   gridCanvas.redraw();
   }
        }
   };
   gridCanvas.addKeyListener(servicesKeyAdapter);
   */
//.....................................................................//

  //************BUTTONS LISTENERS*******//
  simSettingsBtn.addMouseListener(btnsMouseListener);
  passiveBtn.addMouseListener(btnsMouseListener);
  independentBtn.addMouseListener(btnsMouseListener);
  dependentBtn.addMouseListener(btnsMouseListener);
  connectionsBtn.addMouseListener(btnsMouseListener);
  netListBtn.addMouseListener(btnsMouseListener);
  simulateBtn.addMouseListener(btnsMouseListener);
```

```
netlistEditingBtn.addMouseListener(btnsMouseListener);
fileManagerBtn.addMouseListener(btnsMouseListener);
deleteSchematicBtn.addMouseListener(btnsMouseListener);
clearBtn.addMouseListener(btnsMouseListener);
saveBtn.addMouseListener(btnsMouseListener);
loadBtn.addMouseListener(btnsMouseListener);
newSchematicBtn.addMouseListener(btnsMouseListener);
saveSchematicBtn.addMouseListener(btnsMouseListener);
openSchematicBtn.addMouseListener(btnsMouseListener);
zoomInBtn.addMouseListener(btnsMouseListener);
zoomOutBtn.addMouseListener(btnsMouseListener);
newBtn.addMouseListener(btnsMouseListener);
//loadBtn.addMouseListener(btnsMouseListener);
cutBtn.addMouseListener(btnsMouseListener);
copyBtn.addMouseListener(btnsMouseListener);
pasteBtn.addMouseListener(btnsMouseListener);
undoBtn.addMouseListener(btnsMouseListener);
redoBtn.addMouseListener(btnsMouseListener);
deleteBtn.addMouseListener(btnsMouseListener);
contactUsBtn.addMouseListener(btnsMouseListener);
selectAllCompsBtn.addMouseListener(btnsMouseListener);
helpBtn.addMouseListener(btnsMouseListener);
displayPropertiesBtn.addMouseListener(btnsMouseListener);
editComponentBtn.addMouseListener(btnsMouseListener);
rotateRightBtn.addMouseListener(btnsMouseListener);
rotateLeftBtn.addMouseListener(btnsMouseListener);
flipVerticalBtn.addMouseListener(btnsMouseListener);
flipHorizontalBtn.addMouseListener(btnsMouseListener);
rotateTextBtn.addMouseListener(btnsMouseListener);
groupComponentsBtn.addMouseListener(btnsMouseListener);
regroupComponentsBtn.addMouseListener(btnsMouseListener);
ungroupComponentsBtn.addMouseListener(btnsMouseListener);
alignTopBtn.addMouseListener(btnsMouseListener);
alignBottomBtn.addMouseListener(btnsMouseListener);
alignRightBtn.addMouseListener(btnsMouseListener);
alignLeftBtn.addMouseListener(btnsMouseListener);
serverFileManagerBtn.addMouseListener(btnsMouseListener);
handBtn.addMouseListener(btnsMouseListener);
dCursorBtn.addMouseListener(btnsMouseListener);
deletePageBtn.addMouseListener(btnsMouseListener);
fitZoomBtn.addMouseListener(btnsMouseListener);
runBtn.addMouseListener(btnsMouseListener);
insertTextBtn.addMouseListener(btnsMouseListener);
fillColorBtn.addMouseListener(btnsMouseListener);
lineColorBtn.addMouseListener(btnsMouseListener);
```

```
//canvasPropertiesBtn.addMouseListener(btnsMouseListener);
dftCursorBtn.addMouseListener(btnsMouseListener);
circleBtn.addMouseListener(btnsMouseListener);
rectangleBtn.addMouseListener(btnsMouseListener);
lineBtn.addMouseListener(btnsMouseListener);
goToBtn.addMouseListener(btnsMouseListener);
canvasColorsBtn.addMouseListener(btnsMouseListener);
canvasSizeBtn.addMouseListener(btnsMouseListener);
lastForwardBtn.addMouseListener(btnsMouseListener);
forwardBtn.addMouseListener(btnsMouseListener);
backwardBtn.addMouseListener(btnsMouseListener);
lastBackwardBtn.addMouseListener(btnsMouseListener);
refreshBtn.addMouseListener(btnsMouseListener);
gridBtn.addMouseListener(btnsMouseListener);
simulationSettingsBtn.addMouseListener(btnsMouseListener);
createNetlistBtn.addMouseListener(btnsMouseListener);
new1Btn.addMouseListener(btnsMouseListener);
new2Btn.addMouseListener(btnsMouseListener);
new3Btn.addMouseListener(btnsMouseListener);
new4Btn.addMouseListener(btnsMouseListener);
new5Btn.addMouseListener(btnsMouseListener);
VResistorBtn.addMouseListener(btnsMouseListener);
VCapacitorBtn.addMouseListener(btnsMouseListener);
VInductorBtn.addMouseListener(btnsMouseListener);
VPPTransformerBtn.addMouseListener(btnsMouseListener);
VPNTransformerBtn.addMouseListener(btnsMouseListener);
VNPTransformerBtn.addMouseListener(btnsMouseListener);
VNNTransformerBtn.addMouseListener(btnsMouseListener);
VDCVoltageBtn.addMouseListener(btnsMouseListener);
VDCCurrentBtn.addMouseListener(btnsMouseListener);
VACVoltageBtn.addMouseListener(btnsMouseListener);
VACCurrentBtn.addMouseListener(btnsMouseListener);
VVCVSBtn.addMouseListener(btnsMouseListener);
VVCCSBtn.addMouseListener(btnsMouseListener);
VCCVSBtn.addMouseListener(btnsMouseListener);
VCCCSBtn.addMouseListener(btnsMouseListener);
VConnLineBtn.addMouseListener(btnsMouseListener);
VConnNodeBtn.addMouseListener(btnsMouseListener);
VGroundBtn.addMouseListener(btnsMouseListener);
VCurrentMarkerBtn.addMouseListener(btnsMouseListener);
VVoltageMarkerBtn.addMouseListener(btnsMouseListener);
textColorBtn.addMouseListener(btnsMouseListener);
roundRectangleBtn.addMouseListener(btnsMouseListener);
showVoltagesBtn.addMouseListener(btnsMouseListener);
showCurrentsBtn.addMouseListener(btnsMouseListener);
```

```
editNetlistBtn.addMouseListener(btnsMouseListener);
showOutputResultBtn.addMouseListener(btnsMouseListener);
showHideNodesBtn.addMouseListener(btnsMouseListener);
showCIRMLBtn.addMouseListener(btnsMouseListener);
voltageMarkerBtn.addMouseListener(btnsMouseListener);
currentMarkerBtn.addMouseListener(btnsMouseListener);
drawResultBtn.addMouseListener(btnsMouseListener);
voltageDifferentialMarkerBtn.addMouseListener(btnsMouseListener);
clearAllCompsBtn.addMouseListener(btnsMouseListener);

simSettingsBtn.addActionListener(btnsActionListener);
passiveBtn.addActionListener(btnsActionListener);
independentBtn.addActionListener(btnsActionListener);
dependentBtn.addActionListener(btnsActionListener);
connectionsBtn.addActionListener(btnsActionListener);
netListBtn.addActionListener(btnsActionListener);
simulateBtn.addActionListener(btnsActionListener);
netlistEditingBtn.addActionListener(btnsActionListener);
fileManagerBtn.addActionListener(btnsActionListener);
deleteSchematicBtn.addActionListener(btnsActionListener);
clearBtn.addActionListener(btnsActionListener);
saveBtn.addActionListener(btnsActionListener);
loadBtn.addActionListener(btnsActionListener);
newSchematicBtn.addActionListener(btnsActionListener);
saveSchematicBtn.addActionListener(btnsActionListener);
openSchematicBtn.addActionListener(btnsActionListener);
zoomInBtn.addActionListener(btnsActionListener);
zoomOutBtn.addActionListener(btnsActionListener);
newBtn.addActionListener(btnsActionListener);
//loadBtn.addActionListener(btnsActionListener);
cutBtn.addActionListener(btnsActionListener);
copyBtn.addActionListener(btnsActionListener);
pasteBtn.addActionListener(btnsActionListener);
undoBtn.addActionListener(btnsActionListener);
redoBtn.addActionListener(btnsActionListener);
deleteBtn.addActionListener(btnsActionListener);
contactUsBtn.addActionListener(btnsActionListener);
selectAllCompsBtn.addActionListener(btnsActionListener);
helpBtn.addActionListener(btnsActionListener);
displayPropertiesBtn.addActionListener(btnsActionListener);
editComponentBtn.addActionListener(btnsActionListener);
rotateRightBtn.addActionListener(btnsActionListener);
rotateLeftBtn.addActionListener(btnsActionListener);
flipVerticalBtn.addActionListener(btnsActionListener);
flipHorizontalBtn.addActionListener(btnsActionListener);
```

```
rotateTextBtn.addActionListener(btnsActionListener);
groupComponentsBtn.addActionListener(btnsActionListener);
ungroupComponentsBtn.addActionListener(btnsActionListener);
regroupComponentsBtn.addActionListener(btnsActionListener);
alignRightBtn.addActionListener(btnsActionListener);
alignLeftBtn.addActionListener(btnsActionListener);
alignTopBtn.addActionListener(btnsActionListener);
alignBottomBtn.addActionListener(btnsActionListener);
serverFileManagerBtn.addActionListener(btnsActionListener);
handBtn.addActionListener(btnsActionListener);
dCursorBtn.addActionListener(btnsActionListener);
deletePageBtn.addActionListener(btnsActionListener);
fitZoomBtn.addActionListener(btnsActionListener);
runBtn.addActionListener(btnsActionListener);
insertTextBtn.addActionListener(btnsActionListener);
fillColorBtn.addActionListener(btnsActionListener);
lineColorBtn.addActionListener(btnsActionListener);
//canvasPropertiesBtn.addActionListener(btnsActionListener);
dftCursorBtn.addActionListener(btnsActionListener);
rectangleBtn.addActionListener(btnsActionListener);
circleBtn.addActionListener(btnsActionListener);
lineBtn.addActionListener(btnsActionListener);
goToBtn.addActionListener(btnsActionListener);
canvasColorsBtn.addActionListener(btnsActionListener);
canvasSizeBtn.addActionListener(btnsActionListener);
lastBackwardBtn.addActionListener(btnsActionListener);
backwardBtn.addActionListener(btnsActionListener);
forwardBtn.addActionListener(btnsActionListener);
lastForwardBtn.addActionListener(btnsActionListener);
refreshBtn.addActionListener(btnsActionListener);
gridBtn.addActionListener(btnsActionListener);
simulationSettingsBtn.addActionListener(btnsActionListener);
createNetlistBtn.addActionListener(btnsActionListener);
new1Btn.addActionListener(btnsActionListener);
new2Btn.addActionListener(btnsActionListener);
new3Btn.addActionListener(btnsActionListener);
new4Btn.addActionListener(btnsActionListener);
new5Btn.addActionListener(btnsActionListener);
VResistorBtn.addActionListener(btnsActionListener);
VCapacitorBtn.addActionListener(btnsActionListener);
VInductorBtn.addActionListener(btnsActionListener);
VPPTransformerBtn.addActionListener(btnsActionListener);
VPNTransformerBtn.addActionListener(btnsActionListener);
VNPTransformerBtn.addActionListener(btnsActionListener);
VNNTransformerBtn.addActionListener(btnsActionListener);
```

```
VDCVoltageBtn.addActionListener(btnsActionListener);
VDCCurrentBtn.addActionListener(btnsActionListener);
VACVoltageBtn.addActionListener(btnsActionListener);
VACCurrentBtn.addActionListener(btnsActionListener);
VVCVSBtn.addActionListener(btnsActionListener);
VVCCSBtn.addActionListener(btnsActionListener);
VCCVSBtn.addActionListener(btnsActionListener);
VCCCSBtn.addActionListener(btnsActionListener);
VConnLineBtn.addActionListener(btnsActionListener);
VConnNodeBtn.addActionListener(btnsActionListener);
VGroundBtn.addActionListener(btnsActionListener);
VCurrentMarkerBtn.addActionListener(btnsActionListener);
VVoltageMarkerBtn.addActionListener(btnsActionListener);
textColorBtn.addActionListener(btnsActionListener);
roundRectangleBtn.addActionListener(btnsActionListener);
showVoltagesBtn.addActionListener(btnsActionListener);
showCurrentsBtn.addActionListener(btnsActionListener);
editNetlistBtn.addActionListener(btnsActionListener);
showOutputResultBtn.addActionListener(btnsActionListener);
showHideNodesBtn.addActionListener(btnsActionListener);
showCIRMLBtn.addActionListener(btnsActionListener);
voltageMarkerBtn.addActionListener(btnsActionListener);
currentMarkerBtn.addActionListener(btnsActionListener);
drawResultBtn.addActionListener(btnsActionListener);
voltageDifferentialMarkerBtn.addActionListener(btnsActionListener);
clearAllCompsBtn.addActionListener(btnsActionListener);
//************MENUS LISTENERS**************//

enterValue.addActionListener(menusActionListener);
singleElementRotateRight.addActionListener(menusActionListener);
singleElementRotateLeft.addActionListener(menusActionListener);
singleElementDelete.addActionListener(menusActionListener);
singleElementCut.addActionListener(menusActionListener);
multiElementsRotateLeft.addActionListener(menusActionListener);
multiElementsRotateRight.addActionListener(menusActionListener);
multiElementsDelete.addActionListener(menusActionListener);
deleteLine.addActionListener(menusActionListener);
copyLine.addActionListener(menusActionListener);
cutLine.addActionListener(menusActionListener);
undo.addActionListener(menusActionListener);
menuItem10.addActionListener(menusActionListener);
singleElementCopy.addActionListener(menusActionListener);
multiElementsCopy.addActionListener(menusActionListener);
multiElementsCut.addActionListener(menusActionListener);
multiElementsGroup.addActionListener(menusActionListener);
```

```
multiElementsUngroup.addActionListener(menusActionListener);
paste.addActionListener(menusActionListener);
clearAll.addActionListener(menusActionListener);
zoomIn.addActionListener(menusActionListener);
zoomOut.addActionListener(menusActionListener);
cutLine.addActionListener(menusActionListener);
copyLine.addActionListener(menusActionListener);
deleteLine.addActionListener(menusActionListener);
singleElementFlipVertical.addActionListener(menusActionListener);
singleElementFlipHorizontal.addActionListener(menusActionListener);
singleElementDisplayProperties.addActionListener(menusActionListener);
singleElementZoomIn.addActionListener(menusActionListener);
singleElementZoomOut.addActionListener(menusActionListener);
singleElementGoTo.addActionListener(menusActionListener);
multiElementsDisplayProperties.addActionListener(menusActionListener);
multiElementsFlipVertical.addActionListener(menusActionListener);
multiElementsFlipHorizontal.addActionListener(menusActionListener);
multiElementsRegroup.addActionListener(menusActionListener);
multiElementsAlignTop.addActionListener(menusActionListener);
multiElementsAlignLeft.addActionListener(menusActionListener);
multiElementsAlignRight.addActionListener(menusActionListener);
multiElementsAlignBottom.addActionListener(menusActionListener);
multiElementsZoomIn.addActionListener(menusActionListener);
multiElementsZoomOut.addActionListener(menusActionListener);
multiElementsGoTo.addActionListener(menusActionListener);
goToLine.addActionListener(menusActionListener);
goToPage.addActionListener(menusActionListener);
newMenuItem.addActionListener(menusActionListener);
openMenuItem.addActionListener(menusActionListener);
saveMenuItem.addActionListener(menusActionListener);
saveAsMenuItem.addActionListener(menusActionListener);
closePageMenuItem.addActionListener(menusActionListener);
cutMenuItem.addActionListener(menusActionListener);
copyMenuItem.addActionListener(menusActionListener);
pasteMenuItem.addActionListener(menusActionListener);
deleteMenuItem.addActionListener(menusActionListener);
selectAllMenuItem.addActionListener(menusActionListener);
undoMenuItem.addActionListener(menusActionListener);
redoMenuItem.addActionListener(menusActionListener);
zoomInMenuItem.addActionListener(menusActionListener);
fitZoomMenuItem.addActionListener(menusActionListener);
zoomOutMenuItem.addActionListener(menusActionListener);
scrollHandMenuItem.addActionListener(menusActionListener);
defaultCursorMenuItem.addActionListener(menusActionListener);
helpMenuItem.addActionListener(menusActionListener);
```

```
refreshMenuItem.addActionListener(menusActionListener);
standardBarMenuItem.addActionListener(menusActionListener);
visitedElementsBarMenuItem.addActionListener(menusActionListener);
pageExplorerBarMenuItem.addActionListener(menusActionListener);
statusBarMenuItem.addActionListener(menusActionListener);
canvasBarMenuItem.addActionListener(menusActionListener);
componentBarMenuItem.addActionListener(menusActionListener);
paintingBarMenuItem.addActionListener(menusActionListener);
acknowledgmentMenuItem.addActionListener(menusActionListener);
contactUsMenuItem.addActionListener(menusActionListener);
exitMenuItem.addActionListener(menusActionListener);
simulationBarMenuItem.addActionListener(menusActionListener);
canvasColorsMenuItem.addActionListener(menusActionListener);
canvasSizeMenuItem.addActionListener(menusActionListener);
voltageMarkerMenuItem.addActionListener(menusActionListener);
diffVoltageMarkerMenuItem.addActionListener(menusActionListener);
currentMarkerMenuItem.addActionListener(menusActionListener);
hideAllMarkersMenuItem.addActionListener(menusActionListener);
showAllMarkersMenuItem.addActionListener(menusActionListener);
lastPreviousMenuItem.addActionListener(menusActionListener);
previousMenuItem.addActionListener(menusActionListener);
nextMenuItem.addActionListener(menusActionListener);
lastNextMenuItem.addActionListener(menusActionListener);
goToMenuItem.addActionListener(menusActionListener);
editPageInfoMenuItem.addActionListener(menusActionListener);
drawTextMenuItem.addActionListener(menusActionListener);
drawLineMenuItem.addActionListener(menusActionListener);
drawCircleMenuItem.addActionListener(menusActionListener);
drawRectangleMenuItem.addActionListener(menusActionListener);
drawRoundRectangleMenuItem.addActionListener(menusActionListener);
displayPropertiesMenuItem.addActionListener(menusActionListener);
editPropertiesMenuItem.addActionListener(menusActionListener);
rotateRightMenuItem.addActionListener(menusActionListener);
rotateLeftMenuItem.addActionListener(menusActionListener);
flipVerticalMenuItem.addActionListener(menusActionListener);
flipHorizontalMenuItem.addActionListener(menusActionListener);
rotateTextMenuItem.addActionListener(menusActionListener);
alignRightMenuItem.addActionListener(menusActionListener);
alignLeftMenuItem.addActionListener(menusActionListener);
alignTopMenuItem.addActionListener(menusActionListener);
alignBottomMenuItem.addActionListener(menusActionListener);
groupMenuItem.addActionListener(menusActionListener);
ungroupMenuItem.addActionListener(menusActionListener);
regroupMenuItem.addActionListener(menusActionListener);
createNetlistMenuItem.addActionListener(menusActionListener);
```

```
    editNetlistMenuItem.addActionListener(menusActionListener);
    simulationSettingsMenuItem.addActionListener(menusActionListener);
    runMenuItem.addActionListener(menusActionListener);
    showNodesNetNamesMenuItem.addActionListener(menusActionListener);
    showNodesVoltagesMenuItem.addActionListener(menusActionListener);
    showNodesCurrentsMenuItem.addActionListener(menusActionListener);
    viewCIRMLMenuItem.addActionListener(menusActionListener);
    viewSimulationResultMenuItem.addActionListener(menusActionListener);
    drawResultMenuItem.addActionListener(menusActionListener);
    showGridMenuItem.addActionListener(menusActionListener);
    showConnectionPointsMenuItem.addActionListener(menusActionListener);
    clearAllMenuItem.addActionListener(menusActionListener);
    copyPageMenuItem.addActionListener(menusActionListener);
    pastePageMenuItem.addActionListener(menusActionListener);
    selectPageMenuItem.addActionListener(menusActionListener);
    findMenuItem.addActionListener(menusActionListener);
    replaceMenuItem.addActionListener(menusActionListener);
    duplicateMenuItem.addActionListener(menusActionListener);
    page1MenuItem.addActionListener(menusActionListener);
    page2MenuItem.addActionListener(menusActionListener);
    page3MenuItem.addActionListener(menusActionListener);
    page4MenuItem.addActionListener(menusActionListener);
    page5MenuItem.addActionListener(menusActionListener);
    redrawMenuItem.addActionListener(menusActionListener);
    fillColorMenuItem.addActionListener(menusActionListener);
    lineColorMenuItem.addActionListener(menusActionListener);
    textColorMenuItem.addActionListener(menusActionListener);

    //************POPUP MENU LISTENERS*******//

//.....................................................................//

    elementsPanel.addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent me) {
        elementsPanel_mouse_dragged(me);
      }
    });
//.....................................................................//
    elementsPanel.addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent me) {
        elementsPanel_mouse_pressed(me);
      }
    });
//.....................................................................//
    elementsPanel.addMouseListener(new MouseAdapter() {
```

```
      public void mouseReleased(MouseEvent me) {
        elementsPanel_mouse_released(me);
      }
    });
//.....................................................................//
    elementsPanel.addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent me) {
        elementsPanel_mouse_clicked(me);
      }
    });
//.....................................................................//
    class MyAdjustmentListener
        implements AdjustmentListener {
      // This method is called whenever the value of a scrollbar is changed,
      // either by the user or programmatically.
      public void adjustmentValueChanged(AdjustmentEvent evt) {
        Adjustable source = evt.getAdjustable();

        // getValueIsAdjusting() returns true if the user is currently
        // dragging the scrollbar's knob and has not picked a final value
        if (evt.getValueIsAdjusting()) {
          // The user is dragging the knob

          return;
        }

        // Determine which scrollbar fired the event
        int orient = source.getOrientation();
        if (orient == Adjustable.HORIZONTAL) {
          startPoint[0] = evt.getValue();

        }
        else {
          startPoint[1] = evt.getValue();

          // Event from vertical scrollbar
        }

        // Determine the type of event
        int type = evt.getAdjustmentType();

        switch (type) {
          case AdjustmentEvent.UNIT_INCREMENT:

            // Scrollbar was increased by one unit
```

```
          break;
        case AdjustmentEvent.UNIT_DECREMENT:

          // Scrollbar  decreased by one unit
          break;
        case AdjustmentEvent.BLOCK_INCREMENT:

          // Scrollbar. was increased by one block
          break;
        case AdjustmentEvent.BLOCK_DECREMENT:

          // Scrollbar was decreased by one block
          break;
        case AdjustmentEvent.TRACK:

          // The knob on the scrollbar was dragged
          break;
      }

      // Get current value
      if (start) {

        redrawGridCanvas();

      }
    }
  }

  AdjustmentListener listener = new MyAdjustmentListener();
  mainScroll.getHorizontalScrollBar().addAdjustmentListener(listener);
  mainScroll.getVerticalScrollBar().addAdjustmentListener(listener);

//*************************************************************************/
/
//                    END OF ADDING LISTINERS                    //
//*************************************************************************/
/

 }

//*************************************************************************/
/
//                    EVENTS HANDLING FUNCTIONS                  //
//*************************************************************************/
/
```

```
private void btnsActionPerformed(ActionEvent ae) {
  JButton btn = (JButton) ae.getSource();
  String command = btn.getActionCommand();

  if (command.equals("passiveBtn")) {
    passiveBtn_action_performed(ae);
  }
  else if (command.equals("independentBtn")) {
    independentBtn_action_performed(ae);
  }
  if (command.equals("dependentBtn")) {
    dependentBtn_action_performed(ae);
  }
  else if (command.equals("connectionsBtn")) {
    connectionsBtn_action_performed(ae);
  }
  if (command.equals("netListBtn")) {
    netListBtn_action_performed(ae);
  }
  if (command.equals("createNetlistBtn")) {
    netListBtn_action_performed(ae);
  }
  if (command.equals("editNetlistBtn")) {
    editNetlistBtn_action_performed(ae);
  }
  if (command.equals("showHideNodesBtn")) {
    showHideNodesBtn_action_performed(ae);
  }
  if (command.equals("showCIRMLBtn")) {
    showCIRMLBtn_action_performed(ae);
  }
  if (command.equals("voltageMarkerBtn")) {
    voltageMarkerBtn_action_performed(ae);
  }
  if (command.equals("currentMarkerBtn")) {
    currentMarkerBtn_action_performed(ae);
  }
  if (command.equals("drawResultBtn")) {
    drawResultBtn_action_performed(ae);
  }
  if (command.equals("voltageDifferentialMarkerBtn")) {
    voltageDifferentialMarkerBtn_action_performed(ae);
  }
  if (command.equals("showVoltagesBtn")) {
    showVoltagesBtn_action_performed(ae);
```

```
}
if (command.equals("showCurrentsBtn")) {
  showCurrentsBtn_action_performed(ae);
}
if (command.equals("showOutputResultBtn")) {
  showOutputResultBtn_action_performed(ae);
}
else if (command.equals("simulateBtn")) {
  simulateBtn_action_performed(ae);
}
else if (command.equals("netlistEditingBtn")) {
  netlistEditingBtn_action_performed(ae);
}
else if (command.equals("simSettingsBtn")) {
  simSettingsBtn_action_performed(ae);
}
if (command.equals("fileManagerBtn")) {
  fileManagerBtn_action_performed(ae);
}
if (command.equals("deleteSchematicBtn")) {
  deleteSchematicBtn_action_performed(ae);
}
else if (command.equals("clearBtn")) {
  clearBtn_action_performed(ae);
}
if (command.equals("saveBtn")) {
  saveBtn_action_performed(ae);
}
if (command.equals("clearAllCompsBtn")) {
  clearAllCompsBtn_action_performed(ae);
}
else if (command.equals("loadBtn")) {
  loadBtn_action_performed(ae);
}
if (command.equals("newSchematicBtn")) {
  newSchematicBtn_action_performed(ae);
}
else if (command.equals("saveSchematicBtn")) {
  saveSchematicBtn_action_performed(ae);
}
if (command.equals("openSchematicBtn")) {
  loadBtn_action_performed(ae);
}
else if (command.equals("zoomInBtn")) {
  zoomInBtn_action_performed(ae);
```

```
}
if (command.equals("zoomOutBtn")) {
  zoomOutBtn_action_performed(ae);
}
else if (command.equals("newBtn")) {
  newBtn_action_performed(ae);
}
else if (command.equals("cutBtn")) {
  cutBtn_action_performed(ae);
}
else if (command.equals("copyBtn")) {
  copyBtn_action_performed(ae);
}
else if (command.equals("pasteBtn")) {
  pasteBtn_action_performed(ae);
}
else if (command.equals("undoBtn")) {
  undoBtn_action_performed(ae);
}
else if (command.equals("redoBtn")) {
  redoBtn_action_performed(ae);
}
else if (command.equals("helpBtn")) {
  helpBtn_action_performed(ae);
}
else if (command.equals("displayPropertiesBtn")) {
  displayPropertiesBtn_action_performed(ae);
}
else if (command.equals("editComponentBtn")) {
  editComponentBtn_action_performed(ae);
}
else if (command.equals("rotateRightBtn")) {
  rotateRightBtn_action_performed(ae);
}
else if (command.equals("rotateLeftBtn")) {
  rotateLeftBtn_action_performed(ae);
}
else if (command.equals("flipVerticalBtn")) {
  flipVerticalBtn_action_performed(ae);
}
else if (command.equals("flipHorizontalBtn")) {
  flipHorizontalBtn_action_performed(ae);
}
else if (command.equals("rotateTextBtn")) {
  rotateTextBtn_action_performed(ae);
```

```
}
else if (command.equals("groupComponentsBtn")) {
  groupComponentsBtn_action_performed(ae);
}
else if (command.equals("regroupComponentsBtn")) {
  regroupComponentsBtn_action_performed(ae);
}
else if (command.equals("ungroupComponentsBtn")) {
  ungroupComponentsBtn_action_performed(ae);
}
else if (command.equals("alignRightBtn")) {
  alignRightBtn_action_performed(ae);
}
else if (command.equals("alignLeftBtn")) {
  alignLeftBtn_action_performed(ae);
}
else if (command.equals("alignBottomBtn")) {
  alignBottomBtn_action_performed(ae);
}
else if (command.equals("alignTopBtn")) {
  alignTopBtn_action_performed(ae);
}
else if (command.equals("serverFileManagerBtn")) {
  serverFileManagerBtn_action_performed(ae);
}
else if (command.equals("deleteBtn")) {
  deleteBtn_action_performed(ae);
}
else if (command.equals("contactUsBtn")) {
  contactUsBtn_action_performed(ae);
}
else if (command.equals("selectAllCompsBtn")) {
  selectAllCompsBtn_action_performed(ae);
}
else if (command.equals("handBtn")) {
  handBtn_action_performed(ae);
}
else if (command.equals("dCursorBtn")) {
  dCursorBtn_action_performed(ae);
}
else if (command.equals("deletePageBtn")) {
  deletePageBtn_action_performed(ae);
}
else if (command.equals("fitZoomBtn")) {
  fitZoomBtn_action_performed(ae);
```

```
}
else if (command.equals("runBtn")) {
  runBtn_action_performed(ae);
}
else if (command.equals("insertTextBtn")) {
  insertTextBtn_action_performed(ae);
}
else if (command.equals("rectangleBtn")) {
  rectangleBtn_action_performed(ae);
}
else if (command.equals("circleBtn")) {
  circleBtn_action_performed(ae);
}
else if (command.equals("fillColorBtn")) {
  fillColorBtn_action_performed(ae);
}
else if (command.equals("lineColorBtn")) {
  lineColorBtn_action_performed(ae);
}
else if (command.equals("dftCursorBtn")) {
  dftCursorBtn_action_performed(ae);
}
else if (command.equals("canvasPropertiesBtn")) {
  canvasPropertiesBtn_action_performed(ae);
}
else if (command.equals("lineBtn")) {
  lineBtn_action_performed(ae);
}
else if (command.equals("goToBtn")) {
  goToBtn_action_performed(ae);
}
else if (command.equals("canvasSizeBtn")) {
  canvasSizeBtn_action_performed(ae);
}
else if (command.equals("canvasColorsBtn")) {
  canvasColorsBtn_action_performed(ae);
}
else if (command.equals("lastForwardBtn")) {
  lastForwardBtn_action_performed(ae);
}
else if (command.equals("forwardBtn")) {
  forwardBtn_action_performed(ae);
}
else if (command.equals("backwardBtn")) {
  backwardBtn_action_performed(ae);
```

```
    }
    else if (command.equals("lastBackwardBtn")) {
      lastBackwardBtn_action_performed(ae);
    }
    else if (command.equals("refreshBtn")) {
      refreshBtn_action_performed(ae);
    }
    else if (command.equals("gridBtn")) {
      gridBtn_action_performed(ae);
    }
    else if (command.equals("simulationSettingsBtn")) {
      simulationSettingsBtn_action_performed(ae);
    }
    else if (command.equals("new1Btn")) {
      new1Btn_action_performed(ae);
    }
    else if (command.equals("new2Btn")) {
      new2Btn_action_performed(ae);
    }
    else if (command.equals("new3Btn")) {
      new3Btn_action_performed(ae);
    }
    else if (command.equals("new4Btn")) {
      new4Btn_action_performed(ae);
    }
    else if (command.equals("new5Btn")) {
      new5Btn_action_performed(ae);
    }
    else if (command.equals("textColorBtn")) {
      textColorBtn_action_performed(ae);
    }
    else if (command.equals("roundRectangleBtn")) {
      roundRectangleBtn_action_performed(ae);
    }
    else if (command.equals("arcBtn")) {
      arcBtn_action_performed(ae);
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void menusActionPerformed(ActionEvent ae) {

    JMenuItem menuItem = (JMenuItem) ae.getSource();
    String command = menuItem.getActionCommand();
```

```
if (command.equals("enterValue")) {
  enterValue_action_performed(ae);
}
else if (command.equals("singleElementRotateRight")) {
  singleElementRotateRight_action_performed(ae);
}
else if (command.equals("singleElementRotateLeft")) {
  singleElementRotateLeft_action_performed(ae);
}
else if (command.equals("singleElementDelete")) {
  singleElementDelete_action_performed(ae);
}
else if (singleElementCut.equals("singleElementCut")) {
  singleElementCut_action_performed(ae);
}
else if (command.equals("multiElementsRotateLeft")) {
  multiElementsRotateLeft_action_performed(ae);
}
else if (command.equals("multiElementsRotateRight")) {
  multiElementsRotateRight_action_performed(ae);
}
else if (command.equals("multiElementsDelete")) {
  multiElementsDelete_action_performed(ae);
}
else if (command.equals("deleteLine")) {
  deleteLine_action_performed(ae);
}
else if (command.equals("copyLine")) {
  copyLine_action_performed(ae);
}
else if (command.equals("cutLine")) {
  cutLine_action_performed(ae);
}
else if (command.equals("undo")) {
  undo_action_performed(ae);
}
else if (command.equals("menuItem10")) {
  menuItem10_action_performed(ae);
}
else if (command.equals("singleElementCopy")) {
  singleElementCopy_action_performed(ae);
}
else if (command.equals("multiElementsCopy")) {
  multiElementsCopy_action_performed(ae);
```

```
}
else if (command.equals("multiElementsCut")) {
  multiElementsCut_action_performed(ae);
}
else if (command.equals("multiElementsGroup")) {
  multiElementsGroup_action_performed(ae);
}
else if (command.equals("multiElementsUngroup")) {
  multiElementsUngroup_action_performed(ae);
}
else if (command.equals("paste")) {
  paste_action_performed(ae);
}
else if (command.equals("clearAll")) {
  clearAll_action_performed(ae);
}
else if (command.equals("zoomIn")) {
  zoomIn_action_performed(ae);
}
else if (command.equals("zoomOut")) {
  zoomOut_action_performed(ae);
}
else if (command.equals("cutLine")) {
  cutLine_action_performed(ae);
}
else if (command.equals("copyLine")) {
  copyLine_action_performed(ae);
}
else if (command.equals("deleteLine")) {
  deleteLine_action_performed(ae);
}
else if (command.equals("singleElementFlipVertical")) {
  singleElementFlipVertical_action_performed(ae);
}
else if (command.equals("singleElementFlipHorizontal")) {
  singleElementFlipHorizontal_action_performed(ae);
}
else if (command.equals("singleElementDisplayProperties")) {
  singleElementDisplayProperties_action_performed(ae);
}
else if (command.equals("singleElementZoomIn")) {
  singleElementZoomIn_action_performed(ae);
}
else if (command.equals("singleElementZoomOut")) {
  singleElementZoomOut_action_performed(ae);
```

```
    }
    else if (command.equals("singleElementGoTo")) {
      singleElementGoTo_action_performed(ae);
    }
    else if (command.equals("multiElementsDisplayProperties")) {
      multiElementsDisplayProperties_action_performed(ae);
    }
    else if (command.equals("multiElementsFlipVertical")) {
      multiElementsFlipVertical_action_performed(ae);
    }
    else if (command.equals("multiElementsFlipHorizontal")) {
      multiElementsFlipHorizontal_action_performed(ae);
    }
    else if (command.equals("multiElementsRegroup")) {
      multiElementsRegroup_action_performed(ae);
    }
    else if (command.equals("multiElementsAlignTop")) {
      multiElementsAlignTop_action_performed(ae);
    }
    else if (command.equals("multiElementsAlignLeft")) {
      multiElementsAlignLeft_action_performed(ae);
    }
    else if (command.equals("multiElementsAlignRight")) {
      multiElementsAlignRight_action_performed(ae);
    }
    else if (command.equals("multiElementsAlignBottom")) {
      multiElementsAlignBottom_action_performed(ae);
    }
    else if (command.equals("multiElementsZoomIn")) {
      multiElementsZoomIn_action_performed(ae);
    }
    else if (command.equals("multiElementsZoomOut")) {
      multiElementsZoomOut_action_performed(ae);
    }
    else if (command.equals("multiElementsGoTo")) {
      multiElementsGoTo_action_performed(ae);
    }
    else if (command.equals("goToLine")) {
      goToLine_action_performed(ae);
    }
    else if (command.equals("goToPage")) {
      goToPage_action_performed(ae);

//////////////////////////////////////////////////////////////////////////
    }
```

```
else if (command.equals("newMenuItem")) {
  newMenuItem_action_performed(ae);
}
else if (command.equals("openMenuItem")) {
  openMenuItem_action_performed(ae);
}
else if (command.equals("saveMenuItem")) {
  saveMenuItem_action_performed(ae);
}
else if (command.equals("saveAsMenuItem")) {
  saveAsMenuItem_action_performed(ae);
}
else if (command.equals("closePageMenuItem")) {
  closePageMenuItem_action_performed(ae);
}
else if (command.equals("cutMenuItem")) {
  cutMenuItem_action_performed(ae);
}
else if (command.equals("copyMenuItem")) {
  copyMenuItem_action_performed(ae);
}
else if (command.equals("pasteMenuItem")) {
  pasteMenuItem_action_performed(ae);
}
else if (command.equals("deleteMenuItem")) {
  deleteMenuItem_action_performed(ae);
}
else if (command.equals("selectAllMenuItem")) {
  selectAllMenuItem_action_performed(ae);
}
else if (command.equals("undoMenuItem")) {
  undoMenuItem_action_performed(ae);
}
else if (command.equals("redoMenuItem")) {
  redoMenuItem_action_performed(ae);
}
else if (command.equals("zoomInMenuItem")) {
  zoomInMenuItem_action_performed(ae);
}
else if (command.equals("fitZoomMenuItem")) {
  fitZoomMenuItem_action_performed(ae);
}
else if (command.equals("zoomOutMenuItem")) {
  zoomOutMenuItem_action_performed(ae);
}
```

```
else if (command.equals("scrollHandMenuItem")) {
  scrollHandMenuItem_action_performed(ae);
}
else if (command.equals("defaultCursorMenuItem")) {
  defaultCursorMenuItem_action_performed(ae);
}
else if (command.equals("helpMenuItem")) {
  helpMenuItem_action_performed(ae);
}
else if (command.equals("refreshMenuItem")) {
  refreshMenuItem_action_performed(ae);
}
else if (command.equals("standardBarMenuItem")) {
  standardBarMenuItem_action_performed(ae);
}
else if (command.equals("visitedElementsBarMenuItem")) {
  visitedElementsBarMenuItem_action_performed(ae);
}
else if (command.equals("pageExplorerBarMenuItem")) {
  pageExplorerBarMenuItem_action_performed(ae);
}
else if (command.equals("statusBarMenuItem")) {
  statusBarMenuItem_action_performed(ae);
}
else if (command.equals("canvasBarMenuItem")) {
  canvasBarMenuItem_action_performed(ae);
}
else if (command.equals("componentBarMenuItem")) {
  componentBarMenuItem_action_performed(ae);
}
else if (command.equals("paintingBarMenuItem")) {
  paintingBarMenuItem_action_performed(ae);
}
else if (command.equals("acknowledgmentMenuItem")) {
  acknowledgmentMenuItem_action_performed(ae);
}
else if (command.equals("contactUsMenuItem")) {
  contactUsMenuItem_action_performed(ae);
}
else if (command.equals("exitMenuItem")) {
  exitMenuItem_action_performed(ae);
}
else if (command.equals("simulationBarMenuItem")) {
  simulationBarMenuItem_action_performed(ae);
}
```

```
else if (command.equals("canvasColorsMenuItem")) {
  canvasColorsMenuItem_action_performed(ae);
}
else if (command.equals("canvasSizeMenuItem")) {
  canvasSizeMenuItem_action_performed(ae);
}
else if (command.equals("voltageMarkerMenuItem")) {
  voltageMarkerMenuItem_action_performed(ae);
}
else if (command.equals("diffVoltageMarkerMenuItem")) {
  diffVoltageMarkerMenuItem_action_performed(ae);
}
else if (command.equals("currentMarkerMenuItem")) {
  currentMarkerMenuItem_action_performed(ae);
}
else if (command.equals("hideAllMarkersMenuItem")) {
  hideAllMarkersMenuItem_action_performed(ae);
}
else if (command.equals("showAllMarkersMenuItem")) {
  showAllMarkersMenuItem_action_performed(ae);
}
else if (command.equals("lastPreviousMenuItem")) {
  lastPreviousMenuItem_action_performed(ae);
}
else if (command.equals("previousMenuItem")) {
  previousMenuItem_action_performed(ae);
}
else if (command.equals("nextMenuItem")) {
  nextMenuItem_action_performed(ae);
}
else if (command.equals("lastNextMenuItem")) {
  lastNextMenuItem_action_performed(ae);
}
else if (command.equals("goToMenuItem")) {
  goToMenuItem_action_performed(ae);
}
else if (command.equals("editPageInfoMenuItem")) {
  editPageInfoMenuItem_action_performed(ae);
}
else if (command.equals("drawTextMenuItem")) {
  drawTextMenuItem_action_performed(ae);
}
else if (command.equals("drawLineMenuItem")) {
  drawLineMenuItem_action_performed(ae);
}
```

```
else if (command.equals("drawCircleMenuItem")) {
  drawCircleMenuItem_action_performed(ae);
}
else if (command.equals("drawRectangleMenuItem")) {
  drawRectangleMenuItem_action_performed(ae);
}
else if (command.equals("drawRoundRectangleMenuItem")) {
  drawRoundRectangleMenuItem_action_performed(ae);
}
else if (command.equals("displayPropertiesMenuItem")) {
  displayPropertiesMenuItem_action_performed(ae);
}
else if (command.equals("editPropertiesMenuItem")) {
  editPropertiesMenuItem_action_performed(ae);
}
else if (command.equals("rotateRightMenuItem")) {
  rotateRightMenuItem_action_performed(ae);
}
else if (command.equals("rotateLeftMenuItem")) {
  rotateLeftMenuItem_action_performed(ae);
}
else if (command.equals("flipVerticalMenuItem")) {
  flipVerticalMenuItem_action_performed(ae);
}
else if (command.equals("flipHorizontalMenuItem")) {
  flipHorizontalMenuItem_action_performed(ae);
}
else if (command.equals("rotateTextMenuItem")) {
  rotateTextMenuItem_action_performed(ae);
}
else if (command.equals("alignRightMenuItem")) {
  alignRightMenuItem_action_performed(ae);
}
else if (command.equals("alignLeftMenuItem")) {
  alignLeftMenuItem_action_performed(ae);
}
else if (command.equals("alignTopMenuItem")) {
  alignTopMenuItem_action_performed(ae);
}
else if (command.equals("alignBottomMenuItem")) {
  alignBottomMenuItem_action_performed(ae);
}
else if (command.equals("groupMenuItem")) {
  groupMenuItem_action_performed(ae);
}
```

```
else if (command.equals("ungroupMenuItem")) {
  ungroupMenuItem_action_performed(ae);
}
else if (command.equals("regroupMenuItem")) {
  regroupMenuItem_action_performed(ae);
}
else if (command.equals("createNetlistMenuItem")) {
  createNetlistMenuItem_action_performed(ae);
}
else if (command.equals("editNetlistMenuItem")) {
  editNetlistMenuItem_action_performed(ae);
}
else if (command.equals("simulationSettingsMenuItem")) {
  simulationSettingsMenuItem_action_performed(ae);
}
else if (command.equals("runMenuItem")) {
  runMenuItem_action_performed(ae);
}
else if (command.equals("showNodesNetNamesMenuItem")) {
  showNodesNetNamesMenuItem_action_performed(ae);
}
else if (command.equals("showNodesVoltagesMenuItem")) {
  showNodesVoltagesMenuItem_action_performed(ae);
}
else if (command.equals("showNodesCurrentsMenuItem")) {
  showNodesCurrentsMenuItem_action_performed(ae);
}
else if (command.equals("viewCIRMLMenuItem")) {
  viewCIRMLMenuItem_action_performed(ae);
}
else if (command.equals("viewSimulationResultMenuItem")) {
  viewSimulationResultMenuItem_action_performed(ae);
}
else if (command.equals("drawResultMenuItem")) {
  drawResultMenuItem_action_performed(ae);
}
else if (command.equals("showGridMenuItem")) {
  showGridMenuItem_action_performed(ae);
}
else if (command.equals("showConnectionPointsMenuItem")) {
  showConnectionPointsMenuItem_action_performed(ae);
}
else if (command.equals("clearAllMenuItem")) {
  clearAllMenuItem_action_performed(ae);
}
```

```
else if (command.equals("copyPageMenuItem")) {
  copyPageMenuItem_action_performed(ae);
}
else if (command.equals("pastePageMenuItem")) {
  pastePageMenuItem_action_performed(ae);
}
else if (command.equals("selectPageMenuItem")) {
  selectPageMenuItem_action_performed(ae);
}
else if (command.equals("findMenuItem")) {
  findMenuItem_action_performed(ae);
}
else if (command.equals("replaceMenuItem")) {
  replaceMenuItem_action_performed(ae);
}
else if (command.equals("duplicateMenuItem")) {
  duplicateMenuItem_action_performed(ae);
}
else if (command.equals("page1MenuItem")) {
  page1MenuItem_action_performed(ae);
}
else if (command.equals("page2MenuItem")) {
  page2MenuItem_action_performed(ae);
}
else if (command.equals("page3MenuItem")) {
  page3MenuItem_action_performed(ae);
}
else if (command.equals("page4MenuItem")) {
  page4MenuItem_action_performed(ae);
}
else if (command.equals("page5MenuItem")) {
  page5MenuItem_action_performed(ae);
}
else if (command.equals("redrawMenuItem")) {
  redrawMenuItem_action_performed(ae);
}
else if (command.equals("fillColorMenuItem")) {
  fillColorMenuItem_action_performed(ae);
}
else if (command.equals("lineColorMenuItem")) {
  lineColorMenuItem_action_performed(ae);
}
else if (command.equals("textColorMenuItem")) {
  textColorMenuItem_action_performed(ae);
}
```

```
  }

////////////////////////////////////////////////////////////////////////////

  //************SIMULATION EVENT FUNCTIONS*******//
  private void netListBtn_action_performed(ActionEvent ae) {
    generateNetList();

  }

////////////////////////////////////////////////////////////////////////////

  private void editNetlistBtn_action_performed(ActionEvent ae) {
    showEditNetlistFrame();

  }

////////////////////////////////////////////////////////////////////////////
  private void showHideNodesBtn_action_performed(ActionEvent ae) {

    if (!ActiveState.IsNetlistNodesShown) {

      ActiveState.IsNetlistNodesShown = true;
    }
    else {

      ActiveState.IsNetlistNodesShown = false;
    }
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void showCIRMLBtn_action_performed(ActionEvent ae) {
    showCIRMLFrame();
  }

////////////////////////////////////////////////////////////////////////////
  private void voltageMarkerBtn_action_performed(ActionEvent ae) {
    ActiveState.CurComp = ActiveState.createNew(ActiveState.VOLTAGEMARK);
  }

////////////////////////////////////////////////////////////////////////////
  private void currentMarkerBtn_action_performed(ActionEvent ae) {
    ActiveState.CurComp = ActiveState.createNew(ActiveState.CURRENTMARK);
  }
```

```
////////////////////////////////////////////////////////////////////////////
  private void drawResultBtn_action_performed(ActionEvent ae) {
    drawGraphicalResults();
  }

////////////////////////////////////////////////////////////////////////////
  private void voltageDifferentialMarkerBtn_action_performed(ActionEvent ae) {
    ActiveState.CurComp = ActiveState.createNew(ActiveState.NODEL);
  }

////////////////////////////////////////////////////////////////////////////
  private void showVoltagesBtn_action_performed(ActionEvent ae) {
    if (!ActiveState.IsNodesVoltagesShown) {

      ActiveState.IsNodesVoltagesShown = true;
    }
    else {

      ActiveState.IsNodesVoltagesShown = false;
    }
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void showCurrentsBtn_action_performed(ActionEvent ae) {
    if (!ActiveState.IsNodesCurrentsShown) {

      ActiveState.IsNodesCurrentsShown = true;
    }
    else {

      ActiveState.IsNodesCurrentsShown = false;
    }
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void showOutputResultBtn_action_performed(ActionEvent ae) {

    showSimulationResultFrame();
  }

////////////////////////////////////////////////////////////////////////////
  private void simulateBtn_action_performed(ActionEvent ae) {
```

```
    if (generateNetList()) {
      runSimulation();
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void netlistEditingBtn_action_performed(ActionEvent ae) {
    showEditNetlistFrame();

  }

////////////////////////////////////////////////////////////////////////////
  class MyFilter
      extends javax.swing.filechooser.FileFilter {
    public boolean accept(File file) {
      String filename = file.getName();
      return filename.endsWith(".sch");
    }

    public String getDescription() {
      return "*.sch";
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void saveBtn_action_performed(ActionEvent ae) {
    saveSchematicToLocal();

  }

////////////////////////////////////////////////////////////////////////////
  private void clearAllCompsBtn_action_performed(ActionEvent ae) {
    clearCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void saveSchematicBtn_action_performed(ActionEvent ae) {
    saveSchematicToLocal();
  }

////////////////////////////////////////////////////////////////////////////
  private void newSchematicBtn_action_performed(ActionEvent ae) {
    createNewPage();
  }
```

```
////////////////////////////////////////////////////////////////////////////

  private void clearBtn_action_performed(ActionEvent ae) {

    clearCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void loadBtn_action_performed(ActionEvent ae) {
    openLocalSchematic();

  }

////////////////////////////////////////////////////////////////////////////
  private void zoomInBtn_action_performed(ActionEvent ae) {
    ActiveState.reSet();
    ActiveState.zoomInCmd = true;
  }

////////////////////////////////////////////////////////////////////////////
  private void zoomOutBtn_action_performed(ActionEvent ae) {
    ActiveState.reSet();
    ActiveState.zoomOutCmd = true;

    Graphics2D G = (Graphics2D) gridCanvas.gridImage;
    if (ActiveState.currentXScale < 1 && ActiveState.currentXScale > 0.8) {
      ActiveState.currentXScale -= 0.1;
    }
    else {
      ActiveState.currentXScale = 1;
      ActiveState.currentXScale -= 0.1;
    }
    if (ActiveState.currentYScale < 1 && ActiveState.currentYScale > 0.8) {
      ActiveState.currentYScale -= 0.1;
    }
    else {
      ActiveState.currentYScale = 1;
      ActiveState.currentYScale -= 0.1;
    }
    G.scale(ActiveState.currentXScale, ActiveState.currentYScale);
    int[] point = {
        startPoint[0], startPoint[1]};
    gridCanvas.gridImage.clearRect(point[0], point[1], 2500, 2500);
    for (int x = point[0]; x < point[0] + 2500; x += 25) {
```

```
      for (int y = point[1]; y < point[1] + 2500; y += 25) {
        gridCanvas.gridImage.drawLine(x, y, x, y);
       }
     }

    drawAllComps();
    gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  private void newBtn_action_performed(ActionEvent ae) {
    createNewPage();
  }

////////////////////////////////////////////////////////////////////////////
  private void cutBtn_action_performed(ActionEvent ae) {
    cutComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void copyBtn_action_performed(ActionEvent ae) {
    copyComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void pasteBtn_action_performed(ActionEvent ae) {
    pasteComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void undoBtn_action_performed(ActionEvent ae) {
    undoActions();
  }

////////////////////////////////////////////////////////////////////////////
  private void redoBtn_action_performed(ActionEvent ae) {
    redoActions();
  }

////////////////////////////////////////////////////////////////////////////
  private void deleteBtn_action_performed(ActionEvent ae) {

    deleteComps();
    zoomIn();
```

```
    redrawGridCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void contactUsBtn_action_performed(ActionEvent ae) {
    try {
      getAppletContext().showDocument(new URL("mailto:sharb@mail.ucf.edu"));

    }
    catch (IOException ioe) {

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void selectAllCompsBtn_action_performed(ActionEvent ae) {
    selectAllComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void helpBtn_action_performed(ActionEvent ae) {
    showHelpWindow();

  }

////////////////////////////////////////////////////////////////////////////
  private void rotateRightBtn_action_performed(ActionEvent ae) {
    saveOriginalSettings();
    rotateComp(90);
  }

////////////////////////////////////////////////////////////////////////////
  private void rotateLeftBtn_action_performed(ActionEvent ae) {
    saveOriginalSettings();
    rotateComp( -90);
  }

////////////////////////////////////////////////////////////////////////////
  private void flipVerticalBtn_action_performed(ActionEvent ae) {
    flipVertical();

  }
```

```
////////////////////////////////////////////////////////////////////////////
  private void flipHorizontalBtn_action_performed(ActionEvent ae) {
    flipHorizontal();

  }

////////////////////////////////////////////////////////////////////////////
  private void displayPropertiesBtn_action_performed(ActionEvent ae) {

    showDisplayPropertiesFrame();

  }

////////////////////////////////////////////////////////////////////////////
  private void editComponentBtn_action_performed(ActionEvent ae) {
    showUserInput();

  }

////////////////////////////////////////////////////////////////////////////
  private void rotateTextBtn_action_performed(ActionEvent ae) {

  }

////////////////////////////////////////////////////////////////////////////
  private void groupComponentsBtn_action_performed(ActionEvent ae) {
    groupComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void ungroupComponentsBtn_action_performed(ActionEvent ae) {
    ungroupComps();
    deSelectComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void regroupComponentsBtn_action_performed(ActionEvent ae) {
    regroupComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void alignRightBtn_action_performed(ActionEvent ae) {
```

```
    alignRight();

  }

////////////////////////////////////////////////////////////////////////////
  private void alignLeftBtn_action_performed(ActionEvent ae) {

    alignLeft();
  }

////////////////////////////////////////////////////////////////////////////
  private void alignTopBtn_action_performed(ActionEvent ae) {

    alignTop();
  }

////////////////////////////////////////////////////////////////////////////
  private void alignBottomBtn_action_performed(ActionEvent ae) {

    alignBottom();
  }

////////////////////////////////////////////////////////////////////////////

  private void serverFileManagerBtn_action_performed(ActionEvent ae) {

    RemoteFileManagerFrame fm = new RemoteFileManagerFrame();
    fm.show();
  }

////////////////////////////////////////////////////////////////////////////
  private void handBtn_action_performed(ActionEvent ae) {
    ActiveState.reSet();
    ActiveState.handCmd = true;
  }

////////////////////////////////////////////////////////////////////////////
  private void dCursorBtn_action_performed(ActionEvent ae) {
    ActiveState.reSet();
    ActiveState.defaultCursorCmd = true;
  }

////////////////////////////////////////////////////////////////////////////
  private void deletePageBtn_action_performed(ActionEvent ae) {
    deleteCurrentSchematic();
```

```
  }

////////////////////////////////////////////////////////////////////////////
  private void fitZoomBtn_action_performed(ActionEvent ae) {}

////////////////////////////////////////////////////////////////////////////
  private void runBtn_action_performed(ActionEvent ae) {
    simulate();
  }

////////////////////////////////////////////////////////////////////////////
  private void insertTextBtn_action_performed(ActionEvent ae) {
    ActiveState.insertTextCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void fillColorBtn_action_performed(ActionEvent ae) {
    ColorPanel fillColor = new ColorPanel("drawingFillColor", this);
    fillColor.show();
  }

////////////////////////////////////////////////////////////////////////////
  private void lineColorBtn_action_performed(ActionEvent ae) {
    ColorPanel lineColor = new ColorPanel("drawingLineColor", this);
    lineColor.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void dftCursorBtn_action_performed(ActionEvent ae) {
    ActiveState.reSet();
    gridCanvas.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
  }

////////////////////////////////////////////////////////////////////////////
  private void canvasPropertiesBtn_action_performed(ActionEvent ae) {

  }

////////////////////////////////////////////////////////////////////////////
  private void lineBtn_action_performed(ActionEvent ae) {
    deActivateAllComps();
    ActiveState.lineCmd = true;
```

```
  }

////////////////////////////////////////////////////////////////////////////
  private void goToBtn_action_performed(ActionEvent ae) {
    GoToPosition goTo = new GoToPosition(this);
    goTo.show();
  }

////////////////////////////////////////////////////////////////////////////
  private void canvasSizeBtn_action_performed(ActionEvent ae) {
    CanvasPropertiesFrame canvasSize = new CanvasPropertiesFrame(this, 1);
    canvasSize.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void canvasColorsBtn_action_performed(ActionEvent ae) {
    CanvasPropertiesFrame canvasColors = new CanvasPropertiesFrame(this, 0);
    canvasColors.show();
  }

////////////////////////////////////////////////////////////////////////////
  private void lastForwardBtn_action_performed(ActionEvent ae) {
    goToLastPage();
  }

////////////////////////////////////////////////////////////////////////////
  private void forwardBtn_action_performed(ActionEvent ae) {
    goToNextPage();
  }

////////////////////////////////////////////////////////////////////////////
  private void backwardBtn_action_performed(ActionEvent ae) {
    goToPreviousPage();
  }

////////////////////////////////////////////////////////////////////////////
  private void lastBackwardBtn_action_performed(ActionEvent ae) {
    goToFirstPage();
  }

////////////////////////////////////////////////////////////////////////////
  private void refreshBtn_action_performed(ActionEvent ae) {
    refreshCurrentPage();
```

```
  }

////////////////////////////////////////////////////////////////////////////
  private void gridBtn_action_performed(ActionEvent ae) {

    if (ActiveState.showGrid) {
      ActiveState.showGrid = false;
    }
    else {
      ActiveState.showGrid = true;
    }
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void circleBtn_action_performed(ActionEvent ae) {
    deActivateAllComps();
    ActiveState.circleCmd = true;
  }

////////////////////////////////////////////////////////////////////////////
  private void rectangleBtn_action_performed(ActionEvent ae) {
    deActivateAllComps();
    ActiveState.rectangleCmd = true;
  }

////////////////////////////////////////////////////////////////////////////
  private void simulationSettingsBtn_action_performed(ActionEvent ae) {

    showSimulationSettingsFrame();

  }

////////////////////////////////////////////////////////////////////////////
  private void new1Btn_action_performed(ActionEvent ae) {
    activatePage1();
  }

////////////////////////////////////////////////////////////////////////////
  private void new2Btn_action_performed(ActionEvent ae) {
    activatePage2();

  }

////////////////////////////////////////////////////////////////////////////
```

```
  private void new3Btn_action_performed(ActionEvent ae) {
    activatePage3();

  }

////////////////////////////////////////////////////////////////////////////
  private void new4Btn_action_performed(ActionEvent ae) {
    activatePage4();

  }

////////////////////////////////////////////////////////////////////////////
  private void new5Btn_action_performed(ActionEvent ae) {
    activatePage5();

  }

////////////////////////////////////////////////////////////////////////////
  private void textColorBtn_action_performed(ActionEvent ae) {
    ColorPanel fillColor = new ColorPanel("drawingTextColor", this);
    fillColor.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void roundRectangleBtn_action_performed(ActionEvent ae) {
    deActivateAllComps();
    ActiveState.roundRectangleCmd = true;
  }

////////////////////////////////////////////////////////////////////////////
  private void arcBtn_action_performed(ActionEvent ae) {
    ActiveState.arcCmd = true;
  }

////////////////////////////////////////////////////////////////////////////
  private void fileManagerBtn_action_performed(ActionEvent ae) {

    RemoteFileManagerFrame fm = new RemoteFileManagerFrame();
    fm.show();
  }

////////////////////////////////////////////////////////////////////////////
  private void deleteSchematicBtn_action_performed(ActionEvent ae) {
```

```
    deleteCurrentSchematic();
  }

////////////////////////////////////////////////////////////////////////////
  private void simSettingsBtn_action_performed(ActionEvent ae) {

    showSimulationSettingsFrame();

  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementFlipVertical_action_performed(ActionEvent ae) {

    flipVertical();

  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementFlipHorizontal_action_performed(ActionEvent ae) {

    flipHorizontal();

  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementDisplayProperties_action_performed(ActionEvent ae) {

    if (ActiveState.active || ActiveState.multiActive) {
      DisplayPropertiesFrame displayProperties = new DisplayPropertiesFrame(this);
      displayProperties.show();

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementZoomIn_action_performed(ActionEvent ae) {

    ActiveState.reSet();
    ActiveState.zoomInCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementZoomOut_action_performed(ActionEvent ae) {
```

```
    ActiveState.reSet();
    ActiveState.zoomOutCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementGoTo_action_performed(ActionEvent ae) {

    GoToPosition goTo = new GoToPosition(this);
    goTo.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsDisplayProperties_action_performed(ActionEvent ae) {

    if (ActiveState.active || ActiveState.multiActive) {
      DisplayPropertiesFrame displayProperties = new DisplayPropertiesFrame(this);
      displayProperties.show();
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsFlipVertical_action_performed(ActionEvent ae) {

    flipVertical();

  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsFlipHorizontal_action_performed(ActionEvent ae) {

    flipHorizontal();

  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsRegroup_action_performed(ActionEvent ae) {

    regroupComps();

  }
```

```
//////////////////////////////////////////////////////////////////////////
  private void multiElementsAlignTop_action_performed(ActionEvent ae) {

    alignTop();

  }

//////////////////////////////////////////////////////////////////////////
  private void multiElementsAlignLeft_action_performed(ActionEvent ae) {

    alignLeft();

  }

//////////////////////////////////////////////////////////////////////////
  private void multiElementsAlignRight_action_performed(ActionEvent ae) {

    alignRight();

  }

//////////////////////////////////////////////////////////////////////////
  private void multiElementsAlignBottom_action_performed(ActionEvent ae) {

    alignBottom();

  }

//////////////////////////////////////////////////////////////////////////
  private void multiElementsZoomIn_action_performed(ActionEvent ae) {

    ActiveState.reSet();
    ActiveState.zoomInCmd = true;

  }

//////////////////////////////////////////////////////////////////////////
  private void multiElementsZoomOut_action_performed(ActionEvent ae) {

    ActiveState.reSet();
    ActiveState.zoomOutCmd = true;

  }

//////////////////////////////////////////////////////////////////////////
```

```
  private void multiElementsGoTo_action_performed(ActionEvent ae) {

    GoToPosition goTo = new GoToPosition(this);
    goTo.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void goToLine_action_performed(ActionEvent ae) {

    GoToPosition goTo = new GoToPosition(this);
    goTo.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void goToPage_action_performed(ActionEvent ae) {

    //showSimulationSettings();

  }

////////////////////////////////////////////////////////////////////////////
  private void newMenuItem_action_performed(ActionEvent ae) {

    createNewPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void openMenuItem_action_performed(ActionEvent ae) {

    openLocalSchematic();

  }

////////////////////////////////////////////////////////////////////////////
  private void saveMenuItem_action_performed(ActionEvent ae) {

    saveSchematicToLocal();

  }

////////////////////////////////////////////////////////////////////////////
  private void saveAsMenuItem_action_performed(ActionEvent ae) {
```

```
    ActiveState.pageSaved = false;
    ActiveState.pageFile = null;
    saveSchematicToLocal();

  }

////////////////////////////////////////////////////////////////////////////
  private void closePageMenuItem_action_performed(ActionEvent ae) {

    deleteCurrentSchematic();

  }

////////////////////////////////////////////////////////////////////////////
  private void cutMenuItem_action_performed(ActionEvent ae) {

    cutComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void copyMenuItem_action_performed(ActionEvent ae) {

    copyComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void pasteMenuItem_action_performed(ActionEvent ae) {

    pasteComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void deleteMenuItem_action_performed(ActionEvent ae) {

    deleteComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void selectAllMenuItem_action_performed(ActionEvent ae) {
```

```
    selectAllComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void undoMenuItem_action_performed(ActionEvent ae) {

    undoActions();

  }

////////////////////////////////////////////////////////////////////////////
  private void redoMenuItem_action_performed(ActionEvent ae) {

    redoActions();

  }

////////////////////////////////////////////////////////////////////////////
  private void zoomInMenuItem_action_performed(ActionEvent ae) {

    ActiveState.reSet();
    ActiveState.zoomInCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void fitZoomMenuItem_action_performed(ActionEvent ae) {

    ActiveState.reSet();
    ActiveState.zoomInCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void zoomOutMenuItem_action_performed(ActionEvent ae) {

    ActiveState.reSet();
    ActiveState.zoomOutCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void scrollHandMenuItem_action_performed(ActionEvent ae) {
```

```
    //showSimulationSettings();

  }

////////////////////////////////////////////////////////////////////////////
  private void defaultCursorMenuItem_action_performed(ActionEvent ae) {

    //showSimulationSettings();

  }

////////////////////////////////////////////////////////////////////////////
  private void helpMenuItem_action_performed(ActionEvent ae) {

    showHelpWindow();

  }

////////////////////////////////////////////////////////////////////////////
  private void refreshMenuItem_action_performed(ActionEvent ae) {

    refreshCurrentPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void standardBarMenuItem_action_performed(ActionEvent ae) {

    if (ActiveState.showStandardBar) {
      standardToolbar.setVisible(false);
      Point compLoc = componentToolbar.getLocation();
      componentToolbar.setLocation(compLoc.x, compLoc.y - 30);

      Point canvasLoc = canvasToolbar.getLocation();
      canvasToolbar.setLocation(canvasLoc.x, canvasLoc.y - 30);

      Point simulationLoc = simulationToolbar.getLocation();
      simulationToolbar.setLocation(simulationLoc.x, simulationLoc.y - 30);
      switch (ActiveState.toolBarLevel) {
        case 1:
          reLocateGUI(0);
          break;
        case 2:
          reLocateGUI(1);
          break;
```

```
      case 3:
        reLocateGUI(2);
        break;
      }
      ;
      ActiveState.showStandardBar = false;
      standardBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                    ActiveState.MENUNOTCHECKED]));
    }
    else {
      standardToolbar.setVisible(true);
      Point compLoc = componentToolbar.getLocation();
      componentToolbar.setLocation(compLoc.x, compLoc.y + 30);

      Point canvasLoc = canvasToolbar.getLocation();
      canvasToolbar.setLocation(canvasLoc.x, canvasLoc.y + 30);

      Point simulationLoc = simulationToolbar.getLocation();
      simulationToolbar.setLocation(simulationLoc.x, simulationLoc.y + 30);
      switch (ActiveState.toolBarLevel) {
        case 0:
          reLocateGUI(1);
          ActiveState.toolBarLevel = 1;
          break;
        case 1:
          reLocateGUI(2);
          ActiveState.toolBarLevel = 2;
          break;
        case 2:
          reLocateGUI(3);
          ActiveState.toolBarLevel = 3;
          break;
      }

      ActiveState.showStandardBar = true;
      standardBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                    ActiveState.MENUCHECKED]));
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void canvasBarMenuItem_action_performed(ActionEvent ae) {

    if (ActiveState.showCanvasBar) {
```

```
  canvasToolbar.setVisible(false);

  if (!ActiveState.showSimulationBar) {
    Point componentLoc = componentToolbar.getLocation();
    componentToolbar.setLocation(componentLoc.x, componentLoc.y - 30);

    switch (ActiveState.toolBarLevel) {
      case 1:
        reLocateGUI(0);
        ActiveState.toolBarLevel = 0;
        break;
      case 2:
        reLocateGUI(1);
        ActiveState.toolBarLevel = 1;
        break;
      case 3:
        reLocateGUI(2);
        ActiveState.toolBarLevel = 2;
        break;
    }
    ;

  }

  ActiveState.showCanvasBar = false;
  canvasBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUNOTCHECKED]));
}
else {
  canvasToolbar.setVisible(true);

  if (!ActiveState.showSimulationBar) {
    Point componentLoc = componentToolbar.getLocation();
    componentToolbar.setLocation(componentLoc.x, componentLoc.y + 30);

    Point simulationLoc = simulationToolbar.getLocation();
    canvasToolbar.setLocation(simulationLoc);

    switch (ActiveState.toolBarLevel) {
      case 0:
        reLocateGUI(1);
        ActiveState.toolBarLevel = 1;
        break;
      case 1:
        reLocateGUI(2);
```

```
          ActiveState.toolBarLevel = 2;
          break;
        case 2:
          reLocateGUI(3);
          ActiveState.toolBarLevel = 3;
          break;
      }
      ;
    }
    else {
      canvasToolbar.setLocation(418, 35);
    }
    ActiveState.showCanvasBar = true;
    canvasBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                    ActiveState.MENUCHECKED]));
  }

}

////////////////////////////////////////////////////////////////////////////
  private void componentBarMenuItem_action_performed(ActionEvent ae) {

    if (ActiveState.showComponentBar) {
      componentToolbar.setVisible(false);

      switch (ActiveState.toolBarLevel) {
        case 1:
          reLocateGUI(0);
          ActiveState.toolBarLevel = 0;
          break;
        case 2:
          reLocateGUI(1);
          ActiveState.toolBarLevel = 1;
          break;
        case 3:
          reLocateGUI(2);
          ActiveState.toolBarLevel = 2;
          break;
      }
      ;
      ActiveState.showComponentBar = false;
      componentBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                      ActiveState.MENUNOTCHECKED]));
    }
    else {
```

```
    componentToolbar.setVisible(true);

    switch (ActiveState.toolBarLevel) {
      case 0:
        reLocateGUI(1);
        ActiveState.toolBarLevel = 1;
        break;
      case 1:
        reLocateGUI(2);
        ActiveState.toolBarLevel = 2;
        componentToolbar.setLocation(0, 35);
        break;
      case 2:
        reLocateGUI(3);
        ActiveState.toolBarLevel = 3;
        componentToolbar.setLocation(0, 65);
        break;
    }
    ;

    ActiveState.showComponentBar = true;
    componentBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                    ActiveState.MENUCHECKED]));
  }
 }

////////////////////////////////////////////////////////////////////////////
  private void paintingBarMenuItem_action_performed(ActionEvent ae) {

   if (ActiveState.showPaintingBar) {
    paintingToolbar.setVisible(false);
    ActiveState.showPaintingBar = false;
    paintingBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                    ActiveState.MENUNOTCHECKED]));
   }
   else {
    paintingToolbar.setVisible(true);
    ActiveState.showPaintingBar = true;
    paintingBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                    ActiveState.MENUCHECKED]));
   }

  }

////////////////////////////////////////////////////////////////////////////
```

```
  private void visitedElementsBarMenuItem_action_performed(ActionEvent ae) {
    if (ActiveState.showVisitedElementsBar) {
      visitedElementsPanel.setVisible(false);
      ActiveState.showVisitedElementsBar = false;
      visitedElementsBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
          ActiveState.MENUNOTCHECKED]));
    }
    else {
      visitedElementsPanel.setVisible(true);
      ActiveState.showVisitedElementsBar = true;
      visitedElementsBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
          ActiveState.MENUCHECKED]));
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void pageExplorerBarMenuItem_action_performed(ActionEvent ae) {
    if (ActiveState.showPageExplorerBar) {
      pagesExplorerToolbar.setVisible(false);
      ActiveState.showPageExplorerBar = false;
      pageExplorerBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
          ActiveState.MENUNOTCHECKED]));
    }
    else {
      pagesExplorerToolbar.setVisible(true);
      ActiveState.showPageExplorerBar = true;
      pageExplorerBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
          ActiveState.MENUCHECKED]));
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void statusBarMenuItem_action_performed(ActionEvent ae) {

    if (ActiveState.showStatusBar) {
      statusPanel.setVisible(false);
      ActiveState.showStatusBar = false;
      statusBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                  ActiveState.MENUNOTCHECKED]));

    }
    else {
      statusPanel.setVisible(true);
```

```
      ActiveState.showStatusBar = true;
      statusBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                    ActiveState.MENUCHECKED]));
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void acknowledgmentMenuItem_action_performed(ActionEvent ae) {

    Acknowledgments ack = new Acknowledgments();
    ack.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void contactUsMenuItem_action_performed(ActionEvent ae) {

    ContactUS contact = new ContactUS();
    contact.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void exitMenuItem_action_performed(ActionEvent ae) {

    exitApplet();

  }

////////////////////////////////////////////////////////////////////////////
  private void simulationBarMenuItem_action_performed(ActionEvent ae) {

    if (ActiveState.showSimulationBar) {
      simulationToolbar.setVisible(false);

      if (!ActiveState.showCanvasBar) {
        Point componentLoc = componentToolbar.getLocation();
        componentToolbar.setLocation(componentLoc.x, componentLoc.y - 30);

        switch (ActiveState.toolBarLevel) {
          case 1:
            reLocateGUI(0);
            ActiveState.toolBarLevel = 0;
            break;
```

```
      case 2:
        reLocateGUI(1);
        ActiveState.toolBarLevel = 1;
        break;
      case 3:
        reLocateGUI(2);
        ActiveState.toolBarLevel = 2;
        break;
    }
    ;

  }
  else {
    Point simulationLoc = simulationToolbar.getLocation();
    canvasToolbar.setLocation(simulationLoc);

  }
  ActiveState.showSimulationBar = false;
  simulationBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                  ActiveState.MENUNOTCHECKED]));
}
else {
  simulationToolbar.setVisible(true);

  if (!ActiveState.showCanvasBar) {
    Point componentLoc = simulationToolbar.getLocation();
    componentToolbar.setLocation(componentLoc.x, componentLoc.y + 30);

    switch (ActiveState.toolBarLevel) {
      case 0:
        reLocateGUI(1);
        ActiveState.toolBarLevel = 1;
        break;
      case 1:
        reLocateGUI(2);
        ActiveState.toolBarLevel = 2;
        break;
      case 2:
        reLocateGUI(3);
        ActiveState.toolBarLevel = 3;
        break;
    }
  }
  else {
    canvasToolbar.setLocation(418, 35);
```

```
      }
      ActiveState.showSimulationBar = true;
      simulationBarMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
                                        ActiveState.MENUCHECKED]));
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void canvasColorsMenuItem_action_performed(ActionEvent ae) {

    CanvasPropertiesFrame canvasColors = new CanvasPropertiesFrame(this, 0);
    canvasColors.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void canvasSizeMenuItem_action_performed(ActionEvent ae) {
    CanvasPropertiesFrame canvasSize = new CanvasPropertiesFrame(this, 1);
    canvasSize.show(); ;

  }

////////////////////////////////////////////////////////////////////////////
  private void voltageMarkerMenuItem_action_performed(ActionEvent ae) {

    ActiveState.CurComp = ActiveState.createNew(ActiveState.VOLTAGEMARK);

  }

////////////////////////////////////////////////////////////////////////////
  private void diffVoltageMarkerMenuItem_action_performed(ActionEvent ae) {

    ActiveState.CurComp = ActiveState.createNew(ActiveState.NODEL);

  }

////////////////////////////////////////////////////////////////////////////
  private void currentMarkerMenuItem_action_performed(ActionEvent ae) {

    ActiveState.CurComp = ActiveState.createNew(ActiveState.CURRENTCOMP);

  }

////////////////////////////////////////////////////////////////////////////
```

```
  private void hideAllMarkersMenuItem_action_performed(ActionEvent ae) {

    hideAllMarkers();

  }

////////////////////////////////////////////////////////////////////////////
  private void showAllMarkersMenuItem_action_performed(ActionEvent ae) {

    showAllMarkers();

  }

////////////////////////////////////////////////////////////////////////////
  private void lastPreviousMenuItem_action_performed(ActionEvent ae) {

    goToFirstPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void previousMenuItem_action_performed(ActionEvent ae) {

    goToPreviousPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void nextMenuItem_action_performed(ActionEvent ae) {

    goToNextPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void lastNextMenuItem_action_performed(ActionEvent ae) {

    goToLastPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void goToMenuItem_action_performed(ActionEvent ae) {

    GoToPosition goTo = new GoToPosition(this);
```

```
    goTo.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void editPageInfoMenuItem_action_performed(ActionEvent ae) {

    EditPageInfo pageTitle = new EditPageInfo(this);
    pageTitle.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void drawTextMenuItem_action_performed(ActionEvent ae) {

    ActiveState.insertTextCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void drawLineMenuItem_action_performed(ActionEvent ae) {

    deActivateAllComps();
    ActiveState.lineCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void drawCircleMenuItem_action_performed(ActionEvent ae) {

    deActivateAllComps();
    ActiveState.circleCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void drawRectangleMenuItem_action_performed(ActionEvent ae) {

    deActivateAllComps();
    ActiveState.rectangleCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void drawRoundRectangleMenuItem_action_performed(ActionEvent ae) {
```

```
    deActivateAllComps();
    ActiveState.roundRectangleCmd = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void displayPropertiesMenuItem_action_performed(ActionEvent ae) {

    if (ActiveState.active || ActiveState.multiActive) {
      DisplayPropertiesFrame displayProperties = new DisplayPropertiesFrame(this);
      displayProperties.show();

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void editPropertiesMenuItem_action_performed(ActionEvent ae) {

    showUserInput();

  }

////////////////////////////////////////////////////////////////////////////
  private void rotateRightMenuItem_action_performed(ActionEvent ae) {
    saveOriginalSettings();
    rotateComp(90);

  }

////////////////////////////////////////////////////////////////////////////
  private void rotateLeftMenuItem_action_performed(ActionEvent ae) {

    saveOriginalSettings();
    rotateComp( -90);

  }

////////////////////////////////////////////////////////////////////////////
  private void flipVerticalMenuItem_action_performed(ActionEvent ae) {

    flipVertical();

  }
```

```
//////////////////////////////////////////////////////////////////////////
  private void flipHorizontalMenuItem_action_performed(ActionEvent ae) {

    flipHorizontal();

  }

//////////////////////////////////////////////////////////////////////////
  private void rotateTextMenuItem_action_performed(ActionEvent ae) {

    rotateText();

  }

//////////////////////////////////////////////////////////////////////////
  private void alignRightMenuItem_action_performed(ActionEvent ae) {

    alignRight();

  }

//////////////////////////////////////////////////////////////////////////
  private void alignLeftMenuItem_action_performed(ActionEvent ae) {

    alignLeft();

  }

//////////////////////////////////////////////////////////////////////////
  private void alignTopMenuItem_action_performed(ActionEvent ae) {

    alignTop();

  }

//////////////////////////////////////////////////////////////////////////
  private void alignBottomMenuItem_action_performed(ActionEvent ae) {

    alignBottom();

  }

//////////////////////////////////////////////////////////////////////////
  private void groupMenuItem_action_performed(ActionEvent ae) {
```

```
    groupComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void ungroupMenuItem_action_performed(ActionEvent ae) {

    ungroupComps();
    deSelectComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void regroupMenuItem_action_performed(ActionEvent ae) {

    regroupComps();

  }

////////////////////////////////////////////////////////////////////////////
  private void createNetlistMenuItem_action_performed(ActionEvent ae) {

    generateNetList();

  }

////////////////////////////////////////////////////////////////////////////
  private void editNetlistMenuItem_action_performed(ActionEvent ae) {

    showEditNetlistFrame();

  }

////////////////////////////////////////////////////////////////////////////
  private void simulationSettingsMenuItem_action_performed(ActionEvent ae) {

    showSimulationSettingsFrame();

  }

////////////////////////////////////////////////////////////////////////////
  private void runMenuItem_action_performed(ActionEvent ae) {

    simulate();
```

```
  }

////////////////////////////////////////////////////////////////////////////
  private void showNodesNetNamesMenuItem_action_performed(ActionEvent ae) {

    if (!ActiveState.IsNetlistNodesShown) {

      ActiveState.IsNetlistNodesShown = true;
    }
    else {

      ActiveState.IsNetlistNodesShown = false;
    }
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void showNodesVoltagesMenuItem_action_performed(ActionEvent ae) {

    if (!ActiveState.IsNodesVoltagesShown) {

      ActiveState.IsNodesVoltagesShown = true;
    }
    else {

      ActiveState.IsNodesVoltagesShown = false;
    }
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void showNodesCurrentsMenuItem_action_performed(ActionEvent ae) {

    if (!ActiveState.IsNodesCurrentsShown) {

      ActiveState.IsNodesCurrentsShown = true;
    }
    else {

      ActiveState.IsNodesCurrentsShown = false;
    }
    redrawGridCanvas();
  }
```

```
////////////////////////////////////////////////////////////////////////////
  private void viewCIRMLMenuItem_action_performed(ActionEvent ae) {

    showCIRMLFrame();

  }

////////////////////////////////////////////////////////////////////////////
  private void viewSimulationResultMenuItem_action_performed(ActionEvent ae) {

    showSimulationResultFrame();

  }

////////////////////////////////////////////////////////////////////////////
  private void drawResultMenuItem_action_performed(ActionEvent ae) {

    drawGraphicalResults();

  }

////////////////////////////////////////////////////////////////////////////
  private void showGridMenuItem_action_performed(ActionEvent ae) {

    if (ActiveState.showGrid) {
      ActiveState.showGrid = false;
      showGridMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                               MENUNOTCHECKED]));
    }
    else {
      ActiveState.showGrid = true;
      showGridMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                               MENUCHECKED]));
    }
    redrawGridCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void showConnectionPointsMenuItem_action_performed(ActionEvent ae) {

    if (ActiveState.showNodesVisible) {
      setNodesVisible(false);
      redrawGridCanvas();
      showConnectionPointsMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
```

```
        ActiveState.MENUNOTCHECKED]));
      ActiveState.showNodesVisible = false;
    }
    else {
      setNodesVisible(true);
      redrawGridCanvas();
      showConnectionPointsMenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[
        ActiveState.MENUCHECKED]));
      ActiveState.showNodesVisible = true;
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void clearAllMenuItem_action_performed(ActionEvent ae) {

    clearCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void copyPageMenuItem_action_performed(ActionEvent ae) {

    copyPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void pastePageMenuItem_action_performed(ActionEvent ae) {

    pastePage();

  }

////////////////////////////////////////////////////////////////////////////
  private void selectPageMenuItem_action_performed(ActionEvent ae) {

    SelectPageFrame pagesList = new SelectPageFrame(this);
    pagesList.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void findMenuItem_action_performed(ActionEvent ae) {

    find();
```

```
  }

//////////////////////////////////////////////////////////////////////////
  private void replaceMenuItem_action_performed(ActionEvent ae) {

    replace();

  }

//////////////////////////////////////////////////////////////////////////
  private void duplicateMenuItem_action_performed(ActionEvent ae) {

    duplicateComp();

  }

//////////////////////////////////////////////////////////////////////////
  private void page1MenuItem_action_performed(ActionEvent ae) {

    activatePage1();

  }

//////////////////////////////////////////////////////////////////////////
  private void page2MenuItem_action_performed(ActionEvent ae) {

    activatePage2();

  }

//////////////////////////////////////////////////////////////////////////
  private void page3MenuItem_action_performed(ActionEvent ae) {

    activatePage3();

  }

//////////////////////////////////////////////////////////////////////////
  private void page4MenuItem_action_performed(ActionEvent ae) {

    activatePage4();

  }
```

```
////////////////////////////////////////////////////////////////////////////
  private void page5MenuItem_action_performed(ActionEvent ae) {

    activatePage5();

  }

////////////////////////////////////////////////////////////////////////////
  private void redrawMenuItem_action_performed(ActionEvent ae) {

    redrawGridCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void fillColorMenuItem_action_performed(ActionEvent ae) {

    ColorPanel fillColor = new ColorPanel("drawingFillColor", this);
    fillColor.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void lineColorMenuItem_action_performed(ActionEvent ae) {

    ColorPanel lineColor = new ColorPanel("drawingLineColor", this);
    lineColor.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void textColorMenuItem_action_performed(ActionEvent ae) {

    ColorPanel fillColor = new ColorPanel("drawingTextColor", this);
    fillColor.show();

  }

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
  //************CIRCUIT TOOLS EVENT FUNCTIONS*******//
  private void passiveBtn_action_performed(ActionEvent ae) {
//passiveToolbar.setVisible(true);
    passivePanel.setLocation(10, 30);
```

```
  passivePanel.setVisible(true);

}

private void independentBtn_action_performed(ActionEvent ae) {
  independentPanel.setLocation(55, 30);
  independentPanel.setVisible(true);

}

private void dependentBtn_action_performed(ActionEvent ae) {
  dependentPanel.setLocation(10, 205);
  dependentPanel.setVisible(true);

}

private void connectionsBtn_action_performed(ActionEvent ae) {
  connectionsPanel.setLocation(55, 205);
  connectionsPanel.setVisible(true);

}

private void applet_mouseDragged(MouseEvent me) {

}

//************GRIDCANVAS EVENT FUNCTIONS*******//

private void gridCanvas_mouseClicked(MouseEvent me) {

  int maxX = 0, maxY = 0, minX = 1200, minY = 1200;

  int[] xy = new int[2];
  int[] pos = new int[2];

  pos[0] = me.getX();
  pos[1] = me.getY();
  xy = calculateAppPoint(pos);

  clickedPos[0] = pos[0];
  clickedPos[1] = pos[1];

  ActiveState.resetGridMouseAction();
  ActiveState.clicking = true;
  updateStatusBar(me, false);
```

```
gridCanvas.gridImage.setColor(Color.BLUE);

mouseClicked_ToolbarsActions(me);
mouseClicked_drawCompsActions(me);
if (ActiveState.CurComp != null) {

  int type = -1;
  if (ActiveState.CurComp.getComponentType() != ActiveState.CONNECTOR) {
    for (int i = 0; i < componentsCount; i++) {
      if (ActiveState.CurComp.getComponentType() == i) {

        type = i;
        int length = Array.getLength(allComps[i]);
        int index = length - 1;
        allComps[i][length - 1] = ActiveState.CurComp;
        allComps[i][length - 1].setIndex(length - 1);
        allComps[i][length - 1].setNodevisible(ActiveState.showNodesVisible);
        switch (ActiveState.CurComp.getComponentType()) {
          case ActiveState.DEPVCVS:
            allComps[i][length - 1].setPosition(xy);
            allComps[i][length - 1].setControl("vy");
            allComps[i][length -
                1].setComponentName(ActiveState.CurComp.getName() + index);
            break;
          case ActiveState.DEPVCCS:
            allComps[i][length - 1].setPosition(xy);
            allComps[i][length - 1].setControl("vx");
            allComps[i][length -
                1].setComponentName(ActiveState.CurComp.getName() + index);
            break;
          case ActiveState.DEPCCVS:
            allComps[i][length - 1].setPosition(xy);
            allComps[i][length - 1].setControl("iy");
            allComps[i][length -
                1].setComponentName(ActiveState.CurComp.getName() + index);
            break;
          case ActiveState.DEPCCCS:
            allComps[i][length - 1].setPosition(xy);
            allComps[i][length - 1].setControl("ix");
            allComps[i][length -
                1].setComponentName(ActiveState.CurComp.getName() + index);
            break;
          case ActiveState.NODEL:
            allComps[i][length - 1].setPosition(xy);
```

```
  length = Array.getLength(allComps[ActiveState.NODER]);
  allComps[ActiveState.NODER][length -
     1] = ActiveState.createNew(ActiveState.NODER);
  allComps[ActiveState.NODER][length - 1].setIndex(length - 1);
  allComps[ActiveState.NODER][length -
     1].setComponentName(ActiveState.CurComp.getComponentName());
  xy[1] += 50;
  allComps[ActiveState.NODER][length - 1].setPosition(xy);
  allComps[ActiveState.NODER][length - 1].setNodes();
  //xy[1]-=25;
  allComps[ActiveState.NODER] = (Components[]) growArray(allComps[
     ActiveState.NODER]);
  allComps[i][length -
     1].setComponentName(ActiveState.CurComp.getComponentName());
  if (currentComp != null) {
    ( (DepComp) currentComp).controllers[0] = allComps[i][length -
       1];
    allComps[ActiveState.NODER][length -
       1].setControl(currentComp.getControl());
    ( (DepComp) currentComp).controllers[1] = allComps[
       ActiveState.NODER][length - 1];

    currentComp = null;
  }
  break;
case ActiveState.CURRENTCOMP:
  allComps[i][length - 1].setPosition(xy);
  if (ActiveState.CurComp.getComponentName() == null) {
    allComps[i][length -
       1].setComponentName(ActiveState.CurComp.getName() + index);
  }
  else {
    allComps[i][length -
       1].setComponentName(ActiveState.CurComp.getComponentName());
  }
  if (currentComp != null) {
    ( (DepComp) currentComp).controllers[0] = ActiveState.CurComp;
    currentComp = null;
  }
  break;
case ActiveState.VOLTAGEMARK:
  allComps[i][length - 1].setPosition(xy);
  allComps[i][length -
     1].setComponentName(ActiveState.CurComp.getName());
  break;
```

```
        case ActiveState.CURRENTMARK:
          allComps[i][length - 1].setPosition(xy);
          allComps[i][length -
              1].setComponentName(ActiveState.CurComp.getName());

          break;
        case ActiveState.GROUND:

          //xy[0] -= 25;
          allComps[i][length - 1].setPosition(xy);
          allComps[i][length -
              1].setComponentName(ActiveState.CurComp.getName() + index);
          break;
        case ActiveState.CONNODE:
          allComps[i][length - 1].setPosition(xy);
          allComps[i][length -
              1].setComponentName(ActiveState.CurComp.getName() + index);
          splitLineByConenctionNode(allComps[i][length - 1]);

          break;
        default:
          allComps[i][length - 1].setPosition(xy);
          allComps[i][length -
              1].setComponentName(ActiveState.CurComp.getName() + index);
      }
      allComps[i][length - 1].setNodes();
      redrawGridCanvas();
    }
   }
   allComps[type] = (Components[]) growArray(allComps[type]);
   ActiveState.CurComp = null;

  }
  resetSimulationParameters();

  SSSS();
}
else {

  int groupNumb = 0;
  if (!SwingUtilities.isRightMouseButton(me)) {
    ActiveState.active = false;
    ActiveState.activeComp = null;
    ActiveState.multiActive = false;
    ActiveState.grouped = false;
```

```
//activeLabel = null;

xyClicked[0] = xy[0];
xyClicked[1] = xy[1];

for (int type = 0; type < componentsCount; type++) {
  for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
      numbComp++) {

    //if(allComps[type][numbComp].mouseOverArea(pos))
    //{
    if (ActiveState.selectGroup) {
      if (allComps[type][numbComp].mouseOverArea(pos)) {
        allComps[type][numbComp].setActiveState(true);
      }
      if (getNumberOfActiveComponents() > 1) {
        ActiveState.active = false;
        ActiveState.activeComp = null;
        ActiveState.multiActive = true;
      }
      else if (getNumberOfActiveComponents() == 1) {
        ActiveState.active = true;
        ActiveState.activeComp = allComps[type][numbComp];
      }
    }
    else {
      if (allComps[type][numbComp].occupiedArea(pos)) {

        if (allComps[type][numbComp].isGrouped()) {

          groupNumb = allComps[type][numbComp].getGroupNumb();
          ActiveState.grouped = true;
          ActiveState.multiActive = true;

        }
        else {
          // if(getNumberOfActiveComponents()>1)
          // ActiveState.multiActive=true;
          //else if(getNumberOfActiveComponents()==1)
          //{

          ActiveState.active = true;
          ActiveState.nType = type;
          ActiveState.number = numbComp;
          ActiveState.activeComp = allComps[type][numbComp];
```

```
        //}
       }
     } //}
   }
  }
}

if (ActiveState.grouped) {
  for (int type = 0; type < componentsCount; type++) {
    for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
      if (allComps[type][numb].isGrouped() &&
          allComps[type][numb].getGroupNumb() == groupNumb) {
        allComps[type][numb].setActiveState(true);
      }
    }
  }
}

for (int type = 0; type < componentsCount; type++) {
  for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
       numbComp++) {

    if (allComps[type][numbComp].occupiedNameArea(pos) |
        allComps[type][numbComp].occupiedValueArea(pos)) {
      activeComponent = allComps[type][numbComp];
    }

  }
}
for (int type = 0; type < componentsCount; type++) {
  for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
    if ( (allComps[type][numb].getNameState())
        ||
        (allComps[type][numb].getValueState())) {
      ActiveState.txtPressed = true;

    }
  }
}

if (me.getClickCount() >= 2) {
  showUserInput();
  activeComponent = null;

}
```

```
    }
    else {

      xyClicked[0] = pos[0];
      xyClicked[1] = pos[1];
      ActiveState.disableGroup = false;
      ActiveState.disableUngroup = true;

      for (int type = 0; type < componentsCount; type++) {
        for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
          if (allComps[type][numb].getActiveState() &
              allComps[type][numb].isGrouped()) {
            groupNumb = allComps[type][numb].getGroupNumb();
            ActiveState.disableGroup = true;
            ActiveState.disableUngroup = false;
          }
        }
      }
      if (ActiveState.active) {
      }
      showPopupMenu(xy, me);

    }

////////////
    if (ActiveState.grouped) {
      for (int cat = 0; cat < componentsCount; cat++) {
        for (int numb = 0; numb < Array.getLength(allComps[cat]) - 1; numb++) {

          if (allComps[cat][numb].isGrouped() &&
              allComps[cat][numb].getGroupNumb() == groupNumb) {

            for (int numNode = 0; numNode < allComps[cat][numb].getNumNodes();
                 numNode++) {
              int[] POS = allComps[cat][numb].getNode(numNode).getPosition();
              if (POS[0] > maxX) {
                maxX = POS[0];
              }
              if (POS[0] < minX) {
                minX = POS[0];
              }
              if (POS[1] > maxY) {
                maxY = POS[1];
              }
```

```
            if (POS[1] < minY) {
              minY = POS[1];

            }
          }

        }

      }
    }
  }
///////

  }

  ///ActiveState.active=false;
  gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
  gridCanvas.gridImage.setColor(canvasBackgroundColor);
  gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
  gridCanvas.gridImage.setColor(canvasForegroundColor);
  if (ActiveState.showGrid) {
    gridCanvas.drawGridLayout(startPoint);
  }
  drawAllComps();
  drawDrawingComps();
  if (!pageTitle.equals(new String(""))) {
    drawPageTitle();

  }
  if (ActiveState.IsNetlistNodesShown) {
    showNetListNodes();
  }
  if (ActiveState.IsNodesVoltagesShown) {
    showNodesVoltages();
  }
  if (ActiveState.IsNodesCurrentsShown) {
    showNodesCurrents();

  }
  if (ActiveState.grouped) {
    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);

    gridCanvas.gridImage.setColor(new Color(245, 245, 248));
```

```
    gridCanvas.gridImage.fillRect(minX - 25, minY - 25, maxX - minX + 50,
                            maxY - minY + 50);
    gridCanvas.gridImage.setColor(Color.BLUE);
    gridCanvas.gridImage.drawRect(minX - 25, minY - 25, maxX - minX + 50,
                            maxY - minY + 50);

    gridCanvas.gridImage.setColor(canvasForegroundColor);
    if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
    }
    drawAllComps();
    drawDrawingComps();
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();
    }
    if (ActiveState.IsNetlistNodesShown) {
      showNetListNodes();
    }
    if (ActiveState.IsNodesVoltagesShown) {
      showNodesVoltages();
    }
    if (ActiveState.IsNodesCurrentsShown) {
      showNodesCurrents();

    }
    gridCanvas.redraw();

   }

   gridCanvas.redraw();

   countNumberofSelectedItems();
   updateStatusBar(me, false);

  }

////////////////////////////////////////////////////////////////////////////
  private void gridCanvas_mouseReleased(MouseEvent me) {

   int maxX = 0, maxY = 0, minX = 2500, minY = 2500;
   gridCanvas.gridImage.setColor(Color.BLUE);
   xyReleased[0] = me.getX();
   xyReleased[1] = me.getY();
   int[] pos = {
      me.getX(), me.getY()};
```

```
int[] xyPressed = {
   me.getX(), me.getY()};
pos = calculateAppPoint(pos);

if (ActiveState.CurComp != null) {
  if (ActiveState.CurComp.getComponentType() == ActiveState.CONNECTOR) {
    xynew[0] = pos[0];
    xynew[1] = pos[1];
    if (! (xynew[0] == xyold[0] && xynew[1] == xyold[1])) {
      int length = Array.getLength(allComps[ActiveState.CONNECTOR]);
      allComps[ActiveState.CONNECTOR][length - 1] = ActiveState.CurComp;
      int[] points = {
         xyold[0], xyold[1], xynew[0], xynew[1]};
      int[] node1Pos = {
         xyold[0], xyold[1]};
      int[] node2Pos = {
         xynew[0], xynew[1]};
      allComps[ActiveState.CONNECTOR][length - 1].setPoints(points);
      allComps[ActiveState.CONNECTOR][length -
         1].getNode(0).setPosition(node1Pos);
      allComps[ActiveState.CONNECTOR][length -
         1].getNode(1).setPosition(node2Pos);
      allComps[ActiveState.CONNECTOR][length - 1].setNodes();
      allComps[ActiveState.CONNECTOR][length - 1].setIndex(length - 1);
      allComps[ActiveState.CONNECTOR] = (Components[]) growArray(allComps[
         ActiveState.CONNECTOR]);
      ActiveState.CurComp = null;
      splitLines(allComps[ActiveState.CONNECTOR][length - 1]);
    }
    else {
      ActiveState.CurComp = null;
    }
    resetSimulationParameters();
    ActiveState.Resized = false;
  }
}
else {

  if (ActiveState.dragging) {
    int numbOfActiveElements = 0;
    for (int type = 0; type < componentsCount; type++) {
      for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
          numbComp++) {
        if (allComps[type][numbComp].getComponentType() !=
           ActiveState.CONNECTOR) {
```

```
        int[] xypos = allComps[type][numbComp].getPosition();
        if ( (xypos[0] > xyold[0] & xypos[0] < xynew[0]) &
           (xypos[1] > xyold[1] & xypos[1] < xynew[1])) {
          allComps[type][numbComp].setActiveState(true);

          ActiveState.activeComp = allComps[type][numbComp];
          numbOfActiveElements++;
        }
      }
      else {
        int[] p1 = allComps[type][numbComp].getNode(0).getPosition();
        int[] p2 = allComps[type][numbComp].getNode(1).getPosition();
        if ( (p1[0] > xyold[0] & p1[0] < xynew[0]) &
           ( (p1[1] > xyold[1]) & (p1[1] < xynew[1]))
           & ( (p2[0] > xyold[0]) & (p2[0] < xynew[0])) &
           ( (p2[1] > xyold[1]) & (p2[1] < xynew[1]))) {
          allComps[type][numbComp].setActiveState(true);
          ActiveState.activeComp = allComps[type][numbComp];
          numbOfActiveElements++;
        }
      }

    }
  }
  if (numbOfActiveElements == 1) {
    ActiveState.active = true;
  }
  if (numbOfActiveElements > 1) {
    ActiveState.multiActive = true;
    ActiveState.activeComp = null;

  }
}

if (ActiveState.Moved || ActiveState.Resized) {

  for (int type = 0; type < componentsCount; type++) {
    for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
        numbComp++) {

      if (allComps[type][numbComp].getActiveState()) {
        if (type == ActiveState.CONNECTOR) {
          int[] node1 = allComps[ActiveState.CONNECTOR][numbComp].getNode(
             0).getPosition();
          int[] node2 = allComps[ActiveState.CONNECTOR][numbComp].getNode(
```

```
            1).getPosition();
          node1 = calculateAppPoint(node1);
          node2 = calculateAppPoint(node2);
          allComps[ActiveState.CONNECTOR][numbComp].getNode(0).
            setPosition(node1);
          allComps[ActiveState.CONNECTOR][numbComp].getNode(1).
            setPosition(node2);
        }
        else {
          if (moved) {
            int[] p = allComps[type][numbComp].getPosition();
            p = calculateAppPoint(p);

            switch (type) {
              case ActiveState.GROUND:
                allComps[type][numbComp].setPosition(p);
                break;
              default:
                allComps[type][numbComp].setPosition(p);
            }

            allComps[type][numbComp].setNodes();
            moved = false;
          }
        }
      }

    }
  }

////////////////////////////////
//////////////split lines when u resize the line....//////////
    if (ActiveState.activeComp != null) {
      if (ActiveState.active && ActiveState.activeComp.getComponentType()
        == ActiveState.CONNECTOR) {
        int[] n1 = ActiveState.activeComp.getNode(0).getPosition();
        int[] n2 = ActiveState.activeComp.getNode(1).getPosition();
        int[] points = {
          n1[0], n1[1], n2[0], n2[1]};
//splitLines(points,ActiveState.activeComp);
        for (int numb = 0;
          numb < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1;
          numb++) {
          if (allComps[ActiveState.CONNECTOR][numb].equals(ActiveState.
            activeComp)) {
```

```
            splitLines(allComps[ActiveState.CONNECTOR][numb]);

          }

        }
       }
      }
////////////////////////////////////////////////////////
      ActiveState.Resized = false;
      ActiveState.Moved = false;

    }

  }

  if (activeComponent != null && ActiveState.txtPressed) {
    int[] namePos = new int[2];
    int[] valuePos = new int[2];
    int[] POS = allComps[activeComponent.getComponentType()][activeComponent.
       getIndex()].getPosition();

    int[] oldNamePos = allComps[activeComponent.getComponentType()][
       activeComponent.getIndex()].getNamePos();
    int[] oldValuePos = allComps[activeComponent.getComponentType()][
       activeComponent.getIndex()].getValuePos();
    int align = allComps[activeComponent.getComponentType()][activeComponent.
       getIndex()].getAlignment();

    if (align == 1 || align == 3) {
      if (oldNamePos[1] >= POS[1] - 25 && oldNamePos[1] <= POS[1] + 35 &&
          oldNamePos[0] > POS[0] - 25 && oldNamePos[0] < POS[0] + 25) {
        namePos[0] = POS[0] + 15;
        namePos[1] = POS[1] - 5;
        allComps[activeComponent.getComponentType()][activeComponent.getIndex()].
           setNamePos(namePos);
      }

      if (oldValuePos[1] >= POS[1] - 25 && oldValuePos[1] <= POS[1] + 25 &&
          oldValuePos[0] > POS[0] - 25 && oldValuePos[0] < POS[0] + 25) {
        valuePos[0] = POS[0] + 15;
        valuePos[1] = POS[1] + 20;
        allComps[activeComponent.getComponentType()][activeComponent.getIndex()].
           setValuePos(valuePos);
      }
```

```
  }
  else {
    if (oldNamePos[0] >= POS[0] - 50 && oldNamePos[0] <= POS[0] + 25 &&
        oldNamePos[1] > POS[1] - 25 && oldNamePos[1] < POS[1] + 25) {
      namePos[0] = POS[0] - 25;
      namePos[1] = POS[1] - 14;
      allComps[activeComponent.getComponentType()][activeComponent.getIndex()].
          setNamePos(namePos);
    }

    if (oldValuePos[0] >= POS[0] - 25 && oldValuePos[0] <= POS[0] + 25 &&
        oldValuePos[1] > POS[1] - 25 && oldValuePos[1] < POS[1] + 25) {
      valuePos[0] = POS[0] - 10;
      valuePos[1] = POS[1] + 22;
      allComps[activeComponent.getComponentType()][activeComponent.getIndex()].
          setValuePos(valuePos);
    }

  }
  ActiveState.txtPressed = false;
}

if (ActiveState.multiMove) {
  for (int type = 0; type < componentsCount; type++) {
    for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
         numbComp++) {
      if (allComps[type][numbComp].getActiveState() &&
          allComps[type][numbComp].getComponentType() !=
          ActiveState.CONNECTOR) {
        int[] newpos = allComps[type][numbComp].getPosition();
        int[] xypos = calculateAppPoint(newpos);
        allComps[type][numbComp].setPosition(xypos);
        allComps[type][numbComp].setNodes();
      }
      else if (allComps[type][numbComp].getActiveState() &&
               allComps[type][numbComp].getComponentType() ==
               ActiveState.CONNECTOR) {
        int[] node1 = allComps[type][numbComp].getNode(0).getPosition();
        int[] node2 = allComps[type][numbComp].getNode(1).getPosition();
        node1 = calculateAppPoint(node1);
        node2 = calculateAppPoint(node2);
        allComps[type][numbComp].getNode(0).setPosition(node1);
        allComps[type][numbComp].getNode(1).setPosition(node2);
        allComps[type][numbComp].setNodes();
      }
```

```
      }
    }

    ActiveState.multiMove = false;

////////////////////////////////////////////////
    if (ActiveState.grouped) {

      for (int type = 0; type < componentsCount; type++) {
        for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
          if (allComps[type][numb].getActiveState()) {
            for (int numNode = 0; numNode < allComps[type][numb].getNumNodes();
                numNode++) {
              int[] POS = allComps[type][numb].getNode(numNode).getPosition();
              if (POS[0] > maxX) {
                maxX = POS[0];
              }
              if (POS[0] < minX) {
                minX = POS[0];
              }
              if (POS[1] > maxY) {
                maxY = POS[1];
              }
              if (POS[1] < minY) {
                minY = POS[1];

              }
            }

          }
        }
      }
    }

  }

  redrawGridCanvas();

  if (ActiveState.grouped) {

    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(new Color(245, 245, 248));
    gridCanvas.gridImage.fillRect(minX - 25, minY - 25, maxX - minX + 50,
```

```
                        maxY - minY + 50);
  gridCanvas.gridImage.setColor(Color.BLUE);
  gridCanvas.gridImage.drawRect(minX - 25, minY - 25, maxX - minX + 50,
                        maxY - minY + 50);
  gridCanvas.gridImage.setColor(canvasForegroundColor);
  if (ActiveState.showGrid) {
    gridCanvas.drawGridLayout(startPoint);
  }
  drawAllComps();
  drawDrawingComps();
  if (!pageTitle.equals(new String(""))) {
    drawPageTitle();
  }
  if (ActiveState.IsNetlistNodesShown) {
    showNetListNodes();
  }
  if (ActiveState.IsNodesVoltagesShown) {
    showNodesVoltages();
  }
  if (ActiveState.IsNodesCurrentsShown) {
    showNodesCurrents();

  }
  gridCanvas.redraw();

}

gridCanvas.requestFocus();

for (int type = 0; type < componentsCount; type++) {
  for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
       numbComp++) {

  }
}
ActiveState.Moved = false;
ActiveState.multiMove = false;

mouseReleased_ToolbarsActions(me);
mouseReleased_drawCompsActions(me);

ActiveState.resetGridMouseAction();
ActiveState.releasing = true;
countNumberofSelectedItems();
updateStatusBar(me, false);
```

```
    for (int numb = 0;
        numb < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; numb++) {
      int[] node1 = allComps[ActiveState.CONNECTOR][numb].getNode(0).
          getPosition();
      int[] node2 = allComps[ActiveState.CONNECTOR][numb].getNode(1).
          getPosition();
      int[] node3 = allComps[ActiveState.CONNECTOR][numb].getNode(0).
          getEquivPos();
      int[] node4 = allComps[ActiveState.CONNECTOR][numb].getNode(0).
          getEquivPos();

      int[] pos2 = allComps[ActiveState.CONNECTOR][numb].getPosition();

    }
  }

////////////////////////////////////////////////////////////////////////////
  private void gridCanvas_mousePressed(MouseEvent me) {

    xyPressed[0] = me.getX();
    xyPressed[1] = me.getY();

    int[] pos = {
        me.getX(), me.getY()};
    int[] xy = {
        me.getX(), me.getY()};
    pos = calculateAppPoint(pos);

    ActiveState.resetGridMouseAction();
    ActiveState.pressing = true;
    updateStatusBar(me, false);

    xyold[0] = pos[0];
    xyold[1] = pos[1];

    if (ActiveState.CurComp != null) {
      if (ActiveState.CurComp.getComponentType() == ActiveState.CONNECTOR) {
        xyold[0] = pos[0];
        xyold[1] = pos[1];
      }
    }
    else {
      xyold[0] = xy[0];
      xyold[1] = xy[1];
```

```
    for (int type = 0; type < componentsCount; type++) {
      for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
          numbComp++) {

        if (allComps[type][numbComp].mouseOverArea(xy) &&
            allComps[type][numbComp].getActiveState()) {
          ActiveState.Moved = true;

        }
        ///

        if (ActiveState.activeComp != null && ActiveState.active) {
          for (int i = 0; i < allComps[type][numbComp].getNumNodes(); i++) {

            if (ActiveState.activeComp.equals(allComps[type][numbComp]) &&
                allComps[type][numbComp].getNode(i).mouseOverArea(xy)) {
              ActiveState.Moved = false;

              if (ActiveState.activeComp.getComponentType() !=
                  ActiveState.CONNECTOR) {
                ActiveState.CurComp = ActiveState.createNew(ActiveState.
                    CONNECTOR);
                xyold[0] = pos[0];
                xyold[1] = pos[1];
              }
              ActiveState.Resized = true;

            }
          }

        }

        ///
      }
    }

    //ActiveState.txtPressed=false;

////////////////////////////////
    if (ActiveState.multiActive) {

      for (int type = 0; type < componentsCount; type++) {
        for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
            numbComp++) {
```

```
          if (allComps[type][numbComp].mouseOverArea(xy)) {
            ActiveState.multiMove = true;
            ActiveState.movedComp = allComps[type][numbComp];
          }
        }
      }
      oldPos[0] = xy[0];
      oldPos[1] = xy[1];
    }

  }

}

////////////////////////////////////////////////////////////////////////////
  private void gridCanvas_mouseDragged(MouseEvent me) {

    boolean moving = false;
    int maxX = 0, maxY = 0, minX = 1200, minY = 1200;
    gridCanvas.gridImage.setColor(Color.BLUE);

    int[] pos = {
       me.getX(), me.getY()};
    int[] xy = {
       me.getX(), me.getY()};
    pos = calculateAppPoint(pos);

    xynew[0] = xy[0];
    xynew[1] = xy[1];

    ActiveState.resetGridMouseAction();
    ActiveState.dragging = true;
    updateStatusBar(me, false);

    if (ActiveState.CurComp != null) {
      if (ActiveState.CurComp.getComponentType() == ActiveState.CONNECTOR) {
        gridCanvas.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
        gridCanvas.gridImage.setColor(canvasBackgroundColor);
        gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
        gridCanvas.gridImage.setColor(canvasForegroundColor);
        if (ActiveState.showGrid) {
          gridCanvas.drawGridLayout(startPoint);
        }
        drawAllComps();
```

```
  drawDrawingComps();
  if (!pageTitle.equals(new String(""))) {
    drawPageTitle();
  }
  if (ActiveState.IsNetlistNodesShown) {
    showNetListNodes();
  }
  if (ActiveState.IsNodesVoltagesShown) {
    showNodesVoltages();
  }
  if (ActiveState.IsNodesCurrentsShown) {
    showNodesCurrents();

  }
  gridCanvas.gridImage.setColor(new Color(24, 130, 17));
  gridCanvas.gridImage.drawLine(xyold[0], xyold[1], xynew[0], xynew[1]);
  Image img = ImgComponents.imageNormal[ActiveState.CONNODE];
  gridCanvas.gridImage.drawImage(img, xyold[0], xyold[1], this);
  gridCanvas.gridImage.drawImage(img, xynew[0], xynew[1], this);
  gridCanvas.redraw();
 }
 else {
  if (ActiveState.CurComp.getComponentType()
      != ActiveState.CONNECTOR) {

    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasForegroundColor);
    if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
    }
    Image img = ImgComponents.imageNormal[ActiveState.CurComp.
        getComponentType()];
    Image node = ImgComponents.imageNormal[ActiveState.CONNODE];
    //gridCanvas.gridImage.drawString("Current point:(" + pos[0] + "," +
    //                      pos[1] + ")", 250, 10);

    switch (ActiveState.CurComp.getComponentType()) {
      case ActiveState.CONNODE:
        gridCanvas.gridImage.drawImage(img, pos[0], pos[1], this);
        break;
      case ActiveState.CURRENTCOMP:
        gridCanvas.gridImage.drawImage(img, pos[0] - 25, pos[1] - 25, this);
        gridCanvas.gridImage.drawImage(node, pos[0] - 25, pos[1], this);
```

```
  gridCanvas.gridImage.drawImage(node, pos[0] + 25, pos[1], this);
  if (ActiveState.CurComp.getComponentName() != null) {
    gridCanvas.gridImage.drawString(ActiveState.CurComp.
                          getComponentName(), pos[0] - 15,
                          pos[1] - 15);
  }
  else {
    gridCanvas.gridImage.drawString(ActiveState.CurComp.getName(),
                          pos[0] - 15, pos[1] - 15);
  }
  break;
case ActiveState.NODEL:
  gridCanvas.gridImage.drawImage(img, pos[0], pos[1] - 25, this);
  gridCanvas.gridImage.drawImage(img, pos[0], pos[1] + 25, this);
  gridCanvas.gridImage.drawString(ActiveState.CurComp.
                          getComponentName(), pos[0],
                          pos[1] - 27);
  gridCanvas.gridImage.drawString(ActiveState.CurComp.
                          getComponentName(), pos[0],
                          pos[1] + 27);
  break;
case ActiveState.GROUND:
  gridCanvas.gridImage.drawImage(img, xy[0] - 25, xy[1] - 25, this);
  break;
case ActiveState.TRANSFORMERA:
case ActiveState.TRANSFORMERB:
case ActiveState.TRANSFORMERC:
case ActiveState.TRANSFORMERD:
  gridCanvas.gridImage.drawImage(img, pos[0] - 25, pos[1] - 25, this);
  gridCanvas.gridImage.drawImage(node, pos[0] - 25, pos[1] - 25, this);
  gridCanvas.gridImage.drawImage(node, pos[0] + 25, pos[1] - 25, this);
  gridCanvas.gridImage.drawImage(node, pos[0] - 25, pos[1] + 25, this);
  gridCanvas.gridImage.drawImage(node, pos[0] + 25, pos[1] + 25, this);
  gridCanvas.gridImage.drawString(ActiveState.CurComp.getName(),
                          pos[0], pos[1] - 25);
  gridCanvas.gridImage.drawString(ActiveState.CurComp.getValue() +
                          " :1", pos[0] - 10, pos[1] + 35);
  break;
default:
  gridCanvas.gridImage.drawImage(img, pos[0] - 25, pos[1] - 25, this);
  gridCanvas.gridImage.drawImage(node, pos[0] - 25, pos[1], this);
  gridCanvas.gridImage.drawImage(node, pos[0] + 25, pos[1], this);
  gridCanvas.gridImage.drawString(ActiveState.CurComp.getName()
                          , pos[0] - 15, pos[1] - 15);
  gridCanvas.gridImage.drawString(ActiveState.CurComp.getValue()
```

```
                                + " " +
                                ActiveState.CurComp.getUnit(),
                                pos[0] - 15, pos[1] + 20);
          }
        }
      }

    }

    if (ActiveState.active && ActiveState.activeComp != null &&
        ActiveState.CurComp == null) {

      int type = ActiveState.nType;
      int numb = ActiveState.number;
      if (ActiveState.activeComp.getComponentType() != ActiveState.CONNECTOR) {
        if (ActiveState.Moved) {

          moving = true;
          gridCanvas.setCursor(new Cursor(Cursor.MOVE_CURSOR));
          switch (ActiveState.activeComp.getComponentType()) {
            default:
              allComps[type][numb].setPosition(xy);
          }
          allComps[type][numb].setNodes();
          moved = true;
        }

      }
      else {
        xynew[0] = xy[0];
        xynew[1] = xy[1];

//// Moving the line
        if (ActiveState.Moved) {
          moving = true;

          gridCanvas.setCursor(new Cursor(Cursor.MOVE_CURSOR));
          int[] node1 = ActiveState.activeComp.getNode(0).getPosition();
          int[] node2 = ActiveState.activeComp.getNode(1).getPosition();

          int xDif = xynew[0] - xyold[0];
          int yDif = xynew[1] - xyold[1];
          node1[0] += xDif;
          node2[0] += xDif;
          node1[1] += yDif;
```

```
        node2[1] += yDif;
        ActiveState.activeComp.getNode(0).setPosition(node1);
        ActiveState.activeComp.getNode(1).setPosition(node2);
        xyold[0] = xy[0];
        xyold[1] = xy[1];
       }
//else
//{
//ActiveState.activeComp.setActiveState(false);
//}
///

      if (ActiveState.Resized && ActiveState.CurComp == null) {

        for (int numbComps = 0;
            numbComps < Array.getLength(allComps[ActiveState.CONNECTOR]);
            numbComps++) {
          if (allComps[ActiveState.CONNECTOR][numbComps] != null) {
            int numbNodes = allComps[ActiveState.CONNECTOR][numbComps].
               getNumNodes();
            for (int n = 0; n < numbNodes; n++) {
              if (allComps[ActiveState.CONNECTOR][numbComps].getNode(n).
                 getActiveState()) {

                if (allComps[ActiveState.CONNECTOR][numbComps].getActiveState()) {
                  allComps[ActiveState.CONNECTOR][numbComps].getNode(n).
                     setPosition(xy);
                }
               }
             }
           }
         }

      }
     }
     redrawGridCanvas();
   }
   if (ActiveState.multiActive && ActiveState.CurComp == null) {
     if (ActiveState.multiMove) {

      moving = true;
      int[] movedPos = {
         oldPos[0], oldPos[1]};
      int xdiff = xy[0] - movedPos[0];
      int ydiff = xy[1] - movedPos[1];
```

```
      for (int type = 0; type < componentsCount; type++) {
        for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
            numbComp++) {
          if (allComps[type][numbComp].getActiveState()) {
            if (allComps[type][numbComp].getComponentType() ==
               ActiveState.CONNECTOR) {
              int[] node1 = allComps[type][numbComp].getNode(0).getPosition();
              int[] node2 = allComps[type][numbComp].getNode(1).getPosition();

              node1[0] += xdiff;
              node1[1] += ydiff;
              node2[0] += xdiff;
              node2[1] += ydiff;

              allComps[type][numbComp].getNode(0).setPosition(node1);
              allComps[type][numbComp].getNode(1).setPosition(node2);
              allComps[type][numbComp].setNodes();

            }
            else if (allComps[type][numbComp].getComponentType() ==
                  ActiveState.GROUND) {
              int[] compPos = allComps[type][numbComp].getPosition();
              compPos[0] += xdiff;
              compPos[1] += ydiff;
              allComps[type][numbComp].setPosition(compPos);
              allComps[type][numbComp].setNodes();

            }
            else {
              int[] compPos = allComps[type][numbComp].getPosition();
              compPos[0] += xdiff;
              compPos[1] += ydiff;
              allComps[type][numbComp].setPosition(compPos);
              allComps[type][numbComp].setNodes();
            }
          }

        }
      }
//////////////////////////
      if (ActiveState.grouped) {

        for (int type = 0; type < componentsCount; type++) {
          for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
```

```
        if (allComps[type][numb].getActiveState()) {

          for (int numNode = 0;
              numNode < allComps[type][numb].getNumNodes(); numNode++) {
            int[] POS = allComps[type][numb].getNode(numNode).getPosition();
            if (POS[0] > maxX) {
              maxX = POS[0];
            }
            if (POS[0] < minX) {
              minX = POS[0];
            }
            if (POS[1] > maxY) {
              maxY = POS[1];
            }
            if (POS[1] < minY) {
              minY = POS[1];

            }
          }

        }
      }
    }

}

//////////////////////////

    oldPos[0] = xy[0];
    oldPos[1] = xy[1];

    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);

    gridCanvas.gridImage.setColor(new Color(245, 245, 248));
    gridCanvas.gridImage.fillRect(minX - 25, minY - 25, maxX - minX + 50,
                        maxY - minY + 50);
    gridCanvas.gridImage.setColor(Color.BLUE);
    gridCanvas.gridImage.drawRect(minX - 25, minY - 25, maxX - minX + 50,
                        maxY - minY + 50);
    gridCanvas.gridImage.setColor(canvasForegroundColor);
    if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
    }
```

```
    drawAllComps();
    drawDrawingComps();
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();
    }
    if (ActiveState.IsNetlistNodesShown) {
      showNetListNodes();
    }
    if (ActiveState.IsNodesVoltagesShown) {
      showNodesVoltages();
    }
    if (ActiveState.IsNodesCurrentsShown) {
      showNodesCurrents();

    }
    gridCanvas.redraw();

  }
}

if (ActiveState.CurComp == null && !ActiveState.Moved &&
    !ActiveState.Resized
    && !ActiveState.txtPressed && !ActiveState.rectangleCmd &&
    !ActiveState.circleCmd
    && !ActiveState.lineCmd && !ActiveState.textColorCmd &&
    !ActiveState.roundRectangleCmd
    && !ActiveState.arcCmd) {
  xynew[0] = me.getX();
  xynew[1] = me.getY();

  gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
  gridCanvas.gridImage.setColor(canvasBackgroundColor);
  gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
  gridCanvas.gridImage.setColor(canvasForegroundColor);
  if (ActiveState.showGrid) {
    gridCanvas.drawGridLayout(startPoint);
  }
  drawAllComps();
  drawDrawingComps();
  if (!pageTitle.equals(new String(""))) {
    drawPageTitle();

  }
  if (ActiveState.IsNetlistNodesShown) {
    showNetListNodes();
```

```
}
if (ActiveState.IsNodesVoltagesShown) {
  showNodesVoltages();
}
if (ActiveState.IsNodesCurrentsShown) {
  showNodesCurrents();

}
if (!ActiveState.multiMove) {
  gridCanvas.gridImage.setColor(new Color(245, 245, 248));

  gridCanvas.gridImage.fillRect(xyold[0], xyold[1], xynew[0] - xyold[0],
                          xynew[1] - xyold[1]);

  gridCanvas.gridImage.setColor(Color.BLUE);
  gridCanvas.gridImage.drawRect(xyold[0], xyold[1], xynew[0] - xyold[0],
                          xynew[1] - xyold[1]);

  gridCanvas.gridImage.setColor(canvasForegroundColor);
  if (ActiveState.showGrid) {
    gridCanvas.drawGridLayout(startPoint);
  }
  drawAllComps();
  drawDrawingComps();
  if (!pageTitle.equals(new String(""))) {
    drawPageTitle();

  }
  if (ActiveState.IsNetlistNodesShown) {
    showNetListNodes();
  }
  if (ActiveState.IsNodesVoltagesShown) {
    showNodesVoltages();
  }
  if (ActiveState.IsNodesCurrentsShown) {
    showNodesCurrents();

  }
}

for (int type = 0; type < componentsCount; type++) {
  for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
       numbComp++) {
    allComps[type][numbComp].setActiveState(false);
    allComps[type][numbComp].setNameState(false);
```

```
        allComps[type][numbComp].setValueState(false);
      }
    }

    gridCanvas.redraw();
  }

  if (activeComponent != null && ActiveState.txtPressed &&
      !ActiveState.active && !ActiveState.multiActive) {

    redrawGridCanvas();

    gridCanvas.setCursor(new Cursor(Cursor.MOVE_CURSOR));
    if (activeComponent.getNameState()) {
      activeComponent.setNamePos(xy);
      gridCanvas.gridImage.setColor(new Color(235, 250, 255));
      //gridCanvas.gridImage.fillRect(xy[0] - 10, xy[1], 10, 15);
      //gridCanvas.gridImage.drawRect(xy[0], xy[1], 25, 15);
      gridCanvas.redraw();
    }
    else if (activeComponent.getValueState()) {
      switch (activeComponent.getComponentType()) {
        case ActiveState.TRANSFORMERA:
        case ActiveState.TRANSFORMERB:
        case ActiveState.TRANSFORMERC:
        case ActiveState.TRANSFORMERD:
          xy[0] -= 25;
          activeComponent.setValuePos(xy);
          xy[0] += 25;
          break;
        default:
          activeComponent.setValuePos(xy);
      }

      gridCanvas.gridImage.setColor(new Color(235, 250, 255));
      //gridCanvas.gridImage.fillRect(xy[0] - 10, xy[1], 10, 15);
      //gridCanvas.gridImage.drawRect(xy[0], xy[1], 45, 15);
      gridCanvas.redraw();
    }

  }

  mouseDragged_ToolbarsActions(me);
  mouseDragged_drawCompsActions(me);
}
```

```
////////////////////////////////////////////////////////////////////////////
  private void gridCanvas_mouseMoved(MouseEvent me) {

    gridCanvas.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    if (ActiveState.zoomInCmd) {

      gridCanvas.setCursor(zoomInCursor);
    }
    else if (ActiveState.zoomOutCmd) {
      gridCanvas.setCursor(zoomOutCursor);
    }
    else if (ActiveState.handCmd) {
      gridCanvas.setCursor(new Cursor(Cursor.HAND_CURSOR));
    }
    else if (ActiveState.defaultCursorCmd) {
      gridCanvas.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }
    else if (ActiveState.insertTextCmd) {
      gridCanvas.setCursor(new Cursor(Cursor.TEXT_CURSOR));
    }
    else if (ActiveState.lineCmd || ActiveState.circleCmd ||
            ActiveState.rectangleCmd || ActiveState.roundRectangleCmd
            || ActiveState.arcCmd) {
      gridCanvas.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
    }
    else if (ActiveState.defaultCursorCmd) {
      gridCanvas.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }
    else if (ActiveState.simulationCmd) {
      gridCanvas.setCursor(new Cursor(Cursor.WAIT_CURSOR));
    }
    int[] pos = {
       me.getX(), me.getY()};

    ActiveState.resetGridMouseAction();
    ActiveState.moving = true;
    updateStatusBar(me, false);

    if (ActiveState.CurComp != null) {

      gridCanvas.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
      if (ActiveState.CurComp.getComponentType() != ActiveState.CONNECTOR) {

        gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
```

```
gridCanvas.gridImage.setColor(canvasBackgroundColor);
gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
gridCanvas.gridImage.setColor(canvasForegroundColor);
if (ActiveState.showGrid) {
  gridCanvas.drawGridLayout(startPoint);
}
Image img = ActiveState.ImageNormal[ActiveState.CurComp.
    getComponentType()];
Image node = ActiveState.ImageNormal[ActiveState.CONNODE];
// gridCanvas.gridImage.drawString("Current point:(" + pos[0] + "," +
//                          pos[1] + ")", 250, 10);

switch (ActiveState.CurComp.getComponentType()) {
  case ActiveState.CONNODE:
    gridCanvas.gridImage.drawImage(img, pos[0], pos[1], this);
    break;
  case ActiveState.CURRENTCOMP:
    gridCanvas.gridImage.drawImage(img, pos[0] - 25, pos[1] - 25, this);
    gridCanvas.gridImage.drawImage(node, pos[0] - 25, pos[1], this);
    gridCanvas.gridImage.drawImage(node, pos[0] + 25, pos[1], this);
    if (ActiveState.CurComp.getComponentName() != null) {
      gridCanvas.gridImage.drawString(ActiveState.CurComp.
                              getComponentName(), pos[0] - 15,
                              pos[1] - 15);
    }
    else {
      gridCanvas.gridImage.drawString(ActiveState.CurComp.getName(),
                              pos[0] - 15, pos[1] - 15);
    }
    break;
  case ActiveState.NODEL:
    gridCanvas.gridImage.drawImage(img, pos[0], pos[1] - 25, this);
    gridCanvas.gridImage.drawImage(img, pos[0], pos[1] + 25, this);
    gridCanvas.gridImage.drawString(ActiveState.CurComp.
                              getComponentName() + "+", pos[0],
                              pos[1] - 27);
    gridCanvas.gridImage.drawString(ActiveState.CurComp.
                              getComponentName() + "-", pos[0],
                              pos[1] + 27);
    break;
  case ActiveState.VOLTAGEMARK:
    gridCanvas.gridImage.drawImage(img, pos[0], pos[1] - 25, this);
    gridCanvas.gridImage.drawString(ActiveState.CurComp.
                              getComponentName(), pos[0],
                              pos[1] - 27);
```

```
    break;
  case ActiveState.CURRENTMARK:
    gridCanvas.gridImage.drawImage(img, pos[0], pos[1] - 25, this);
    gridCanvas.gridImage.drawString(ActiveState.CurComp.
                            getComponentName(), pos[0],
                            pos[1] - 27);
    break;

  case ActiveState.GROUND:
    gridCanvas.gridImage.drawImage(img, pos[0] - 25, pos[1] - 25, this);
    break;
  case ActiveState.TRANSFORMERA:
  case ActiveState.TRANSFORMERB:
  case ActiveState.TRANSFORMERC:
  case ActiveState.TRANSFORMERD:
    gridCanvas.gridImage.drawImage(img, pos[0] - 25, pos[1] - 25, this);
    gridCanvas.gridImage.drawImage(node, pos[0] - 25, pos[1] - 25, this);
    gridCanvas.gridImage.drawImage(node, pos[0] + 25, pos[1] - 25, this);
    gridCanvas.gridImage.drawImage(node, pos[0] - 25, pos[1] + 25, this);
    gridCanvas.gridImage.drawImage(node, pos[0] + 25, pos[1] + 25, this);
    gridCanvas.gridImage.drawString(ActiveState.CurComp.getName(),
                            pos[0], pos[1] - 25);
    gridCanvas.gridImage.drawString(ActiveState.CurComp.getValue() +
                            " :1", pos[0] - 10, pos[1] + 35);
    break;
  default:
    gridCanvas.gridImage.drawImage(img, pos[0] - 25, pos[1] - 25, this);
    gridCanvas.gridImage.drawImage(node, pos[0] - 25, pos[1], this);
    gridCanvas.gridImage.drawImage(node, pos[0] + 25, pos[1], this);
    gridCanvas.gridImage.drawString(ActiveState.CurComp.getName()
                            , pos[0] - 15, pos[1] - 15);
    gridCanvas.gridImage.drawString(ActiveState.CurComp.getValue()
                            + " " + ActiveState.CurComp.getUnit(),
                            pos[0] - 15, pos[1] + 20);
}
drawAllComps();
drawDrawingComps();
if (!pageTitle.equals(new String(""))) {
  drawPageTitle();
}
gridCanvas.redraw();
if (ActiveState.IsNetlistNodesShown) {
  showNetListNodes();
}
if (ActiveState.IsNodesVoltagesShown) {
```

```
        showNodesVoltages();
       }
       if (ActiveState.IsNodesCurrentsShown) {
        showNodesCurrents();

       }
      }
    }

    for (int type = 0; type < componentsCount; type++) {
     for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
          numbComp++) {

       if (allComps[type][numbComp].mouseOverArea(pos)) {
        gridCanvas.setCursor(new Cursor(Cursor.MOVE_CURSOR));
       }
       else if (allComps[type][numbComp].mouseOverNameArea(pos) ||
              allComps[type][numbComp].mouseOverValueArea(pos)) {
        gridCanvas.setCursor(new Cursor(Cursor.TEXT_CURSOR));
       }
      }
    }

    mouseMoved_drawingCompsActions(me);

  }

////////////////////////////////////////////////////////////////////////////
  private void gridCanvas_mouseExited(MouseEvent me) {
    updateStatusBar(me, true);

  }

////////////////////////////////////////////////////////////////////////////
  private void gridCanvas_keyPressed(KeyEvent me) {

    switch (me.getKeyCode()) {
      case 16:
        ActiveState.selectGroup = true;
        break;
      case 17:
        ActiveState.functionalKey = true;
        break;
      case 114:
        showDisplayPropertiesFrame();
```

```
    break;
  case 113:
    showUserInput();
    break;
  case 112:
    showHelpWindow();
    break;
  case 127:
    deleteComps();
    break;
  case 117:
    showEditNetlistFrame();
    break;
  case 116:
    showCreateNetlistFrame();
    break;
  case 118:
    showSimulationSettingsFrame();
    break;
  case 120:
    simulate();
    break;
  case 119:
    showCIRMLFrame();
    break;
  case 121:
    showSimulationResultFrame();
    break;
  case 27:
    resetActiveComp();
    break;

}
;

if (me.isShiftDown()) {
  switch (me.getKeyCode()) {
    case 71:
      groupComps();
      break;
    case 85:
      ungroupComps();
      break;
    case 82:
      regroupComps();
```

```
      break;
    case 73:
      zoomInCanvas();
      break;
    case 70:
      fitZoomCanvas();
      break;
    case 79:
      zoomOutCanvas();
      break;

  }
  ;

  if (me.isShiftDown() && me.isControlDown()) {
    switch (me.getKeyCode()) {

      case 84:
        alignTop();
        break;
      case 76:
        alignLeft();
        break;
      case 82:
        alignRight();
        break;
      case 66:
        alignBottom();
        break;

    }
    ;

  }

}

if (me.isControlDown()) {
  switch (me.getKeyCode()) {

    case 78:
      createNewPage();
      break;
    case 79:
      openLocalSchematic();
```

```
      break;
    case 83:
      saveSchematicToLocal();
      break;
    case 90:
      undoActions();
      break;
    case 87:
      redoActions();
      break;
    case 88:
      cutComps();
      break;
    case 67:
      copyComps();
      break;
    case 86:
      pasteComps();
      break;
    case 65:
      selectAllComps();
      break;
    case 68:
      duplicateComp();
      break;
    case 82:
      saveOriginalSettings();
      rotateComp(90); ;
      break;
    case 76:
      saveOriginalSettings();
      rotateComp( -90); ;
      break;
    case 72:
      flipHorizontal();
      break;
    case 70:
      flipVertical();
      break;
  }
  ;

}
if (me.isAltDown()) {
  if (me.getKeyCode() == 115) {
```

```
        deleteCurrentSchematic();
      }
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void gridCanvas_keyReleased(KeyEvent me) {

    switch (me.getKeyCode()) {
      case 16:
        ActiveState.selectGroup = false;
        break;
      case 17:
        ActiveState.functionalKey = false;
        break;
    }
    ;

  }

////////////////////////////////////////////////////////////////////////////

  //************   EVENT FUNCTIONS*******//

  private void enterValue_action_performed(ActionEvent ae) {
    if (!enterValue.isEnabled()) {
      return;
    }
    showUserInput();
  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementRotateRight_action_performed(ActionEvent ae) {

    if (!singleElementRotateRight.isEnabled()) {
      return;
    }

    saveOriginalSettings();
    rotateComp(90);
  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementRotateLeft_action_performed(ActionEvent ae) {
```

```
    if (!singleElementRotateLeft.isEnabled()) {
      return;
    }
    saveOriginalSettings();
    rotateComp( -90);
  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementDelete_action_performed(ActionEvent ae) {
    if (!singleElementDelete.isEnabled()) {
      return;
    }
    saveOriginalSettings();
    if (ActiveState.active) {
      deleteComponent(true);

    }
  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementCut_action_performed(ActionEvent ae) {
    if (!singleElementCut.isEnabled()) {
      return;
    }
    cutComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void singleElementCopy_action_performed(ActionEvent ae) {
    if (!singleElementCopy.isEnabled()) {
      return;
    }
    copyComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsRotateLeft_action_performed(ActionEvent ae) {

    if (!multiElementsRotateLeft.isEnabled()) {
      return;
    }
    saveOriginalSettings();
    if (!rotateGroup) {
      rotateComp( -90);
    }
```

```
    else {
      rotateGroup(1);
      rotateGroup = false;
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsRotateRight_action_performed(ActionEvent ae) {
    if (!multiElementsRotateRight.isEnabled()) {
      return;
    }
    saveOriginalSettings();
    if (!rotateGroup) {
      rotateComp(90);
    }
    else {
      rotateGroup(0);
      rotateGroup = false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsDelete_action_performed(ActionEvent ae) {
    if (!multiElementsDelete.isEnabled()) {
      return;
    }
    saveOriginalSettings();
    deleteComponent(true);
    ActiveState.multiActive = false;
  }

////////////////////////////////////////////////////////////////////////////
  private void deleteLine_action_performed(ActionEvent ae) {
    if (!deleteLine.isEnabled()) {
      return;
    }
    saveOriginalSettings();
    if (ActiveState.active) {
      deleteComponent(true);

    }
  }

////////////////////////////////////////////////////////////////////////////
```

```
  private void copyLine_action_performed(ActionEvent ae) {
    if (!copyLine.isEnabled()) {
      return;
    }
    if (ActiveState.active) {
      ActiveState.copiedComp = ActiveState.activeComp;

      ActiveState.singleCopied = true;

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void cutLine_action_performed(ActionEvent ae) {

    if (!cutLine.isEnabled()) {
      return;
    }
    if (ActiveState.active) {
      saveOriginalSettings();
      ActiveState.copiedComp = ActiveState.activeComp;
      ActiveState.singleCopied = true;
      deleteComponent(true);

    }
  }

////////////////////////////////////////////////////////////////////////////
  private void undo_action_performed(ActionEvent ae) {
    if (!undo.isEnabled()) {
      return;
    }
    restoreOriginalSettings();
  }

////////////////////////////////////////////////////////////////////////////
  private void menuItem10_action_performed(ActionEvent ae) {
    if (!menuItem10.isEnabled()) {
      return;
    }
    switch (ActiveState.activePanel) {
      case 0:
        passivePanel.setVisible(false);
        break;
```

```
      case 1:
        independentPanel.setVisible(false);
        break;
      case 2:
        dependentPanel.setVisible(false);
        break;
      case 3:
        connectionsPanel.setVisible(false);
        break;
      case 4:
        transformersPanel.setVisible(false);
        break;
    }
    ActiveState.activePanel = -1;

  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsCopy_action_performed(ActionEvent ae) {
    if (!multiElementsCopy.isEnabled()) {
      return;
    }
    copyComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsCut_action_performed(ActionEvent ae) {
    if (!multiElementsCut.isEnabled()) {
      return;
    }
    cutComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsGroup_action_performed(ActionEvent ae) {
    groupComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void multiElementsUngroup_action_performed(ActionEvent ae) {
    ungroupComps();
    deSelectComps();
  }

////////////////////////////////////////////////////////////////////////////
```

```
  private void paste_action_performed(ActionEvent ae) {
    pasteComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void clearAll_action_performed(ActionEvent ae) {
    clearAllComps();
    ActiveState.DELETED = true;
  }

////////////////////////////////////////////////////////////////////////////
  private void zoomIn_action_performed(ActionEvent ae) {

    /*Graphics2D G = (Graphics2D) gridCanvas.gridImage;
        ActiveState.currentXScale += 0.1;
        ActiveState.currentYScale += 0.1;
        G.scale(ActiveState.currentXScale, ActiveState.currentYScale);
        G.translate(ActiveState.currentXScale, ActiveState.currentYScale);
        //AffineTransform atx = new AffineTransform(3.0f,0.0f,0.0f,3.0f,0.0f,0.0f);
        //G.setTransform(atx);
        gridCanvas.gridImage.clearRect(0, 0, 1800, 1800);
        if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
        }
        drawAllComps();
        drawDrawingComps();
        if (!pageTitle.equals("")) {
      drawPageTitle();
        }
        gridCanvas.redraw();*/

  }

////////////////////////////////////////////////////////////////////////////
  private void zoomOut_action_performed(ActionEvent ae) {
    /*Graphics2D G = (Graphics2D) gridCanvas.gridImage;
        ActiveState.currentXScale -= 0.1;
        ActiveState.currentYScale -= 0.1;
        G.scale(ActiveState.currentXScale, ActiveState.currentYScale);
        gridCanvas.gridImage.clearRect(0, 0, 1800, 1800);
        if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
        }
        drawAllComps();
        drawDrawingComps();
```

```
        if (!pageTitle.equals("")) {
      drawPageTitle();
        }
        gridCanvas.redraw();*/
  }

  //************BUTTONS EVENT FUNCTIONS************//

////////////////////////////////////////////////////////////////////////////
  private void btns_action_performed(ActionEvent ae) {

    JButton btn = (JButton) ae.getSource();
    String command = btn.getActionCommand();

    if (command.equals("Resistor") || command.equals("Inductor")
        || command.equals("Capacitor") || command.equals("Transformer")) {
      ActiveState.activePanel = 0;
    }

    else if (command.equals("DC Voltage Source") ||
            command.equals("DC Current Source")
            || command.equals("AC Voltage Source") ||
            command.equals("AC Current Source")) {
      ActiveState.activePanel = 1;
    }
    else if (command.equals("Voltage Controlled Voltage Source") ||
            command.equals("Voltage Controlled Current Source")
            || command.equals("Current Controlled Voltage Source") ||
            command.equals("Current Controlled Current Source")) {
      ActiveState.activePanel = 2;
    }
    else if (command.equals("Connector") || command.equals("Ground")
            || command.equals("Current Marker") ||
            command.equals("Voltage Differential Marker")
            || command.equals("Connection Point") ||
            command.equals("Voltage Marker")) {
      ActiveState.activePanel = 3;
    }
    else if (command.equals("PPTransformer") || command.equals("PNTransformer")
            || command.equals("NPTransformer") ||
            command.equals("NNTransformer")) {
      ActiveState.activePanel = 4;
    }

    activateElementsPanel();
```

```
int index = 0;
Components tempcomp = null;
for (int i = 0; i < componentsCount; i++) {
  tempcomp = ActiveState.createNew(i);

  if (command.equals(tempcomp.getGenericName())) {
    if (tempcomp.getComponentType() != ActiveState.TRANSFORMER) {

      ActiveState.CurComp = tempcomp;
      index = i;

    }
    else {
      ActiveState.CurComp = null;
      transformersPanel.setLocation(10, 365);
      transformersPanel.setVisible(true);
      //dependentPanel.setVisible(false);
    }
  }

}

ActiveState.active = false;
ActiveState.multiActive = false;
for (int type = 0; type < componentsCount; type++) {
  for (int count = 0; count < Array.getLength(allComps[type]); count++) {
    if (allComps[type][count] != null) {
      allComps[type][count].setActiveState(false);
      allComps[type][count].setNameState(false);
      allComps[type][count].setValueState(false);
      int numbNodes = allComps[type][count].getNumNodes();
      for (int n = 0; n < numbNodes; n++) {
        allComps[type][count].getNode(n).setActiveState(false);
      }
    }
  }
}

if (!command.equals("Transformer")) {

  int size = btns.size();

  if (size == 0) {
```

```
  getVisitedActionButton(command).setLocation(0, y);
  getVisitedActionButton(command).setVisible(true);
  getVisitedActionButton(command).addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      btns_action_performed(ae);
    }
  }
  );
  getVisitedActionButton(command).addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent me) {
      btns_mousePressed(me);
    }
  }
  );

  y += 22;
  btns.add(getVisitedActionButton(command));

 }
 else {
  boolean btnExist = false;
  for (int i = 0; i < btns.size(); i++) {
    JButton btnset = (JButton) btns.get(i);
    String com = btnset.getActionCommand();
    if (com.equals(command)) {
      btnExist = true;
    }
  }

  if (!btnExist) {

    getVisitedActionButton(command).setLocation(0, y);
    getVisitedActionButton(command).setVisible(true);
    getVisitedActionButton(command).addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        btns_action_performed(ae);
      }
    }
    );
    getVisitedActionButton(command).addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent me) {
        btns_mousePressed(me);
      }
    }
    );
```

```
        btns.add(getVisitedActionButton(command));
        y += 22;
       }

     }

   } ////

   redrawGridCanvas();
   gridCanvas.requestFocus();
  }

////////////////////////////////////////////////////////////////////////////
  /*  private void btns_mousePressed(MouseEvent me)
    {
   int x=me.getX();
     int y=me.getY();
   System.out.println(x+","+y);
     JButton btn = (JButton) me.getSource();
     String command = btn.getActionCommand();
     System.out.println(command);
   }*/
  /*
////////////////////////////////////////////////////////////////////////////
    private void btns_mouseReleased(MouseEvent me) {
     JButton btn = (JButton) me.getSource();
     btn.setBackground(Color.WHITE);
    }
   */
////////////////////////////////////////////////////////////////////////////
  private void btns_mousePressed(MouseEvent me) {

    JButton btn = (JButton) me.getSource();
    String command = btn.getActionCommand();

    if (command.equals("Resistor") || command.equals("Inductor")
        || command.equals("Capacitor") || command.equals("Transformer")) {
      ActiveState.activePanel = 0;
    }

    else if (command.equals("DC Voltage Source") ||
            command.equals("DC Current Source")
            || command.equals("AC Voltage Source") ||
            command.equals("AC Current Source")) {
```

```
  ActiveState.activePanel = 1;
}
else if (command.equals("Voltage Controlled Voltage Source") ||
        command.equals("Voltage Controlled Current Source")
        || command.equals("Current Controlled Voltage Source") ||
        command.equals("Current Controlled Current Source")) {
  ActiveState.activePanel = 2;
}
else if (command.equals("Connector") || command.equals("Ground")
        || command.equals("Current Marker") ||
        command.equals("Voltage Differential Marker")
        || command.equals("Connection Point") ||
        command.equals("Voltage Marker")) {
  ActiveState.activePanel = 3;
}
else if (command.equals("PPTransformer") || command.equals("PNTransformer")
        || command.equals("NPTransformer") ||
        command.equals("NNTransformer")) {
  ActiveState.activePanel = 4;
}

activateElementsPanel();

int index = 0;
Components tempcomp = null;
for (int i = 0; i < componentsCount; i++) {
  tempcomp = ActiveState.createNew(i);

  if (command.equals(tempcomp.getGenericName())) {
    if (tempcomp.getComponentType() != ActiveState.TRANSFORMER) {

      ActiveState.CurComp = tempcomp;
      index = i;

    }
    else {
      ActiveState.CurComp = null;
      transformersPanel.setLocation(10, 365);
      transformersPanel.setVisible(true);
      //dependentPanel.setVisible(false);
    }
  }

}
```

```
ActiveState.active = false;
ActiveState.multiActive = false;
for (int type = 0; type < componentsCount; type++) {
  for (int count = 0; count < Array.getLength(allComps[type]); count++) {
    if (allComps[type][count] != null) {
      allComps[type][count].setActiveState(false);
      allComps[type][count].setNameState(false);
      allComps[type][count].setValueState(false);
      int numbNodes = allComps[type][count].getNumNodes();
      for (int n = 0; n < numbNodes; n++) {
        allComps[type][count].getNode(n).setActiveState(false);
      }
    }
  }
}

if (!command.equals("Transformer")) {

  int size = btns.size();

  if (size == 0) {

    getVisitedActionButton(command).setLocation(0, y);
    getVisitedActionButton(command).setVisible(true);
    getVisitedActionButton(command).addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        btns_action_performed(ae);
      }
    }
    );
    getVisitedActionButton(command).addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent me) {
        btns_mousePressed(me);
      }
    }
    );

    y += 22;
    btns.add(getVisitedActionButton(command));

  }
  else {
    boolean btnExist = false;
    for (int i = 0; i < btns.size(); i++) {
      JButton btnset = (JButton) btns.get(i);
```

```
        String com = btnset.getActionCommand();
        if (com.equals(command)) {
          btnExist = true;
        }
      }

      if (!btnExist) {

        getVisitedActionButton(command).setLocation(0, y);
        getVisitedActionButton(command).setVisible(true);
        getVisitedActionButton(command).addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent ae) {
            btns_action_performed(ae);
          }
        }
        );
        getVisitedActionButton(command).addMouseListener(new MouseAdapter() {
          public void mousePressed(MouseEvent me) {
            btns_mousePressed(me);
          }
        }
        );

        btns.add(getVisitedActionButton(command));
        y += 22;
      }

    }

  } ////

  redrawGridCanvas();
  gridCanvas.requestFocus();
  gridCanvas.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
 }

////////////////////////////////////////////////////////////////////////////
 private void btns_mouseEntered(MouseEvent me) {

  JButton btn = (JButton) me.getSource();
  String btnCmd = (String) btn.getActionCommand();

  if (btnCmd.equals("passiveBtn")) {
    btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                        PASSIVEELEMENTS]));
```

```
}
else if (btnCmd.equals("independentBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    INDEPENDENTSOURCES]));
}
else if (btnCmd.equals("dependentBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    DEPENDENTSOURCES]));
}
else if (btnCmd.equals("connectionsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.CONNECTIONS]));
}
else if (btnCmd.equals("saveSchematicBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    SAVESCHEMATIC]));
}
else if (btnCmd.equals("openSchematicBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    OPENSCHEMATIC]));
}
else if (btnCmd.equals("newSchematicBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    NEWSCHEMATIC]));
}
else if (btnCmd.equals("createNetlistBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    CREATENETLIST]));
}
else if (btnCmd.equals("simulateBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SIMULATE]));
}
else if (btnCmd.equals("netlistEditingBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    NETLISTEDITING]));
}
else if (btnCmd.equals("simSettingsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SIMSETTINGS]));
}
else if (btnCmd.equals("fileManagerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.FILEMANAGER]));
}
else if (btnCmd.equals("deleteSchematicBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    DELETESCHEMATIC]));
}
```

```
else if (btnCmd.equals("newBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW]));
}
else if (btnCmd.equals("openBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.OPEN]));
}
else if (btnCmd.equals("saveBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SAVE]));
}
else if (btnCmd.equals("clearAllCompsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    CLEARALLCOMPS]));
}
else if (btnCmd.equals("cutBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.CUT]));
}
else if (btnCmd.equals("copyBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.COPY]));
}
else if (btnCmd.equals("pasteBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.PASTE]));
}
else if (btnCmd.equals("undoBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.UNDO]));
}
else if (btnCmd.equals("redoBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.REDO]));
}
else if (btnCmd.equals("deleteBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.REMOVE]));
}
else if (btnCmd.equals("contactUsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.CONTACTUS]));
}
else if (btnCmd.equals("selectAllCompsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    SELECTALLCOMPS]));
}
else if (btnCmd.equals("deletePageBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.DELETEPAGE]));
}
else if (btnCmd.equals("runBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.RUN]));
}
else if (btnCmd.equals("handBtn")) {
```

```
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.HAND]));
}
else if (btnCmd.equals("dCursorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.DCURSOR]));
}
else if (btnCmd.equals("zoomInBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ZOOMIN]));
}
else if (btnCmd.equals("fitZoomBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.FITZOOM]));
}

else if (btnCmd.equals("zoomOutBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ZOOMOUT]));
}
else if (btnCmd.equals("helpBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.HELP]));
}
else if (btnCmd.equals("serverFileManagerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    SERVERFILEMANAGER]));
}
else if (btnCmd.equals("Resistor")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SRESISTOR]));
}
else if (btnCmd.equals("Inductor")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SINDUCTOR]));
}
else if (btnCmd.equals("Capacitor")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SCAPACITOR]));
}
else if (btnCmd.equals("Transformer")) {

  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    STRANSFORMER]));
}
else if (btnCmd.equals("PPTransformer")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    STRANSFORMERA]));
}
else if (btnCmd.equals("PNTransformer")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    STRANSFORMERB]));
}
else if (btnCmd.equals("NPTransformer")) {
```

```
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    STRANSFORMERC]));
}
else if (btnCmd.equals("NNTransformer")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    STRANFORMERD]));
}
else if (btnCmd.equals("DC Voltage Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SDCVSOURCE]));

}
else if (btnCmd.equals("DC Current Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SDCCSOURCE]));
}
else if (btnCmd.equals("AC Voltage Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SACVSOURCE]));
}
else if (btnCmd.equals("AC Current Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SACCSOURCE]));
}
else if (btnCmd.equals("Current Controlled Current Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SDEPCCCS]));
}
else if (btnCmd.equals("Current Controlled Voltage Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SDEPCCVS]));
}
else if (btnCmd.equals("Voltage Controlled Current Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SDEPVCCS]));
}
else if (btnCmd.equals("Voltage Controlled Voltage Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SDEPVCVS]));
}
else if (btnCmd.equals("Connector")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SCONNECTOR]));
}

else if (btnCmd.equals("Current Marker")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.CURRENTM]));
}
else if (btnCmd.equals("Ground")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SGROUND]));
}
else if (btnCmd.equals("Connection Point")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SCONNODE]));
}
```

```
else if (btnCmd.equals("Voltage Differential Marker")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SNODEL]));
}
else if (btnCmd.equals("Voltage Marker")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.VOLTAGEM]));
}
else if (btnCmd.equals("new1Btn")) {
  deActivatePages();
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW1]));
}
else if (btnCmd.equals("new2Btn")) {
  deActivatePages();
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW2]));
}
else if (btnCmd.equals("new3Btn")) {
  deActivatePages();
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW3]));
}
else if (btnCmd.equals("new4Btn")) {
  deActivatePages();
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW4]));
}
else if (btnCmd.equals("new5Btn")) {
  deActivatePages();
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW5]));
}
else if (btnCmd.equals("insertTextBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.TEXT]));
}
else if (btnCmd.equals("fillColorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.FILLCOLOR]));
}
else if (btnCmd.equals("lineColorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.LINECOLOR]));
}
else if (btnCmd.equals("canvasPropertiesBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    CANVASPROPERTIES]));
}
else if (btnCmd.equals("dftCursorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.DCURSOR]));
}
else if (btnCmd.equals("rectangleBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SQUARE]));
}
```

```
else if (btnCmd.equals("circleBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.CIRCLE]));
}
else if (btnCmd.equals("lineBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.LINE]));
}
else if (btnCmd.equals("goToBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.GOTO]));
}
else if (btnCmd.equals("canvasSizeBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.CANVASSIZE]));
}
else if (btnCmd.equals("canvasColorsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    CANVASCOLORS]));
}
else if (btnCmd.equals("lastForwardBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.LASTFORWARD]));
}
else if (btnCmd.equals("forwardBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.FORWARD]));
}
else if (btnCmd.equals("backwardBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.BACKWARD]));
}
else if (btnCmd.equals("lastBackwardBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    LASTBACKWARD]));
}
else if (btnCmd.equals("refreshBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.REFRESH]));
}
else if (btnCmd.equals("gridBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.GRID]));
}
else if (btnCmd.equals("simulationSettingsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SETTINGS]));
}
else if (btnCmd.equals("netListBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NETLIST]));
}
else if (btnCmd.equals("editNetlistBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.EDITNETLIST]));
}
else if (btnCmd.equals("showVoltagesBtn")) {
```

```
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    SHOWVOLTAGES]));
}
else if (btnCmd.equals("showCurrentsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    SHOWCURRENTS]));
}
else if (btnCmd.equals("showHideNodesBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    SHOWHIDENODES]));
}
else if (btnCmd.equals("showCIRMLBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.SHOWCIRML]));
}
else if (btnCmd.equals("voltageMarkerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    VOLTAGEMARKER]));
}
else if (btnCmd.equals("currentMarkerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    CURRENTMARKER]));
}
else if (btnCmd.equals("voltageDifferentialMarkerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    VOLTAGEDIFFERENTIALMARKER]));
}
else if (btnCmd.equals("drawResultBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.DRAWRESULT]));
}
else if (btnCmd.equals("showOutputResultBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    SHOWOUTPUTRESULT]));
}
else if (btnCmd.equals("textColorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.TEXTCOLOR]));
}
else if (btnCmd.equals("roundRectangleBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    ROUNDRECTANGLE]));
}
else if (btnCmd.equals("arcBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ARC]));
}
else if (btnCmd.equals("displayPropertiesBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
```

```
                    DISPLAYPROPERTIES]));
}
else if (btnCmd.equals("editComponentBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    TOOLBAREDITPROPERTIES]));
}
else if (btnCmd.equals("rotateRightBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ROTATERIGHT]));
}
else if (btnCmd.equals("rotateLeftBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ROTATELEFT]));
}
else if (btnCmd.equals("flipVerticalBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    FLIPVERTICAL]));
}
else if (btnCmd.equals("flipHorizontalBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                    FLIPHORIZANTAL]));
}
else if (btnCmd.equals("rotateTextBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ROTATETEXT]));
}
else if (btnCmd.equals("groupComponentsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.GROUP]));
}
else if (btnCmd.equals("regroupComponentsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.REGROUP]));
}
else if (btnCmd.equals("ungroupComponentsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.UNGROUP]));
}
else if (btnCmd.equals("alignRightBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ALIGNRIGHT]));
}
else if (btnCmd.equals("alignLeftBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ALIGNLEFT]));
}
else if (btnCmd.equals("alignTopBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ALIGNUP]));
}
else if (btnCmd.equals("alignBottomBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.ALIGNDOWN]));
}
```

```
  }

////////////////////////////////////////////////////////////////////////////
  private void btns_mouseExited(MouseEvent me) {
    JButton btn = (JButton) me.getSource();
    String btnCmd = (String) btn.getActionCommand();

    if (btnCmd.equals("passiveBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                  PASSIVEELEMENTS]));
    }
    else if (btnCmd.equals("independentBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                  INDEPENDENTSOURCES]));
    }
    else if (btnCmd.equals("dependentBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                  DEPENDENTSOURCES]));
    }
    else if (btnCmd.equals("connectionsBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.CONNECTIONS]));
    }
    else if (btnCmd.equals("saveSchematicBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                  SAVESCHEMATIC]));
    }
    else if (btnCmd.equals("openSchematicBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                  OPENSCHEMATIC]));
    }
    else if (btnCmd.equals("newSchematicBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEWSCHEMATIC]));
    }
    else if (btnCmd.equals("createNetlistBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                  CREATENETLIST]));
    }
    else if (btnCmd.equals("simulateBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SIMULATE]));
    }
    else if (btnCmd.equals("netlistEditingBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                  NETLISTEDITING]));
    }
    else if (btnCmd.equals("simSettingsBtn")) {
```

```
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SIMSETTINGS]));
}
else if (btnCmd.equals("fileManagerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.FILEMANAGER]));
}
else if (btnCmd.equals("deleteSchematicBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    DELETESCHEMATIC]));
}
else if (btnCmd.equals("newBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW]));
}
else if (btnCmd.equals("openBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.OPEN]));
}
else if (btnCmd.equals("saveBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SAVE]));
}
else if (btnCmd.equals("clearAllCompsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    CLEARALLCOMPS]));
}
else if (btnCmd.equals("cutBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.CUT]));
}
else if (btnCmd.equals("copyBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.COPY]));
}
else if (btnCmd.equals("pasteBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.PASTE]));
}
else if (btnCmd.equals("undoBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.UNDO]));
}
else if (btnCmd.equals("redoBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.REDO]));
}
else if (btnCmd.equals("deleteBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.REMOVE]));
}
else if (btnCmd.equals("contactUsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.CONTACTUS]));
}
else if (btnCmd.equals("selectAllCompsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
```

```
                    SELECTALLCOMPS]));
}
else if (btnCmd.equals("deletePageBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.DELETEPAGE]));
}
else if (btnCmd.equals("runBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.RUN]));
}
else if (btnCmd.equals("handBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.HAND]));

}
else if (btnCmd.equals("dCursorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.DCURSOR]));
}
else if (btnCmd.equals("zoomInBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ZOOMIN]));
}
else if (btnCmd.equals("fitZoomBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.FITZOOM]));
}
else if (btnCmd.equals("zoomOutBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ZOOMOUT]));
}
else if (btnCmd.equals("helpBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.HELP]));
}
else if (btnCmd.equals("serverFileManagerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    SERVERFILEMANAGER]));
}
else if (btnCmd.equals("Resistor")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SRESISTOR]));
}
else if (btnCmd.equals("Inductor")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SINDUCTOR]));
}
else if (btnCmd.equals("Capacitor")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SCAPACITOR]));
}
else if (btnCmd.equals("Transformer")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.STRANSFORMER]));
}
else if (btnCmd.equals("PPTransformer")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
```

```
                    STRANSFORMERA]));
}
else if (btnCmd.equals("PNTransformer")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    STRANSFORMERB]));
}
else if (btnCmd.equals("NPTransformer")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    STRANSFORMERC]));
}
else if (btnCmd.equals("NNTransformer")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.STRANFORMERD]));
}
else if (btnCmd.equals("DC Voltage Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SDCVSOURCE]));

}
else if (btnCmd.equals("DC Current Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SDCCSOURCE]));
}
else if (btnCmd.equals("AC Voltage Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SACVSOURCE]));
}
else if (btnCmd.equals("AC Current Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SACCSOURCE]));
}
else if (btnCmd.equals("Current Controlled Current Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SDEPCCCS]));
}
else if (btnCmd.equals("Current Controlled Voltage Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SDEPCCVS]));
}
else if (btnCmd.equals("Voltage Controlled Current Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SDEPVCCS]));
}
else if (btnCmd.equals("Voltage Controlled Voltage Source")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SDEPVCVS]));
}
else if (btnCmd.equals("Connector")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SCONNECTOR]));
}
else if (btnCmd.equals("Current Marker")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.CURRENTM]));
}
else if (btnCmd.equals("Ground")) {
```

```
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SGROUND]));
}
else if (btnCmd.equals("Connection Point")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SCONNODE]));
}
else if (btnCmd.equals("Voltage Differential Marker")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SNODEL]));
}
else if (btnCmd.equals("Voltage Marker")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.VOLTAGEM]));
}
else if (btnCmd.equals("new1Btn")) {
  if (!ActiveState.page1Active) {
    btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW1]));
  }
}
else if (btnCmd.equals("new2Btn")) {
  if (!ActiveState.page2Active) {
    btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW2]));
  }
}
else if (btnCmd.equals("new3Btn")) {
  if (!ActiveState.page3Active) {
    btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW3]));
  }
}
else if (btnCmd.equals("new4Btn")) {
  if (!ActiveState.page4Active) {
    btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW4]));
  }
}
else if (btnCmd.equals("new5Btn")) {
  if (!ActiveState.page5Active) {
    btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW5]));
  }
}
else if (btnCmd.equals("insertTextBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.TEXT]));
}
else if (btnCmd.equals("fillColorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.FILLCOLOR]));
}
else if (btnCmd.equals("lineColorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.LINECOLOR]));
}
```

```
else if (btnCmd.equals("canvasPropertiesBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                        CANVASPROPERTIES]));
}
else if (btnCmd.equals("dftCursorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.DCURSOR]));
}
else if (btnCmd.equals("rectangleBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SQUARE]));
}
else if (btnCmd.equals("circleBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.CIRCLE]));
}
else if (btnCmd.equals("lineBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.LINE]));
}
else if (btnCmd.equals("goToBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.GOTO]));
}
else if (btnCmd.equals("canvasColorsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.CANVASCOLORS]));
}
else if (btnCmd.equals("canvasSizeBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.CANVASSIZE]));
}
else if (btnCmd.equals("lastForwardBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.LASTFORWARD]));
}
else if (btnCmd.equals("forwardBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.FORWARD]));
}
else if (btnCmd.equals("backwardBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.BACKWARD]));
}
else if (btnCmd.equals("lastBackwardBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.LASTBACKWARD]));
}
else if (btnCmd.equals("refreshBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.REFRESH]));
}
else if (btnCmd.equals("gridBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.GRID]));
}
else if (btnCmd.equals("simulationSettingsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SETTINGS]));
```

```
}
else if (btnCmd.equals("netListBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NETLIST]));
}
else if (btnCmd.equals("editNetlistBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.EDITNETLIST]));
}
else if (btnCmd.equals("showVoltagesBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SHOWVOLTAGES]));
}
else if (btnCmd.equals("showCurrentsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SHOWCURRENTS]));
}
else if (btnCmd.equals("showHideNodesBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    SHOWHIDENODES]));
}
else if (btnCmd.equals("showCIRMLBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.SHOWCIRML]));
}
else if (btnCmd.equals("voltageMarkerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    VOLTAGEMARKER]));
}
else if (btnCmd.equals("currentMarkerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    CURRENTMARKER]));
}
else if (btnCmd.equals("voltageDifferentialMarkerBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    VOLTAGEDIFFERENTIALMARKER]));
}
else if (btnCmd.equals("drawResultBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.DRAWRESULT]));
}
else if (btnCmd.equals("showOutputResultBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    SHOWOUTPUTRESULT]));
}
else if (btnCmd.equals("textColorBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.TEXTCOLOR]));
}
else if (btnCmd.equals("roundRectangleBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    ROUNDRECTANGLE]));
```

```
}
else if (btnCmd.equals("arcBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ARC]));
}
else if (btnCmd.equals("displayPropertiesBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    DISPLAYPROPERTIES]));
}
else if (btnCmd.equals("editComponentBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    TOOLBAREDITPROPERTIES]));
}
else if (btnCmd.equals("rotateRightBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ROTATERIGHT]));
}
else if (btnCmd.equals("rotateLeftBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ROTATELEFT]));
}
else if (btnCmd.equals("flipVerticalBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.FLIPVERTICAL]));
}
else if (btnCmd.equals("flipHorizontalBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                    FLIPHORIZANTAL]));
}
else if (btnCmd.equals("rotateTextBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ROTATETEXT]));
}
else if (btnCmd.equals("groupComponentsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.GROUP]));
}
else if (btnCmd.equals("ungroupComponentsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.UNGROUP]));
}
else if (btnCmd.equals("regroupComponentsBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.REGROUP]));
}
else if (btnCmd.equals("alignRightBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ALIGNRIGHT]));
}
else if (btnCmd.equals("alignLeftBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ALIGNLEFT]));
}
else if (btnCmd.equals("alignTopBtn")) {
  btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ALIGNUP]));
```

```
    }
    else if (btnCmd.equals("alignBottomBtn")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.ALIGNDOWN]));
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void elementsPanel_mouse_dragged(MouseEvent me) {
    int x = me.getX();
    int y = me.getY();
    if (x - 20 < 30) {
      x = 22;
    }
    if (x - 20 > 65) {
      x = 90;
    }
    if (y < 2) {
      y = 2;
    }
    if (y > 395) {
      y = 395;
    }
    elementsPanel.setCursor(new Cursor(Cursor.MOVE_CURSOR));
    switch (ActiveState.activePanel) {
      case 0:
        passivePanel.setLocation(x - 20, y);
        passivePanel.setFocusable(true);
        break;
      case 1:
        independentPanel.setLocation(x - 20, y);
        independentPanel.setFocusable(true);

        break;
      case 2:
        dependentPanel.setLocation(x - 20, y);
        dependentPanel.setFocusable(true);

        break;
      case 3: {
        if (y < 2) {
          y = 2;
        }
        if (y > 370) {
          y = 370;
```

```
      }
      connectionsPanel.setLocation(x - 20, y);
     }
     break;
     case 4: {
      transformersPanel.setLocation(x - 20, y);

     }
     break;
     default:
      elementsPanel.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }
   }

////////////////////////////////////////////////////////////////////////////
  private void elementsPanel_mouse_pressed(MouseEvent me) {
    int x = me.getX();
    int y = me.getY();
    java.awt.Rectangle loc1 = passivePanel.getBounds();
    java.awt.Rectangle loc2 = independentPanel.getBounds();
    java.awt.Rectangle loc3 = dependentPanel.getBounds();
    java.awt.Rectangle loc4 = connectionsPanel.getBounds();
    java.awt.Rectangle loc5 = transformersPanel.getBounds();

    if ( (x > loc1.x && x < loc1.x + loc1.width) &
       (y > loc1.y && y < loc1.y + loc1.width)) {
     ActiveState.activePanel = 0;
    }
    else if ( (x > loc2.x && x < loc2.x + loc2.width) &
          (y > loc2.y && y < loc2.y + loc2.width)) {
     ActiveState.activePanel = 1;
    }
    else if ( (x > loc3.x && x < loc3.x + loc3.width) &
          (y > loc3.y && y < loc3.y + loc3.width)) {
     ActiveState.activePanel = 2;
    }
    else if ( (x > loc4.x && x < loc4.x + loc4.width) &
          (y > loc4.y && y < loc4.y + loc4.width)) {
     ActiveState.activePanel = 3;
    }
    else if ( (x > loc5.x && x < loc5.x + loc5.width) &
          (y > loc5.y && y < loc5.y + loc5.width)) {
     ActiveState.activePanel = 4;
    }
    else {
```

```
      ActiveState.activePanel = -1;
    }

    activateElementsPanel();
  }

////////////////////////////////////////////////////////////////////////////
  private void elementsPanel_mouse_released(MouseEvent me) {

    elementsPanel.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    ActiveState.activePanel = -1;
  }

////////////////////////////////////////////////////////////////////////////
  private void elementsPanel_mouse_clicked(MouseEvent me) {

    int x = me.getX();
    int y = me.getY();

    if (SwingUtilities.isRightMouseButton(me)) {
      java.awt.Rectangle loc1 = passivePanel.getBounds();
      java.awt.Rectangle loc2 = independentPanel.getBounds();
      java.awt.Rectangle loc3 = dependentPanel.getBounds();
      java.awt.Rectangle loc4 = connectionsPanel.getBounds();
      java.awt.Rectangle loc5 = transformersPanel.getBounds();
      if ( (x > loc1.x && x < loc1.x + loc1.width) &
          (y > loc1.y && y < loc1.y + loc1.width)) {
        ActiveState.activePanel = 0;
      }
      else if ( (x > loc2.x && x < loc2.x + loc2.width) &
            (y > loc2.y && y < loc2.y + loc2.width)) {
        ActiveState.activePanel = 1;
      }
      else if ( (x > loc3.x && x < loc3.x + loc3.width) &
            (y > loc3.y && y < loc3.y + loc3.width)) {
        ActiveState.activePanel = 2;
      }
      else if ( (x > loc4.x && x < loc4.x + loc4.width) &
            (y > loc4.y && y < loc4.y + loc4.width)) {
        ActiveState.activePanel = 3;
      }
      else if ( (x > loc5.x && x < loc5.x + loc5.width) &
            (y > loc5.y && y < loc5.y + loc5.width)) {
        ActiveState.activePanel = 4;
      }
```

```
      else {
        ActiveState.activePanel = -1;
      }
      if (ActiveState.activePanel != -1) {
        menuItem10.setEnabled(true);
        closeMenu.show(elementsPanel, x, y);
      }
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void mouse_entered_action(MouseEvent me) {

    JButton btn = (JButton) me.getSource();
    String cmd = btn.getActionCommand();

    if (cmd.equals("passive")) {

      Image img = getImage(getDocumentBase(),
                    "ECS/images/passive Elements_active.jpg");
      btn = new JButton(new ImageIcon(img));

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void mouse_exited_action(MouseEvent me) {
    JButton btn = (JButton) me.getSource();
    btn.setBackground(new Color(179, 186, 241));
  }

////////////////////////////////////////////////////////////////////////////
  private void btn_closed_action(MouseEvent me) {
    JButton btn = (JButton) me.getSource();
    String command = btn.getActionCommand();
    btn.setFocusable(false);

    if (command.equals("passive")) {
      passivePanel.setVisible(false);
    }
    else if (command.equals("independent")) {
      independentPanel.setVisible(false);
    }
```

```
    else if (command.equals("dependent")) {
      dependentPanel.setVisible(false);
    }
    else if (command.equals("connections")) {
      connectionsPanel.setVisible(false);
    }
    else {
      transformersPanel.setVisible(false);
    }
  }

  public static Object growArray(Object a) {
    Class cl = a.getClass();
    if (!cl.isArray()) {
      return null;
    }
    Class componentType = a.getClass().getComponentType();
    int length = Array.getLength(a);
    int newLength = length + 1;

    Object newArray = Array.newInstance(componentType,
                                        newLength);
    System.arraycopy(a, 0, newArray, 0, length);

    return newArray;
  }

////////////////////////////////////////////////////////////////////////////
  private void fc_action_performed(ActionEvent ae) {

  }

////////////////////////////////////////////////////////////////////////////

  static Object cutArray(Object a) {
    Class cl = a.getClass();
    if (!cl.isArray()) {
      return null;
    }
    Class componentType = a.getClass().getComponentType();
    int newLength = 1;
    Object newArray = Array.newInstance(componentType,
                                        newLength);

    return newArray;
```

```
  }

////////////////////////////////////////////////////////////////////////////

  static Object shrinkArray(Object a, int n) {
    Class cl = a.getClass();
    if (!cl.isArray()) {
      return null;
    }
    Class componentType = a.getClass().getComponentType();
    int length = Array.getLength(a);
    int newLength = length - 1;
    Object newArray = Array.newInstance(componentType,
                                        newLength);
    System.arraycopy(a, 0, newArray, 0, n);
    System.arraycopy(a, n + 1, newArray, n, length - 1 - n);

    return newArray;
  }

  ////////////////////////////////////////////////////////////////////////////

//private void enterValue_action_performed(ActionEvent ae){;}
//private void singleElementRotateRight_action_performed(ActionEvent ae){;}//
//private void singleElementRotateLeft_action_performed(ActionEvent ae){;}

//*************************************************************************/
/
//                    END OF EVENTS HANDLING FUNCTIONS                    //
//*************************************************************************/
/

//*************************************************************************/
/

  public void start() {

    start = true;

    InfoMessage getStart = new InfoMessage(
        "You can draw components using the left GUI panels");
    getStart.show();

    WelcomeMessage welcome = new WelcomeMessage();
    welcome.show();
```

```
  }

//*************************************************************************/
/

//*************************************************************************/
/
//                          MEMBER FUNCTIONS                          //
//*************************************************************************/
/
// Add Components to the ContentPane
  private void add(Component c, int x, int y, int width, int height) {
    c.setBounds(x, y, width, height);
    this.getContentPane().add(c);
  }

////////////////////////////////////////////////////////////////////////////
// Add Components to the Panel
  private void addToPanel(JPanel pnl, Component c, int x, int y, int width,
                          int height) {
    c.setBounds(x, y, width, height);
    pnl.add(c);
  }

////////////////////////////////////////////////////////////////////////////
  private void addImageButton(final JPanel panel, JButton btn, int index, int x,
                              int y,
                              int width, int height) {
    String compName;
    Image img;
    Components tempComp = null;

    if (index != ActiveState.NODEL) {
      tempComp = ActiveState.createNew(index);
    }
    else {
      tempComp = new NodeL();
    }
    img = this.getImage(getDocumentBase(),
                        "ECS/images/" + tempComp.getImageName());

    btn = new JButton(new ImageIcon(img));
    //btn.setMargin(new Insets(0, 0, 0, 0));
    btn.setBorder(null);
```

```
  btn.setToolTipText(tempComp.getGenericName());
  btn.setBounds(x, y, width, height);
  btn.setActionCommand(tempComp.getGenericName());
  btn.addMouseListener(btnsMouseListener);
  btn.addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent me) {
      btns_mousePressed(me);
    }
  });

  btn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      btns_action_performed(ae);
    }
  });

  /*
     bar.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
         // This method is called whenever the orientation of the toolbar is changed
         public void propertyChange(java.beans.PropertyChangeEvent evt) {
             String propName = evt.getPropertyName();
             if ("orientation".equals(propName)) {
                 // Get the old orientation
                 Integer oldValue = (Integer)evt.getOldValue();
                 // Get the new orientation
                 Integer newValue = (Integer)evt.getNewValue();
                 if (newValue.intValue() == JToolBar.HORIZONTAL) {
                   //bar.setLayout(new BorderLayout());
                   bar.setOrientation(1);
                 } else {
                    // toolbar now has vertical orientation
                 }
             }
         }
     });*/

  //bar.setRollover(true);
  //bar.addSeparator(new Dimension(1,1));

  panel.add(btn);

  tempComp = null;
 }
```

```
////////////////////////////////////////////////////////////////////////////
  private void drawToolsPanel() {

    elementsPanel = new GraphicalPanel();
    elementsPanel.setLayout(null);

    // elementsPanel.setBackground(new Color(223, 226, 248));

    //elementsPanel.setBorder(BorderFactory.createEtchedBorder
    //                    (Color.WHITE, Color.BLUE));
    add(elementsPanel, 660, 40, 105, 527);

    toolsPanel = new JPanel();
    toolsPanel.setBounds(765, 40, 150, 250);
    toolsPanel.setBackground(new Color(117, 159, 255));
    toolsPanel.setLayout(null);
    toolsPanel.setBorder(BorderFactory.createEtchedBorder
                    (Color.BLUE, Color.WHITE));

    this.getContentPane().add(toolsPanel);

    Image img = getImage(getDocumentBase(), "ECS/images/Drawing Tools.jpg");
    toolLabel = new JLabel(new ImageIcon(img));
    addToPanel(toolsPanel, toolLabel, 0, 0, 155, 65);

    img = ActiveState.btnsExited[ActiveState.PASSIVEELEMENTS];
    passiveBtn = new JButton(new ImageIcon(img));
    passiveBtn.setActionCommand("passiveBtn");
    passiveBtn.setBorder(null);
    addToPanel(toolsPanel, passiveBtn, 5, 60, 140, 30);

    img = img = ActiveState.btnsExited[ActiveState.INDEPENDENTSOURCES];
    independentBtn = new JButton(new ImageIcon(img));
    independentBtn.setActionCommand("independentBtn");
    independentBtn.setBackground(new Color(179, 186, 241));
    independentBtn.setFont(myFont);
    independentBtn.setBorder(null);
    independentBtn.setForeground(Color.WHITE);
    addToPanel(toolsPanel, independentBtn, 5, 110, 140, 30);

    img = img = ActiveState.btnsExited[ActiveState.DEPENDENTSOURCES];
    dependentBtn = new JButton(new ImageIcon(img));
    dependentBtn.setActionCommand("dependentBtn");
    dependentBtn.setBackground(new Color(179, 186, 241));
    dependentBtn.setFont(myFont);
```

```
    dependentBtn.setBorder(null);
    dependentBtn.setForeground(Color.WHITE);
    addToPanel(toolsPanel, dependentBtn, 5, 160, 140, 30);

    img = img = ActiveState.btnsExited[ActiveState.CONNECTIONS];
    connectionsBtn = new JButton(new ImageIcon(img));
    connectionsBtn.setActionCommand("connectionsBtn");
    connectionsBtn.setBackground(new Color(179, 186, 241));
    connectionsBtn.setFont(myFont);
    connectionsBtn.setBorder(null);
    connectionsBtn.setForeground(Color.WHITE);
    addToPanel(toolsPanel, connectionsBtn, 5, 210, 140, 30);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawFileManagerPanel() {

    fileManagerPanel = new JPanel();
    fileManagerPanel.setBounds(765, 290, 150, 300);
    fileManagerPanel.setBackground(new Color(117, 159, 255));
    fileManagerPanel.setLayout(null);
    fileManagerPanel.setBorder(BorderFactory.createEtchedBorder
                        (Color.BLUE, Color.WHITE));
    this.getContentPane().add(fileManagerPanel);

    Image img = getImage(getDocumentBase(), "ECS/images/File Manager Tools.jpg");
    toolLabel = new JLabel(new ImageIcon(img));
    addToPanel(fileManagerPanel, toolLabel, 0, 0, 155, 65);
    //circuitLabel=new JLabel("[Circuit Elements]");
    //addToPanel(toolsPanel,circuitLabel,40,30,155,30);
    //toolsBar=new JToolBar();
    //addToPanel(toolsPanel,toolsBar,40,70,150,200);

    //toolsBar.add(passiveBtn);
    img = img = ActiveState.btnsExited[ActiveState.OPENSCHEMATIC];
    openSchematicBtn = new JButton(new ImageIcon(img));
    openSchematicBtn.setActionCommand("openSchematicBtn");
    openSchematicBtn.setBackground(new Color(179, 186, 241));
    openSchematicBtn.setFont(myFont);
    openSchematicBtn.setBorder(null); ;
    openSchematicBtn.setForeground(Color.WHITE);
    addToPanel(fileManagerPanel, openSchematicBtn, 5, 110, 140, 30);

    img = img = ActiveState.btnsExited[ActiveState.SAVESCHEMATIC];
```

```
    saveSchematicBtn = new JButton(new ImageIcon(img));
    saveSchematicBtn.setActionCommand("saveSchematicBtn");
    saveSchematicBtn.setBackground(new Color(179, 186, 241));
    saveSchematicBtn.setFont(myFont);
    saveSchematicBtn.setBorder(null);
    saveSchematicBtn.setForeground(Color.WHITE);
    addToPanel(fileManagerPanel, saveSchematicBtn, 5, 160, 140, 30);

    img = img = ActiveState.btnsExited[ActiveState.NEWSCHEMATIC];
    newSchematicBtn = new JButton(new ImageIcon(img));
    newSchematicBtn.setActionCommand("newSchematicBtn");
    newSchematicBtn.setBackground(new Color(179, 186, 241));
    newSchematicBtn.setFont(myFont);
    newSchematicBtn.setBorder(null);
    newSchematicBtn.setForeground(Color.WHITE);
    addToPanel(fileManagerPanel, newSchematicBtn, 5, 60, 140, 30);

    img = img = ActiveState.btnsExited[ActiveState.DELETESCHEMATIC];
    deleteSchematicBtn = new JButton(new ImageIcon(img));
    deleteSchematicBtn.setActionCommand("deleteSchematicBtn");
    deleteSchematicBtn.setBackground(new Color(179, 186, 241));
    deleteSchematicBtn.setFont(myFont);
    deleteSchematicBtn.setBorder(null);
    deleteSchematicBtn.setForeground(Color.WHITE);
    addToPanel(fileManagerPanel, deleteSchematicBtn, 5, 210, 140, 30);

    img = ActiveState.btnsExited[ActiveState.FILEMANAGER];
    fileManagerBtn = new JButton(new ImageIcon(img));
    fileManagerBtn.setActionCommand("fileManagerBtn");
    fileManagerBtn.setBackground(new Color(179, 186, 241));
    fileManagerBtn.setBorder(null);
    fileManagerBtn.setForeground(Color.WHITE);
    fileManagerBtn.setFont(myFont);
    addToPanel(fileManagerPanel, fileManagerBtn, 5, 260, 140, 30);

  }

////////////////////////////////////////////////////////////////////////////

  private void drawSimulationPanel() {
    simulationPanel = new JPanel();
    simulationPanel.setBounds(765, 590, 150, 212);
    simulationPanel.setBackground(new Color(117, 159, 255));
    simulationPanel.setLayout(null);
    simulationPanel.setBorder(BorderFactory.createEtchedBorder
```

```
                    (Color.BLUE, Color.WHITE));
  this.getContentPane().add(simulationPanel);

  Image img = getImage(getDocumentBase(), "ECS/images/Simulation Tools.jpg");
  simulationLabel = new JLabel(new ImageIcon(img));
  addToPanel(simulationPanel, simulationLabel, 0, 0, 155, 65);

  img = ActiveState.btnsExited[ActiveState.CREATENETLIST];
  createNetlistBtn = new JButton(new ImageIcon(img));
  createNetlistBtn.setActionCommand("createNetlistBtn");
  createNetlistBtn.setBackground(new Color(179, 186, 241));
  createNetlistBtn.setBorder(null);
  createNetlistBtn.setForeground(Color.WHITE);
  createNetlistBtn.setFont(myFont);
  addToPanel(simulationPanel, createNetlistBtn, 5, 60, 140, 30);

  img = ActiveState.btnsExited[ActiveState.SIMSETTINGS];
  simSettingsBtn = new JButton(new ImageIcon(img));
  simSettingsBtn.setActionCommand("simSettingsBtn");
  simSettingsBtn.setBackground(new Color(179, 186, 241));
  simSettingsBtn.setBorder(null);
  simSettingsBtn.setFont(myFont);
  simSettingsBtn.setForeground(Color.WHITE);
  addToPanel(simulationPanel, simSettingsBtn, 5, 100, 140, 30);

  img = ActiveState.btnsExited[ActiveState.SIMULATE];
  simulateBtn = new JButton(new ImageIcon(img));
  simulateBtn.setActionCommand("simulateBtn");
  simulateBtn.setBackground(new Color(179, 186, 241));
  simulateBtn.setBorder(null);
  simulateBtn.setFont(myFont);
  simulateBtn.setForeground(Color.WHITE);
  addToPanel(simulationPanel, simulateBtn, 5, 140, 140, 30);

  img = ActiveState.btnsExited[ActiveState.NETLISTEDITING];
  netlistEditingBtn = new JButton(new ImageIcon(img));
  netlistEditingBtn.setActionCommand("netlistEditingBtn");
  netlistEditingBtn.setBackground(new Color(179, 186, 241));
  netlistEditingBtn.setBorder(null);
  netlistEditingBtn.setFont(myFont);
  netlistEditingBtn.setForeground(Color.WHITE);
  //addToPanel(simulationPanel, netlistEditingBtn, 5, 60, 140, 30);

}
```

```
////////////////////////////////////////////////////////////////////////////
  private void drawGridArea() {
    //drawAreaPanel=new JPanel();
    //drawAreaPanel.setLayout(null);
    //drawAreaPanel.setBorder(BorderFactory.createLoweredBevelBorder());
    //add(drawAreauePanel,30,40,600,500);
    mainScroll = new JScrollPane();
    mainScroll.addMouseListener(mainScrollMouseListener);
    mainScroll.addMouseMotionListener(mainScrollMouseMotionListener);

    mainScroll.setBounds(new java.awt.Rectangle(33, 35, 600, 500));
    gridCanvas = new GridCanvas(2500, 2500, canvasBackgroundColor,
                          canvasForegroundColor);
    gridCanvas.setLayout(null);
    //gridCanvas.setPreferredSize(new Dimension(1800, 1800));
    this.getContentPane().add(mainScroll);
    //startPoint[0]=0;startPoint[1]=0;
    //gridCanvas.setBounds(startPoint[0],startPoint[1],2500,2500);
    mainScroll.getViewport().add(gridCanvas, null);
    mainScroll.getHorizontalScrollBar().setBackground(new Color(201, 212, 255));
    mainScroll.getVerticalScrollBar().setBackground(new Color(201, 212, 255));
    mainScroll.setBackground(new Color(201, 212, 255));

  }

////////////////////////////////////////////////////////////////////////////
  private void drawNetListPanel() {
    netlistPanel = new NetlistOutputPanel();

//  String netListString = "*Schematic Netlist*\n";
    //netList.netText.setText(netListString);
    this.getContentPane().add(netlistPanel);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawGraphicalPanel() {

    graphicalResult = new GraphicalResultPanel();
    this.getContentPane().add(graphicalResult);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawPassiveElements() {
```

```
    passivePanel = new JPanel();
    passivePanel.setLayout(null);
    passivePanel.setBackground(new Color(71, 127, 255)); //(121, 133, 220));
    addToPanel(elementsPanel, passivePanel, 10, 10, 35, 130);
    passivePanel.setVisible(false);
    Image img = this.getImage(getDocumentBase(), "ECS/images/close.gif");

    JButton close = new JButton(new ImageIcon(img));
    close.setBorder(null);
    close.setBounds(20, 0, 12, 12);
    close.setActionCommand("passive");
    close.addMouseListener(new MouseAdapter() {
      public void mouseReleased(MouseEvent me) {
        btn_closed_action(me);
      }
    }
    );
    passivePanel.add(close);
    addImageButton(passivePanel, resistorBtn, ActiveState.RESISTOR, 5, 15, 25,
                25);

    addImageButton(passivePanel, capacitorBtn, ActiveState.CAPACITOR, 5, 43, 25,
                25);

    addImageButton(passivePanel, inductorBtn, ActiveState.INDUCTOR, 5, 72, 25,
                25);

    addImageButton(passivePanel, transformerBtn, ActiveState.TRANSFORMER, 5,
                100, 25, 25);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawIndependentSources() {

    independentPanel = new JPanel();
    independentPanel.setLayout(null);
    independentPanel.setBackground(new Color(71, 127, 255));
    addToPanel(elementsPanel, independentPanel, 10, 10, 35, 130);
    independentPanel.setVisible(false);

    Image img = this.getImage(getDocumentBase(), "ECS/images/close.gif");
    JButton close = new JButton(new ImageIcon(img));
    close.setBorder(null);
```

```
    close.setBounds(20, 0, 12, 12);
    close.setActionCommand("independent");
    close.addMouseListener(new MouseAdapter() {
      public void mouseReleased(MouseEvent me) {
        btn_closed_action(me);
      }
    }
    );

    independentPanel.add(close);

    addImageButton(independentPanel, dcVSBtn, ActiveState.DCVSOURCE, 5, 15, 25,
                25);

    addImageButton(independentPanel, dcCSBtn, ActiveState.DCCSOURCE, 5, 43, 25,
                25);

    addImageButton(independentPanel, acVSBtn, ActiveState.ACVSOURCE, 5, 72, 25,
                25);

    addImageButton(independentPanel, acCSBtn, ActiveState.ACCSOURCE, 5,
                100, 25, 25);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawDependentSources() {

    dependentPanel = new JPanel();
    dependentPanel.setLayout(null);
    dependentPanel.setBackground(new Color(71, 127, 255));
    addToPanel(elementsPanel, dependentPanel, 10, 10, 35, 130);
    dependentPanel.setVisible(false);

    Image img = this.getImage(getDocumentBase(), "ECS/images/close.gif");
    JButton close = new JButton(new ImageIcon(img));
    close.setBorder(null);
    close.setBounds(20, 0, 12, 12);
    close.setActionCommand("dependent");
    close.addMouseListener(new MouseAdapter() {
      public void mouseReleased(MouseEvent me) {
        btn_closed_action(me);
      }
    }
    );
```

```
    dependentPanel.add(close);

    addImageButton(dependentPanel, VCVSBtn, ActiveState.DEPVCVS, 5, 15, 25,
            25);
    addImageButton(dependentPanel, VCCSBtn, ActiveState.DEPVCCS, 5, 43, 25,
            25);

    addImageButton(dependentPanel, CCVSBtn, ActiveState.DEPCCVS, 5, 72, 25,
            25);

    addImageButton(dependentPanel, CCCSBtn, ActiveState.DEPCCCS, 5,
            100, 25, 25);
  }

////////////////////////////////////////////////////////////////////////////

  private void drawConnections() {

    connectionsPanel = new JPanel();
    connectionsPanel.setLayout(null);
    connectionsPanel.setBackground(new Color(71, 127, 255));
    addToPanel(elementsPanel, connectionsPanel, 10, 10, 35, 156);
    connectionsPanel.setVisible(false);

    Image img = this.getImage(getDocumentBase(), "ECS/images/close.gif");
    JButton close = new JButton(new ImageIcon(img));
    close.setBorder(null);
    close.setBounds(20, 0, 12, 12);
    close.setActionCommand("connections");
    close.addMouseListener(new MouseAdapter() {
      public void mouseReleased(MouseEvent me) {
        btn_closed_action(me);
      }
    }
    );

    connectionsPanel.add(close);

    addImageButton(connectionsPanel, connectionBtn, ActiveState.CONNECTOR, 5,
            15, 25,
            25);

    //addImageButton(connectionsPanel, nodeBtn, ActiveState.CONNODE, 5, 43, 25,
    //        25);
```

```
    addImageButton(connectionsPanel, groundBtn, ActiveState.GROUND, 5, 43, 25,
            25);

    addImageButton(connectionsPanel, voltageMBtn, ActiveState.VOLTAGEMARK, 5,
            72, 25,
            25);

    addImageButton(connectionsPanel, diffVoltageMBtn, ActiveState.NODEL, 5,
            100, 25, 25);

    addImageButton(connectionsPanel, currentMBtn, ActiveState.CURRENTMARK, 5,
            128, 25, 25);

  }

////////////////////////////////////////////////////////////////////////////
  public void drawTransformers() {
    transformersPanel = new JPanel();
    transformersPanel.setLayout(null);
    transformersPanel.setBackground(new Color(71, 127, 255));
    addToPanel(elementsPanel, transformersPanel, 10, 245, 35, 130);
    transformersPanel.setVisible(false);

    Image img = this.getImage(getDocumentBase(), "ECS/images/close.gif");
    JButton close = new JButton(new ImageIcon(img));
    close.setBorder(null);
    close.setBounds(20, 0, 12, 12);
    close.setActionCommand("transformers");
    close.addMouseListener(new MouseAdapter() {
      public void mouseReleased(MouseEvent me) {
        btn_closed_action(me);
      }
    }
    );

    transformersPanel.add(close);

    addImageButton(transformersPanel, transformerABtn, ActiveState.TRANSFORMERA,
            5, 15, 25, 25);

    addImageButton(transformersPanel, transformerBBtn, ActiveState.TRANSFORMERB,
            5, 43, 25, 25);

    addImageButton(transformersPanel, transformerCBtn, ActiveState.TRANSFORMERC,
```

```
            5, 72, 25, 25);

    addImageButton(transformersPanel, transformerDBtn, ActiveState.TRANSFORMERD,
            5, 100, 25, 25);
  }

////////////////////////////////////////////////////////////////////////////

  private void buildPopupMenus() {

    Image img = this.getImage(getDocumentBase(), "ECS/images/close.gif");
    menuItem10 = new JMenuItem("\u0425" + "   Close");

    img = ActiveState.btnsExited[ActiveState.MENUROTATERIGHT];
    singleElementRotateRight = new JMenuItem("Rotate Right", new ImageIcon(img));
    singleElementRotateRight.setActionCommand("singleElementRotateRight");

    img = ActiveState.btnsExited[ActiveState.MENUROTATELEFT];
    singleElementRotateLeft = new JMenuItem("Rotate Left", new ImageIcon(img));
    singleElementRotateLeft.setActionCommand("singleElementRotateLeft");

    img = ActiveState.btnsExited[ActiveState.MENUFLIPVERTICAL];
    singleElementFlipVertical = new JMenuItem("Flip Vertical",
                                new ImageIcon(img));
    singleElementFlipVertical.setActionCommand("singleElementFlipVertical");

    img = ActiveState.btnsExited[ActiveState.MENUFLIPHORIZONTAL];
    singleElementFlipHorizontal = new JMenuItem("Flip Horizontal",
                                  new ImageIcon(img));
    singleElementFlipHorizontal.setActionCommand("singleElementFlipHorizontal");

    img = ActiveState.btnsExited[ActiveState.MENUDISPLAYPROPERTIES];
    singleElementDisplayProperties = new JMenuItem("Display Properties",
       new ImageIcon(img));
    singleElementDisplayProperties.setActionCommand(
       "singleElementDisplayProperties");

    img = ActiveState.btnsExited[ActiveState.MENUZOOMIN];
    singleElementZoomIn = new JMenuItem("Zoom In", new ImageIcon(img));
    singleElementZoomIn.setActionCommand("singleElementZoomIn");

    img = ActiveState.btnsExited[ActiveState.MENUZOOMOUT];
    singleElementZoomOut = new JMenuItem("Zoom Out", new ImageIcon(img));
    singleElementZoomOut.setActionCommand("singleElementZoomOut");
```

```
img = ActiveState.btnsExited[ActiveState.MENUGOTO];
singleElementGoTo = new JMenuItem("Go To...", new ImageIcon(img));
singleElementGoTo.setActionCommand("singleElementGoTo");

img = ActiveState.btnsExited[ActiveState.MENUCUT];
singleElementCut = new JMenuItem("Cut", new ImageIcon(img));
singleElementCut.setActionCommand("singleElementCut");

img = ActiveState.btnsExited[ActiveState.MENUCOPY]; ;
singleElementCopy = new JMenuItem("Copy", new ImageIcon(img));
singleElementCopy.setActionCommand("singleElementCopy");

img = ActiveState.btnsExited[ActiveState.MENUDELETE]; ;
singleElementDelete = new JMenuItem("Delete", new ImageIcon(img));
singleElementDelete.setActionCommand("singleElementDelete");

img = ActiveState.btnsExited[ActiveState.MENUEDITPROPERTIES];
enterValue = new JMenuItem(" Edit Attributes", new ImageIcon(img));
enterValue.setActionCommand("enterValue");

img = ActiveState.btnsExited[ActiveState.MENUDISPLAYPROPERTIES];
multiElementsDisplayProperties = new JMenuItem("Display Properties",
   new ImageIcon(img));
multiElementsDisplayProperties.setActionCommand(
   "multiElementsDisplayProperties");

img = ActiveState.btnsExited[ActiveState.MENUROTATERIGHT];
multiElementsRotateRight = new JMenuItem("Rotate Right", new ImageIcon(img));
multiElementsRotateRight.setActionCommand("multiElementsRotateRight");

img = ActiveState.btnsExited[ActiveState.MENUROTATELEFT];
multiElementsRotateLeft = new JMenuItem("Rotate Left", new ImageIcon(img));
multiElementsRotateLeft.setActionCommand("multiElementsRotateLeft");

img = ActiveState.btnsExited[ActiveState.MENUFLIPVERTICAL];
multiElementsFlipVertical = new JMenuItem("Flip Vertical",
                                new ImageIcon(img));

img = ActiveState.btnsExited[ActiveState.MENUFLIPHORIZONTAL];
multiElementsFlipHorizontal = new JMenuItem("Flip Horizontal",
                                  new ImageIcon(img));

img = ActiveState.btnsExited[ActiveState.MENUCUT];
multiElementsCut = new JMenuItem("Cut", new ImageIcon(img));
multiElementsCut.setActionCommand("multiElementsCut");
```

```
img = ActiveState.btnsExited[ActiveState.MENUCOPY];
multiElementsCopy = new JMenuItem("Copy", new ImageIcon(img));
multiElementsCopy.setActionCommand("multiElementsCopy");

img = ActiveState.btnsExited[ActiveState.MENUDELETE];
multiElementsDelete = new JMenuItem("Delete", new ImageIcon(img));
multiElementsDelete.setActionCommand("multiElementsDelete");

img = ActiveState.btnsExited[ActiveState.MENUGROUP];
multiElementsGroup = new JMenuItem("Group", new ImageIcon(img));
multiElementsGroup.setActionCommand("multiElementsGroup");

img = ActiveState.btnsExited[ActiveState.MENUUNGROUP];
multiElementsUngroup = new JMenuItem("Ungroup", new ImageIcon(img));
multiElementsUngroup.setActionCommand("multiElementsUngroup");

img = ActiveState.btnsExited[ActiveState.MENUREGROUP];
multiElementsRegroup = new JMenuItem("Regroup", new ImageIcon(img));
multiElementsRegroup.setActionCommand("multiElementsRegroup");

img = ActiveState.btnsExited[ActiveState.MENUALIGNTOP];
multiElementsAlignTop = new JMenuItem("Align Top", new ImageIcon(img));
multiElementsAlignTop.setActionCommand("multiElementsAlignTop");

img = ActiveState.btnsExited[ActiveState.MENUALIGNLEFT];
multiElementsAlignLeft = new JMenuItem("Align Left", new ImageIcon(img));
multiElementsAlignLeft.setActionCommand("multiElementsAlignLeft");

img = ActiveState.btnsExited[ActiveState.MENUALIGNRIGHT];
multiElementsAlignRight = new JMenuItem("Align Right", new ImageIcon(img));
multiElementsAlignRight.setActionCommand("multiElementsAlignRight");

img = ActiveState.btnsExited[ActiveState.MENUALIGNBOTTOM];
multiElementsAlignBottom = new JMenuItem("Align Bottom", new ImageIcon(img));
multiElementsAlignBottom.setActionCommand("multiElementsAlignBottom");

img = ActiveState.btnsExited[ActiveState.MENUZOOMIN];
multiElementsZoomIn = new JMenuItem("Zoom In", new ImageIcon(img));
multiElementsZoomIn.setActionCommand("multiElementsZoomIn");

img = ActiveState.btnsExited[ActiveState.MENUZOOMOUT];
multiElementsZoomOut = new JMenuItem("Zoom Out", new ImageIcon(img));
multiElementsZoomOut.setActionCommand("multiElementsZoomOut");
```

```
img = ActiveState.btnsExited[ActiveState.MENUGOTO];
multiElementsGoTo = new JMenuItem("Go To...", new ImageIcon(img));
multiElementsGoTo.setActionCommand("multiElementsGoTo");

img = ActiveState.btnsExited[ActiveState.MENUDELETE];
clearAll = new JMenuItem("Clear All", new ImageIcon(img));
clearAll.setActionCommand("clearAll");

img = ActiveState.btnsExited[ActiveState.MENUUNDO];
undo = new JMenuItem("Undo", new ImageIcon(img));
undo.setActionCommand("undo");

img = ActiveState.btnsExited[ActiveState.MENUPASTE];
paste = new JMenuItem("Paste", new ImageIcon(img));
paste.setActionCommand("paste");

img = ActiveState.btnsExited[ActiveState.MENUZOOMIN];
zoomIn = new JMenuItem("Zoom In", new ImageIcon(img));
zoomIn.setActionCommand("zoomIn");

img = ActiveState.btnsExited[ActiveState.MENUZOOMOUT];
zoomOut = new JMenuItem("Zoom Out", new ImageIcon(img));
zoomOut.setActionCommand("zoomOut");

img = ActiveState.btnsExited[ActiveState.MENUGOTO];
goToLine = new JMenuItem("Go To...", new ImageIcon(img));
goToLine.setActionCommand("goToLine");

goToPage = new JMenuItem("Go to Page...");
goToPage.setActionCommand("goToPage");

img = ActiveState.btnsExited[ActiveState.MENUCUT];
cutLine = new JMenuItem("Cut", new ImageIcon(img));
cutLine.setActionCommand("cutLine");

img = ActiveState.btnsExited[ActiveState.MENUCOPY];
copyLine = new JMenuItem("Copy", new ImageIcon(img));
copyLine.setActionCommand("copyLine");

img = ActiveState.btnsExited[ActiveState.MENUDELETE];
deleteLine = new JMenuItem("Delete", new ImageIcon(img));
deleteLine.setActionCommand("deleteLine");

singleElementShapePopMenu.setEnabled(true);
singleElementShapePopMenu.setBackground(new Color(214, 223, 247));
```

```
singleElementShapePopMenu.add(singleElementRotateRight);
singleElementShapePopMenu.add(singleElementRotateLeft);
singleElementShapePopMenu.add(singleElementFlipVertical);
singleElementShapePopMenu.add(singleElementFlipHorizontal);

singleElementViewPopMenu.setEnabled(true);
singleElementViewPopMenu.setBackground(new Color(214, 223, 247));
singleElementViewPopMenu.add(singleElementDisplayProperties);
//singleElementViewPopMenu.add(singleElementZoomIn);
//singleElementViewPopMenu.add(singleElementZoomOut);
singleElementViewPopMenu.add(singleElementGoTo);

multiElementsShapePopMenu.setEnabled(true);
multiElementsShapePopMenu.setBackground(new Color(214, 223, 247));
multiElementsShapePopMenu.add(multiElementsRotateRight);
multiElementsShapePopMenu.add(multiElementsRotateLeft);
multiElementsShapePopMenu.add(multiElementsFlipVertical);
multiElementsShapePopMenu.add(multiElementsFlipHorizontal);
multiElementsShapePopMenu.addSeparator();
multiElementsShapePopMenu.add(multiElementsGroup);
multiElementsShapePopMenu.add(multiElementsUngroup);
multiElementsShapePopMenu.add(multiElementsRegroup);
multiElementsShapePopMenu.addSeparator();
multiElementsShapePopMenu.add(multiElementsAlignTop);
multiElementsShapePopMenu.add(multiElementsAlignLeft);
multiElementsShapePopMenu.add(multiElementsAlignRight);
multiElementsShapePopMenu.add(multiElementsAlignBottom);

multiElementsViewPopMenu.setEnabled(true);
multiElementsViewPopMenu.setBackground(new Color(214, 223, 247));
multiElementsViewPopMenu.add(multiElementsDisplayProperties);
//multiElementsViewPopMenu.add(multiElementsZoomIn);
//multiElementsViewPopMenu.add(multiElementsZoomOut);
multiElementsViewPopMenu.add(multiElementsGoTo);

singleElementPopmenu.setEnabled(true);

singleElementPopmenu.add(singleElementViewPopMenu);
singleElementPopmenu.addSeparator();
singleElementPopmenu.add(singleElementShapePopMenu);
singleElementPopmenu.addSeparator();
singleElementPopmenu.add(enterValue);
singleElementPopmenu.addSeparator();
singleElementPopmenu.add(singleElementCut);
singleElementPopmenu.add(singleElementCopy);
```

```
singleElementPopmenu.add(singleElementDelete);
singleElementPopmenu.setBackground(Color.WHITE);
singleElementPopmenu.setForeground(Color.WHITE);

multiElementsPopmenu.setEnabled(true);
multiElementsPopmenu.add(multiElementsViewPopMenu);
multiElementsPopmenu.addSeparator();
multiElementsPopmenu.add(multiElementsShapePopMenu);
multiElementsPopmenu.addSeparator();
multiElementsPopmenu.add(multiElementsCopy);
multiElementsPopmenu.add(multiElementsCut);
multiElementsPopmenu.add(multiElementsDelete);
multiElementsPopmenu.setBackground(Color.WHITE);
multiElementsPopmenu.setForeground(Color.WHITE);

gridPopupMenu.setEnabled(true);

gridPopupMenu.add(clearAll);
//gridPopupMenu.add(zoomIn);
//gridPopupMenu.add(zoomOut);
gridPopupMenu.add(goToLine);

gridPopupMenu.add(undo);
gridPopupMenu.add(paste);
gridPopupMenu.setBackground(Color.WHITE);
gridPopupMenu.setForeground(Color.WHITE);

connectionLinePopupMenu.setEnabled(true);
connectionLinePopupMenu.add(cutLine);
connectionLinePopupMenu.add(copyLine);
connectionLinePopupMenu.add(deleteLine);
connectionLinePopupMenu.setBackground(Color.WHITE);
connectionLinePopupMenu.setForeground(Color.WHITE);

closeMenu.setEnabled(true);
closeMenu.add(menuItem10);
closeMenu.setBackground(Color.WHITE);
closeMenu.setForeground(Color.WHITE);

enterValue.setBackground(new Color(214, 223, 247));
singleElementRotateRight.setBackground(new Color(214, 223, 247));
singleElementRotateLeft.setBackground(new Color(214, 223, 247));
singleElementCut.setBackground(new Color(214, 223, 247));
singleElementCopy.setBackground(new Color(214, 223, 247));
singleElementDelete.setBackground(new Color(214, 223, 247));
```

```
multiElementsRotateLeft.setBackground(new Color(214, 223, 247));
multiElementsRotateRight.setBackground(new Color(214, 223, 247));
multiElementsCut.setBackground(new Color(214, 223, 247));
multiElementsCopy.setBackground(new Color(214, 223, 247));
multiElementsDelete.setBackground(new Color(214, 223, 247));
multiElementsGroup.setBackground(new Color(214, 223, 247));
multiElementsUngroup.setBackground(new Color(214, 223, 247));
paste.setBackground(new Color(214, 223, 247));
zoomIn.setBackground(new Color(214, 223, 247));
zoomOut.setBackground(new Color(214, 223, 247));
deleteLine.setBackground(new Color(214, 223, 247));
copyLine.setBackground(new Color(214, 223, 247));
cutLine.setBackground(new Color(214, 223, 247));
undo.setBackground(new Color(214, 223, 247));
menuItem10.setBackground(new Color(214, 223, 247));
clearAll.setBackground(new Color(214, 223, 247));
goToLine.setBackground(new Color(214, 223, 247));
goToPage.setBackground(new Color(214, 223, 247));
multiElementsRegroup.setBackground(new Color(214, 223, 247));
singleElementDisplayProperties.setBackground(new Color(214, 223, 247));
singleElementFlipVertical.setBackground(new Color(214, 223, 247));
singleElementFlipHorizontal.setBackground(new Color(214, 223, 247));
singleElementZoomIn.setBackground(new Color(214, 223, 247));
singleElementZoomOut.setBackground(new Color(214, 223, 247));
singleElementGoTo.setBackground(new Color(214, 223, 247));
multiElementsDisplayProperties.setBackground(new Color(214, 223, 247));
multiElementsFlipVertical.setBackground(new Color(214, 223, 247));
multiElementsFlipHorizontal.setBackground(new Color(214, 223, 247));
multiElementsGroup.setBackground(new Color(214, 223, 247));
multiElementsUngroup.setBackground(new Color(214, 223, 247));
multiElementsRegroup.setBackground(new Color(214, 223, 247));
multiElementsAlignRight.setBackground(new Color(214, 223, 247));
multiElementsAlignLeft.setBackground(new Color(214, 223, 247));
multiElementsAlignTop.setBackground(new Color(214, 223, 247));
multiElementsAlignBottom.setBackground(new Color(214, 223, 247));
multiElementsZoomIn.setBackground(new Color(214, 223, 247));
multiElementsZoomOut.setBackground(new Color(214, 223, 247));
multiElementsGoTo.setBackground(new Color(214, 223, 247));

enterValue.setForeground(Color.BLACK);
singleElementRotateRight.setForeground(Color.BLACK);
singleElementRotateLeft.setForeground(Color.BLACK);
singleElementCut.setForeground(Color.BLACK);
singleElementCopy.setForeground(Color.BLACK);
singleElementDelete.setForeground(Color.BLACK);
```

```
    multiElementsRotateLeft.setForeground(Color.BLACK);
    multiElementsRotateRight.setForeground(Color.BLACK);
    multiElementsCut.setForeground(Color.BLACK);
    multiElementsCopy.setForeground(Color.BLACK);
    multiElementsDelete.setForeground(Color.BLACK);
    multiElementsGroup.setForeground(Color.BLACK);
    multiElementsUngroup.setForeground(Color.BLACK);
    paste.setForeground(Color.BLACK);
    zoomIn.setForeground(Color.BLACK);
    zoomOut.setForeground(Color.BLACK);
    deleteLine.setForeground(Color.BLACK);
    undo.setForeground(Color.BLACK);
    menuItem10.setForeground(Color.BLACK);
    clearAll.setForeground(Color.BLACK);

    gridCanvas.add(singleElementPopmenu);
    gridCanvas.add(multiElementsPopmenu);
    gridCanvas.add(gridPopupMenu);
    gridCanvas.add(connectionLinePopupMenu);
    elementsPanel.add(closeMenu);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawMenu() {
    Frame fr;
    Object ob = getParent();
    while (! (ob instanceof Frame)) {
      ob = ( (Component) ob).getParent();
    }
    fr = (Frame) ob;
    MenuBar mb = new MenuBar();
    fr.setMenuBar(mb);
    Menu menu = new Menu("File");
    MenuItem open = new MenuItem("Open");
    menu.add(open);
    mb.add(menu);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawStandrardToolbar() {
    standardToolbar = new JToolBar();
    standardToolbar.setFloatable(false);
    standardToolbar.setBackground(new Color(178, 197, 251));
```

```
//Image img = getImage(getDocumentBase(), "ECS/images/new.jpg");
Image img = ActiveState.btnsExited[ActiveState.NEW];
newBtn = new JButton(new ImageIcon(img));
newBtn.setActionCommand("newBtn");
newBtn.setBackground(new Color(55, 115, 247));
newBtn.setBorder(null);
//loadBtn.setPreferredSize(new Dimension(5,5));
// loadBtn.setBackground(new Color(129, 133, 254));
newBtn.setToolTipText("New Page");
//loadBtn.setForeground(Color.WHITE);
//standardToolbar.setRollover(true);

//upperToolbar.addSeparator();
img = ActiveState.btnsExited[ActiveState.OPEN];
loadBtn = new JButton(new ImageIcon(img));
loadBtn.setBackground(new Color(55, 115, 247));
loadBtn.setBorder(null);
loadBtn.setActionCommand("loadBtn");
//loadBtn.setPreferredSize(new Dimension(5,5));
// loadBtn.setBackground(new Color(129, 133, 254));
loadBtn.setToolTipText("Open Page");
//loadBtn.setForeground(Color.WHITE);
//standardToolbar.setRollover(true);

img = ActiveState.btnsExited[ActiveState.CUT];
cutBtn = new JButton(new ImageIcon(img));
cutBtn.setActionCommand("cutBtn");
cutBtn.setBackground(new Color(55, 115, 247));
cutBtn.setBorder(null);
//loadBtn.setPreferredSize(new Dimension(5,5));
// loadBtn.setBackground(new Color(129, 133, 254));
cutBtn.setToolTipText("Cut");
//loadBtn.setForeground(Color.WHITE);
//standardToolbar.setRollover(true);

//upperToolbar.addSeparator();
img = ActiveState.btnsExited[ActiveState.COPY];
copyBtn = new JButton(new ImageIcon(img));
copyBtn.setActionCommand("copyBtn");
copyBtn.setBackground(new Color(55, 115, 247));
copyBtn.setBorder(null);
//loadBtn.setPreferredSize(new Dimension(5,5));
// loadBtn.setBackground(new Color(129, 133, 254));
copyBtn.setToolTipText("Copy");
```

```
    //loadBtn.setForeground(Color.WHITE);
    //standardToolbar.setRollover(true);

    //upperToolbar.addSeparator();
    img = ActiveState.btnsExited[ActiveState.PASTE];
    pasteBtn = new JButton(new ImageIcon(img));
    pasteBtn.setActionCommand("pasteBtn");
    pasteBtn.setBackground(new Color(55, 115, 247));
    pasteBtn.setBorder(null);
    //loadBtn.setPreferredSize(new Dimension(5,5));
    // loadBtn.setBackground(new Color(129, 133, 254));
    pasteBtn.setToolTipText("Paste");
    //loadBtn.setForeground(Color.WHITE);
    //standardToolbar.setRollover(true);

    //upperToolbar.addSeparator();
    img = ActiveState.btnsExited[ActiveState.UNDO];
    undoBtn = new JButton(new ImageIcon(img));
    undoBtn.setActionCommand("undoBtn");
    undoBtn.setBackground(new Color(55, 115, 247));
    undoBtn.setBorder(null);
    //loadBtn.setPreferredSize(new Dimension(5,5));
    // loadBtn.setBackground(new Color(129, 133, 254));
    undoBtn.setToolTipText("Undo");
    //loadBtn.setForeground(Color.WHITE);
    //standardToolbar.setRollover(true);

    //upperToolbar.addSeparator();
    img = ActiveState.btnsExited[ActiveState.REDO];
    redoBtn = new JButton(new ImageIcon(img));
    redoBtn.setActionCommand("redoBtn");
    redoBtn.setBackground(new Color(55, 115, 247));
    redoBtn.setBorder(null);
//loadBtn.setPreferredSize(new Dimension(5,5));
// loadBtn.setBackground(new Color(129, 133, 254));
    redoBtn.setToolTipText("Redo");
//loadBtn.setForeground(Color.WHITE);
//standardToolbar.setRollover(true);

    img = ActiveState.btnsExited[ActiveState.NEWSCHEMATIC];
    clearBtn = new JButton(new ImageIcon(img));
    clearBtn.setActionCommand("clearBtn");
    clearBtn.setBackground(new Color(55, 115, 247));
    clearBtn.setToolTipText("Clear All");
    clearBtn.setBackground(new Color(179, 186, 241));
```

```
clearBtn.setBorder(null);
//clearBtn.setForeground(Color.WHITE);
// clearBtn.setPreferredSize(new Dimension(30,30));

img = ActiveState.btnsExited[ActiveState.SAVE];
saveBtn = new JButton(new ImageIcon(img));
saveBtn.setActionCommand("saveBtn");
saveBtn.setBackground(new Color(55, 115, 247));
saveBtn.setToolTipText("Save Page");
saveBtn.setBackground(new Color(179, 186, 241));
saveBtn.setBorder(null);

//saveBtn.setForeground(Color.WHITE);

img = ActiveState.btnsExited[ActiveState.HELP];
helpBtn = new JButton(new ImageIcon(img));
helpBtn.setActionCommand("helpBtn");
helpBtn.setBackground(new Color(55, 115, 247));
helpBtn.setToolTipText("Help Topics");
helpBtn.setBackground(new Color(179, 186, 241));
helpBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.RUN];
runBtn = new JButton(new ImageIcon(img));
runBtn.setActionCommand("runBtn");
runBtn.setBackground(new Color(55, 115, 247));
runBtn.setToolTipText("Run");
runBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.DELETEPAGE];
deletePageBtn = new JButton(new ImageIcon(img));
deletePageBtn.setActionCommand("deletePageBtn");
deletePageBtn.setBackground(new Color(55, 115, 247));
deletePageBtn.setToolTipText("Delete Current Page");
deletePageBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.SERVERFILEMANAGER];
serverFileManagerBtn = new JButton(new ImageIcon(img));
serverFileManagerBtn.setActionCommand("serverFileManagerBtn");
serverFileManagerBtn.setBackground(new Color(55, 115, 247));
serverFileManagerBtn.setToolTipText("Server File Manager");
serverFileManagerBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.REMOVE];
deleteBtn = new JButton(new ImageIcon(img));
```

```
deleteBtn.setActionCommand("deleteBtn");
deleteBtn.setBackground(new Color(55, 115, 247));
deleteBtn.setToolTipText("Delete");
deleteBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.SELECTALLCOMPS];
selectAllCompsBtn = new JButton(new ImageIcon(img));
selectAllCompsBtn.setActionCommand("selectAllCompsBtn");
selectAllCompsBtn.setBackground(new Color(55, 115, 247));
selectAllCompsBtn.setToolTipText("Select All");
selectAllCompsBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.FORWARD];
forwardBtn = new JButton(new ImageIcon(img));
forwardBtn.setActionCommand("forwardBtn");
forwardBtn.setBackground(new Color(55, 115, 247));
forwardBtn.setToolTipText("Next Page");
forwardBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.BACKWARD];
backwardBtn = new JButton(new ImageIcon(img));
backwardBtn.setActionCommand("backwardBtn");
backwardBtn.setBackground(new Color(55, 115, 247));
backwardBtn.setToolTipText("Previous Page");
backwardBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.REFRESH];
refreshBtn = new JButton(new ImageIcon(img));
refreshBtn.setActionCommand("refreshBtn");
refreshBtn.setBackground(new Color(55, 115, 247));
refreshBtn.setToolTipText("Refresh");
refreshBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.LASTBACKWARD];
lastBackwardBtn = new JButton(new ImageIcon(img));
lastBackwardBtn.setActionCommand("lastBackwardBtn");
lastBackwardBtn.setBackground(new Color(55, 115, 247));
lastBackwardBtn.setToolTipText("First Page");
lastBackwardBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.LASTFORWARD];
lastForwardBtn = new JButton(new ImageIcon(img));
lastForwardBtn.setActionCommand("lastForwardBtn");
lastForwardBtn.setBackground(new Color(55, 115, 247));
lastForwardBtn.setToolTipText("Last Page");
```

```
lastForwardBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.GRID];
gridBtn = new JButton(new ImageIcon(img));
gridBtn.setActionCommand("gridBtn");
gridBtn.setBackground(new Color(55, 115, 247));
gridBtn.setToolTipText("Show/Hide Grid");
gridBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.CLEARALLCOMPS];
clearAllCompsBtn = new JButton(new ImageIcon(img));
clearAllCompsBtn.setActionCommand("clearAllCompsBtn");
clearAllCompsBtn.setBackground(new Color(55, 115, 247));
clearAllCompsBtn.setToolTipText("Clear All");
clearAllCompsBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.CONTACTUS];
contactUsBtn = new JButton(new ImageIcon(img));
contactUsBtn.setActionCommand("contactUsBtn");
contactUsBtn.setBackground(new Color(55, 115, 247));
contactUsBtn.setToolTipText("Contact us");
contactUsBtn.setBorder(null);

img = ActiveState.diffImages[ActiveState.SEPARATOR];
JButton sep1 = new JButton(new ImageIcon(img));
JButton sep2 = new JButton(new ImageIcon(img));
JButton sep3 = new JButton(new ImageIcon(img));
JButton sep4 = new JButton(new ImageIcon(img));
JButton sep5 = new JButton(new ImageIcon(img));
JButton sep6 = new JButton(new ImageIcon(img));
JButton sep7 = new JButton(new ImageIcon(img));
JButton sep8 = new JButton(new ImageIcon(img));
JButton sep9 = new JButton(new ImageIcon(img));

sep1.setBackground(new Color(178, 197, 251));
sep2.setBackground(new Color(178, 197, 251));
sep3.setBackground(new Color(178, 197, 251));
sep4.setBackground(new Color(178, 197, 251));
sep5.setBackground(new Color(178, 197, 251));
sep6.setBackground(new Color(178, 197, 251));
sep7.setBackground(new Color(178, 197, 251));
sep8.setBackground(new Color(178, 197, 251));
sep9.setBackground(new Color(178, 197, 251));

sep1.setBorder(null);
```

```
sep2.setBorder(null);
sep3.setBorder(null);
sep4.setBorder(null);
sep5.setBorder(null);
sep6.setBorder(null);
sep7.setBorder(null);
sep8.setBorder(null);
sep9.setBorder(null);

standardToolbar.add(sep2);
standardToolbar.add(sep3);
standardToolbar.addSeparator();
standardToolbar.add(newBtn);
standardToolbar.addSeparator();
standardToolbar.add(loadBtn);
standardToolbar.addSeparator();
standardToolbar.add(saveBtn);
standardToolbar.addSeparator();
standardToolbar.add(deletePageBtn);
standardToolbar.addSeparator();
standardToolbar.add(serverFileManagerBtn);
standardToolbar.addSeparator();
standardToolbar.add(sep4);
standardToolbar.addSeparator();
standardToolbar.add(lastBackwardBtn);
standardToolbar.addSeparator();
standardToolbar.add(backwardBtn);
standardToolbar.addSeparator();
standardToolbar.add(forwardBtn);
standardToolbar.addSeparator();
standardToolbar.add(lastForwardBtn);
standardToolbar.addSeparator();
standardToolbar.add(refreshBtn);
standardToolbar.addSeparator();
standardToolbar.add(sep5);
standardToolbar.addSeparator();
standardToolbar.add(cutBtn);
standardToolbar.addSeparator();
standardToolbar.add(copyBtn);
standardToolbar.addSeparator();
standardToolbar.add(pasteBtn);
standardToolbar.addSeparator();
standardToolbar.add(deleteBtn);
standardToolbar.addSeparator();
standardToolbar.add(undoBtn);
```

```
    //standardToolbar.addSeparator();
    //standardToolbar.add(redoBtn);
    standardToolbar.addSeparator();
    standardToolbar.add(sep6);
    standardToolbar.addSeparator();
    standardToolbar.add(gridBtn);
    standardToolbar.addSeparator();
    standardToolbar.add(selectAllCompsBtn);
    standardToolbar.addSeparator();
    standardToolbar.add(clearAllCompsBtn);
    standardToolbar.addSeparator();
    standardToolbar.add(sep7);
    standardToolbar.addSeparator();

    standardToolbar.add(contactUsBtn);
    standardToolbar.addSeparator();
    standardToolbar.add(helpBtn);
    standardToolbar.addSeparator();

    standardToolbar.add(sep8);

    standardToolbar.setBounds(0, 5, 645, 25);
    //standardToolbar.setBounds(0, 5, 671, 25);
    this.getContentPane().add(standardToolbar);

//addToPanel(simulationPanel, upperToolbar, 0, 222, 147, 25);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawComponentToolbar() {

    componentToolbar = new JToolBar();
    componentToolbar.setFloatable(false);
    componentToolbar.setBackground(new Color(178, 197, 251));
    componentToolbar.setVisible(false);

    Image img = ActiveState.btnsExited[ActiveState.DISPLAYPROPERTIES];
    displayPropertiesBtn = new JButton(new ImageIcon(img));
    displayPropertiesBtn.setActionCommand("displayPropertiesBtn");
    displayPropertiesBtn.setBackground(new Color(55, 115, 247));
    displayPropertiesBtn.setToolTipText("Display Format");
    displayPropertiesBtn.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.TOOLBAREDITPROPERTIES];
```

```
editComponentBtn = new JButton(new ImageIcon(img));
editComponentBtn.setActionCommand("editComponentBtn");
editComponentBtn.setBackground(new Color(55, 115, 247));
editComponentBtn.setToolTipText("Edit Attributes");
editComponentBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ROTATERIGHT];
rotateRightBtn = new JButton(new ImageIcon(img));
rotateRightBtn.setActionCommand("rotateRightBtn");
rotateRightBtn.setBackground(new Color(55, 115, 247));
rotateRightBtn.setToolTipText("Rotate Right");
rotateRightBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ROTATELEFT];
rotateLeftBtn = new JButton(new ImageIcon(img));
rotateLeftBtn.setActionCommand("rotateLeftBtn");
rotateLeftBtn.setBackground(new Color(55, 115, 247));
rotateLeftBtn.setToolTipText("Rotate Left");
rotateLeftBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.FLIPHORIZANTAL];
flipHorizontalBtn = new JButton(new ImageIcon(img));
flipHorizontalBtn.setActionCommand("flipHorizontalBtn");
flipHorizontalBtn.setBackground(new Color(55, 115, 247));
flipHorizontalBtn.setToolTipText("flip Horizontal");
flipHorizontalBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.FLIPVERTICAL];
flipVerticalBtn = new JButton(new ImageIcon(img));
flipVerticalBtn.setActionCommand("flipVerticalBtn");
flipVerticalBtn.setBackground(new Color(55, 115, 247));
flipVerticalBtn.setToolTipText("Flip Vertical");
flipVerticalBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ROTATETEXT];
rotateTextBtn = new JButton(new ImageIcon(img));
rotateTextBtn.setActionCommand("rotateTextBtn");
rotateTextBtn.setBackground(new Color(55, 115, 247));
rotateTextBtn.setToolTipText("Rotate Text");
rotateTextBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.GROUP];
groupComponentsBtn = new JButton(new ImageIcon(img));
groupComponentsBtn.setActionCommand("groupComponentsBtn");
groupComponentsBtn.setBackground(new Color(55, 115, 247));
```

```
groupComponentsBtn.setToolTipText("Group Components");
groupComponentsBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.UNGROUP];
ungroupComponentsBtn = new JButton(new ImageIcon(img));
ungroupComponentsBtn.setActionCommand("ungroupComponentsBtn");
ungroupComponentsBtn.setBackground(new Color(55, 115, 247));
ungroupComponentsBtn.setToolTipText("Ungroup Components");
ungroupComponentsBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ALIGNRIGHT];
alignRightBtn = new JButton(new ImageIcon(img));
alignRightBtn.setActionCommand("alignRightBtn");
alignRightBtn.setBackground(new Color(55, 115, 247));
alignRightBtn.setToolTipText("Align Right");
alignRightBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ALIGNLEFT];
alignLeftBtn = new JButton(new ImageIcon(img));
alignLeftBtn.setActionCommand("alignLeftBtn");
alignLeftBtn.setBackground(new Color(55, 115, 247));
alignLeftBtn.setToolTipText("Align Left");
alignLeftBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ALIGNUP];
alignTopBtn = new JButton(new ImageIcon(img));
alignTopBtn.setActionCommand("alignTopBtn");
alignTopBtn.setBackground(new Color(55, 115, 247));
alignTopBtn.setToolTipText("Align Top");
alignTopBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ALIGNDOWN];
alignBottomBtn = new JButton(new ImageIcon(img));
alignBottomBtn.setActionCommand("alignBottomBtn");
alignBottomBtn.setBackground(new Color(55, 115, 247));
alignBottomBtn.setToolTipText("Align Bottom");
alignBottomBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.REGROUP];
regroupComponentsBtn = new JButton(new ImageIcon(img));
regroupComponentsBtn.setActionCommand("regroupComponentsBtn");
regroupComponentsBtn.setBackground(new Color(55, 115, 247));
regroupComponentsBtn.setToolTipText("Regroup Components");
regroupComponentsBtn.setBorder(null);
```

```
img = ActiveState.diffImages[ActiveState.SEPARATOR];
JButton sep1 = new JButton(new ImageIcon(img));
JButton sep2 = new JButton(new ImageIcon(img));
JButton sep3 = new JButton(new ImageIcon(img));
JButton sep4 = new JButton(new ImageIcon(img));
JButton sep5 = new JButton(new ImageIcon(img));
JButton sep6 = new JButton(new ImageIcon(img));

sep1.setBackground(new Color(178, 197, 251));
sep2.setBackground(new Color(178, 197, 251));
sep3.setBackground(new Color(178, 197, 251));
sep4.setBackground(new Color(178, 197, 251));
sep5.setBackground(new Color(178, 197, 251));
sep6.setBackground(new Color(178, 197, 251));

sep1.setBorder(null);
sep2.setBorder(null);
sep3.setBorder(null);
sep4.setBorder(null);
sep5.setBorder(null);
sep6.setBorder(null);

componentToolbar.add(sep1);
componentToolbar.add(sep2);

componentToolbar.addSeparator();
componentToolbar.add(displayPropertiesBtn);
componentToolbar.addSeparator();
componentToolbar.add(editComponentBtn);
componentToolbar.addSeparator();
componentToolbar.add(rotateRightBtn);
componentToolbar.addSeparator();
componentToolbar.add(rotateLeftBtn);
componentToolbar.addSeparator();
componentToolbar.add(flipHorizontalBtn);
componentToolbar.addSeparator();
componentToolbar.add(flipVerticalBtn);
//componentToolbar.addSeparator();
//componentToolbar.add(rotateTextBtn);
componentToolbar.addSeparator();
componentToolbar.add(groupComponentsBtn);
componentToolbar.addSeparator();
componentToolbar.add(ungroupComponentsBtn);
componentToolbar.addSeparator();
componentToolbar.add(regroupComponentsBtn);
```

```
    componentToolbar.addSeparator();
    componentToolbar.add(alignLeftBtn);
    componentToolbar.addSeparator();
    componentToolbar.add(alignBottomBtn);
    componentToolbar.addSeparator();
    componentToolbar.add(alignTopBtn);
    componentToolbar.addSeparator();
    componentToolbar.add(alignRightBtn);
    componentToolbar.addSeparator();
    componentToolbar.add(sep5);

    componentToolbar.setBounds(0, 65, 400, 25);

    this.getContentPane().add(componentToolbar);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawSimulationToolbar() {
    simulationToolbar = new JToolBar();
    simulationToolbar.setFloatable(false);
    simulationToolbar.setBackground(new Color(178, 197, 251));
    simulationToolbar.setVisible(false);

    Image img = ActiveState.btnsExited[ActiveState.NETLIST];
    netListBtn = new JButton(new ImageIcon(img));
    netListBtn.setActionCommand("netListBtn");
    netListBtn.setBackground(new Color(55, 115, 247));
    netListBtn.setToolTipText("Create Netlist");
    netListBtn.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.SETTINGS];
    simulationSettingsBtn = new JButton(new ImageIcon(img));
    simulationSettingsBtn.setActionCommand("simulationSettingsBtn");
    simulationSettingsBtn.setBackground(new Color(55, 115, 247));
    simulationSettingsBtn.setToolTipText(" Edit Simulation Settings");
    simulationSettingsBtn.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.EDITNETLIST];
    editNetlistBtn = new JButton(new ImageIcon(img));
    editNetlistBtn.setActionCommand("editNetlistBtn");
    editNetlistBtn.setBackground(new Color(55, 115, 247));
    editNetlistBtn.setToolTipText("Edit Netlist");
    editNetlistBtn.setBorder(null);
```

```
img = ActiveState.btnsExited[ActiveState.SHOWVOLTAGES];
showVoltagesBtn = new JButton(new ImageIcon(img));
showVoltagesBtn.setActionCommand("showVoltagesBtn");
showVoltagesBtn.setBackground(new Color(55, 115, 247));
showVoltagesBtn.setToolTipText("Show Voltages");
showVoltagesBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.SHOWCURRENTS];
showCurrentsBtn = new JButton(new ImageIcon(img));
showCurrentsBtn.setActionCommand("showCurrentsBtn");
showCurrentsBtn.setBackground(new Color(55, 115, 247));
showCurrentsBtn.setToolTipText("Show Currents");
showCurrentsBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.SHOWHIDENODES];
showHideNodesBtn = new JButton(new ImageIcon(img));
showHideNodesBtn.setActionCommand("showHideNodesBtn");
showHideNodesBtn.setBackground(new Color(55, 115, 247));
showHideNodesBtn.setToolTipText("Show Netlist Nodes");
showHideNodesBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.SHOWCIRML];
showCIRMLBtn = new JButton(new ImageIcon(img));
showCIRMLBtn.setActionCommand("showCIRMLBtn");
showCIRMLBtn.setBackground(new Color(55, 115, 247));
showCIRMLBtn.setToolTipText("View CIRML Output");
showCIRMLBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.SHOWOUTPUTRESULT];
showOutputResultBtn = new JButton(new ImageIcon(img));
showOutputResultBtn.setActionCommand("showOutputResultBtn");
showOutputResultBtn.setBackground(new Color(55, 115, 247));
showOutputResultBtn.setToolTipText("View Simulation Result");
showOutputResultBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.VOLTAGEMARKER];
voltageMarkerBtn = new JButton(new ImageIcon(img));
voltageMarkerBtn.setActionCommand("voltageMarkerBtn");
voltageMarkerBtn.setBackground(new Color(55, 115, 247));
voltageMarkerBtn.setToolTipText("Voltage Marker");
voltageMarkerBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.CURRENTMARKER];
currentMarkerBtn = new JButton(new ImageIcon(img));
currentMarkerBtn.setActionCommand("currentMarkerBtn");
```

```
currentMarkerBtn.setBackground(new Color(55, 115, 247));
currentMarkerBtn.setToolTipText("Current Marker ");
currentMarkerBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.VOLTAGEDIFFERENTIALMARKER];
voltageDifferentialMarkerBtn = new JButton(new ImageIcon(img));
voltageDifferentialMarkerBtn.setActionCommand(
   "voltageDifferentialMarkerBtn");
voltageDifferentialMarkerBtn.setBackground(new Color(55, 115, 247));
voltageDifferentialMarkerBtn.setToolTipText("Voltage Differential Marker");
voltageDifferentialMarkerBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.DRAWRESULT];
drawResultBtn = new JButton(new ImageIcon(img));
drawResultBtn.setActionCommand("drawResultBtn");
drawResultBtn.setBackground(new Color(55, 115, 247));
drawResultBtn.setToolTipText("Draw Result");
drawResultBtn.setBorder(null);

img = ActiveState.diffImages[ActiveState.SEPARATOR];
JButton sep1 = new JButton(new ImageIcon(img));
JButton sep2 = new JButton(new ImageIcon(img));
JButton sep3 = new JButton(new ImageIcon(img));
JButton sep4 = new JButton(new ImageIcon(img));
JButton sep5 = new JButton(new ImageIcon(img));
JButton sep6 = new JButton(new ImageIcon(img));

sep1.setBackground(new Color(178, 197, 251));
sep2.setBackground(new Color(178, 197, 251));
sep3.setBackground(new Color(178, 197, 251));
sep4.setBackground(new Color(178, 197, 251));
sep5.setBackground(new Color(178, 197, 251));
sep6.setBackground(new Color(178, 197, 251));

sep1.setBorder(null);
sep2.setBorder(null);
sep3.setBorder(null);
sep4.setBorder(null);
sep5.setBorder(null);
sep6.setBorder(null);

simulationToolbar.add(sep1);
simulationToolbar.add(sep2);

//simulationToolbar.addSeparator();
```

```
    //simulationToolbar.add(editNetlistBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(netListBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(showHideNodesBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(sep3);
    simulationToolbar.addSeparator();
    simulationToolbar.add(voltageMarkerBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(voltageDifferentialMarkerBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(currentMarkerBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(sep4);
    simulationToolbar.addSeparator();
    simulationToolbar.add(simulationSettingsBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(runBtn);
    simulationToolbar.addSeparator();

    simulationToolbar.add(showVoltagesBtn);
    simulationToolbar.addSeparator();

    simulationToolbar.add(showCurrentsBtn);

    simulationToolbar.addSeparator();
    simulationToolbar.add(showOutputResultBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(showCIRMLBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.add(drawResultBtn);
    simulationToolbar.addSeparator();
    simulationToolbar.addSeparator();
    simulationToolbar.addSeparator();
    //simulationToolbar.addSeparator();
    simulationToolbar.add(sep5);

    simulationToolbar.setBounds(0, 35, 417, 25);

    this.getContentPane().add(simulationToolbar);

  }

////////////////////////////////////////////////////////////////////////////
```

```
private void drawVisitedElementsPanel() {
  Image img = getImage(getDocumentBase(), "ECS/images/horizSep.jpg");
  visitedElementsPanel = new JPanel();
  visitedElementsPanel.setBackground(new Color(131, 167, 248));
  visitedElementsPanel.setLayout(null);

  JButton sep1Btn = new JButton(new ImageIcon(img));
  JButton sep2Btn = new JButton(new ImageIcon(img));
  sep1Btn.setBorder(null);
  sep2Btn.setBorder(null);

  addToPanel(visitedElementsPanel, sep1Btn, 0, 3, 25, 4);
  addToPanel(visitedElementsPanel, sep2Btn, 0, 444, 25, 4);

  img = ActiveState.btnsExited[ActiveState.SRESISTOR];
  VResistorBtn = new JButton(new ImageIcon(img));
  VResistorBtn.setActionCommand("Resistor");
  VResistorBtn.setBackground(new Color(179, 186, 241));
  VResistorBtn.setBorder(null);
  VResistorBtn.setForeground(Color.WHITE);
  VResistorBtn.setVisible(false);
  addToPanel(visitedElementsPanel, VResistorBtn, 0, 3, 25, 20);

  img = ActiveState.btnsExited[ActiveState.SCAPACITOR];
  VCapacitorBtn = new JButton(new ImageIcon(img));
  VCapacitorBtn.setActionCommand("Capacitor");
  VCapacitorBtn.setBackground(new Color(179, 186, 241));
  VCapacitorBtn.setBorder(null);
  VCapacitorBtn.setForeground(Color.WHITE);
  VCapacitorBtn.setVisible(false);
  addToPanel(visitedElementsPanel, VCapacitorBtn, 0, 25, 25, 20);

  img = ActiveState.btnsExited[ActiveState.SINDUCTOR];
  VInductorBtn = new JButton(new ImageIcon(img));
  VInductorBtn.setActionCommand("Inductor");
  VInductorBtn.setBackground(new Color(179, 186, 241));
  VInductorBtn.setBorder(null);
  VInductorBtn.setForeground(Color.WHITE);
  VInductorBtn.setVisible(false);
  addToPanel(visitedElementsPanel, VInductorBtn, 0, 47, 25, 20);

  img = ActiveState.btnsExited[ActiveState.STRANSFORMERA];
  VPPTransformerBtn = new JButton(new ImageIcon(img));
  VPPTransformerBtn.setActionCommand("PPTransformer");
  VPPTransformerBtn.setBackground(new Color(179, 186, 241));
```

```
VPPTransformerBtn.setBorder(null);
VPPTransformerBtn.setForeground(Color.WHITE);
VPPTransformerBtn.setVisible(false);
addToPanel(visitedElementsPanel, VPPTransformerBtn, 0, 69, 25, 20);

img = ActiveState.btnsExited[ActiveState.STRANSFORMERB];
VPNTransformerBtn = new JButton(new ImageIcon(img));
VPNTransformerBtn.setActionCommand("PNTransformer");
VPNTransformerBtn.setBackground(new Color(179, 186, 241));
VPNTransformerBtn.setBorder(null);
VPNTransformerBtn.setForeground(Color.WHITE);
VPNTransformerBtn.setVisible(false);
addToPanel(visitedElementsPanel, VPNTransformerBtn, 0, 91, 25, 20);

img = ActiveState.btnsExited[ActiveState.STRANSFORMERC];
VNPTransformerBtn = new JButton(new ImageIcon(img));
VNPTransformerBtn.setActionCommand("NPTransformer");
VNPTransformerBtn.setBackground(new Color(179, 186, 241));
VNPTransformerBtn.setBorder(null);
VNPTransformerBtn.setForeground(Color.WHITE);
VNPTransformerBtn.setVisible(false);
addToPanel(visitedElementsPanel, VNPTransformerBtn, 0, 113, 25, 20);

img = ActiveState.btnsExited[ActiveState.STRANFORMERD];
VNNTransformerBtn = new JButton(new ImageIcon(img));
VNNTransformerBtn.setActionCommand("NNTransformer");
VNNTransformerBtn.setBackground(new Color(179, 186, 241));
VNNTransformerBtn.setBorder(null);
VNNTransformerBtn.setForeground(Color.WHITE);
VNNTransformerBtn.setVisible(false);
addToPanel(visitedElementsPanel, VNNTransformerBtn, 0, 135, 25, 20);

img = ActiveState.btnsExited[ActiveState.SDCVSOURCE];
VDCVoltageBtn = new JButton(new ImageIcon(img));
VDCVoltageBtn.setActionCommand("DC Voltage Source");
VDCVoltageBtn.setBackground(new Color(179, 186, 241));
VDCVoltageBtn.setBorder(null);
VDCVoltageBtn.setForeground(Color.WHITE);
VDCVoltageBtn.setVisible(false);
addToPanel(visitedElementsPanel, VDCVoltageBtn, 0, 157, 25, 20);

img = ActiveState.btnsExited[ActiveState.SDCCSOURCE];
VDCCurrentBtn = new JButton(new ImageIcon(img));
VDCCurrentBtn.setActionCommand("DC Current Source");
VDCCurrentBtn.setBackground(new Color(179, 186, 241));
```

```
VDCCurrentBtn.setBorder(null);
VDCCurrentBtn.setForeground(Color.WHITE);
VDCCurrentBtn.setVisible(false);
addToPanel(visitedElementsPanel, VDCCurrentBtn, 0, 179, 25, 20);

img = ActiveState.btnsExited[ActiveState.SACVSOURCE];
VACVoltageBtn = new JButton(new ImageIcon(img));
VACVoltageBtn.setActionCommand("AC Voltage Source");
VACVoltageBtn.setBackground(new Color(179, 186, 241));
VACVoltageBtn.setBorder(null);
VACVoltageBtn.setForeground(Color.WHITE);
VACVoltageBtn.setVisible(false);
addToPanel(visitedElementsPanel, VACVoltageBtn, 0, 201, 25, 20);

img = ActiveState.btnsExited[ActiveState.SACCSOURCE];
VACCurrentBtn = new JButton(new ImageIcon(img));
VACCurrentBtn.setActionCommand("AC Current Source");
VACCurrentBtn.setBackground(new Color(179, 186, 241));
VACCurrentBtn.setBorder(null);
VACCurrentBtn.setForeground(Color.WHITE);
VACCurrentBtn.setVisible(false);
addToPanel(visitedElementsPanel, VACCurrentBtn, 0, 223, 25, 20);

img = ActiveState.btnsExited[ActiveState.SDEPVCVS];
VVCVSBtn = new JButton(new ImageIcon(img));
VVCVSBtn.setActionCommand("Voltage Controlled Voltage Source");
VVCVSBtn.setBackground(new Color(179, 186, 241));
VVCVSBtn.setBorder(null);
VVCVSBtn.setForeground(Color.WHITE);
VVCVSBtn.setVisible(false);
addToPanel(visitedElementsPanel, VVCVSBtn, 0, 245, 25, 20);

img = ActiveState.btnsExited[ActiveState.SDEPVCCS];
VVCCSBtn = new JButton(new ImageIcon(img));
VVCCSBtn.setActionCommand("Voltage Controlled Current Source");
VVCCSBtn.setBackground(new Color(179, 186, 241));
VVCCSBtn.setBorder(null);
VVCCSBtn.setForeground(Color.WHITE);
VVCCSBtn.setVisible(false);
addToPanel(visitedElementsPanel, VVCCSBtn, 0, 267, 25, 20);

img = ActiveState.btnsExited[ActiveState.SDEPCCVS];
VCCVSBtn = new JButton(new ImageIcon(img));
VCCVSBtn.setActionCommand("Current Controlled Voltage Source");
VCCVSBtn.setBackground(new Color(179, 186, 241));
```

```
VCCVSBtn.setBorder(null);
VCCVSBtn.setForeground(Color.WHITE);
VCCVSBtn.setVisible(false);
addToPanel(visitedElementsPanel, VCCVSBtn, 0, 289, 25, 20);

img = ActiveState.btnsExited[ActiveState.SDEPCCCS];
VCCCSBtn = new JButton(new ImageIcon(img));
VCCCSBtn.setActionCommand("Current Controlled Current Source");
VCCCSBtn.setBackground(new Color(179, 186, 241));
VCCCSBtn.setBorder(null);
VCCCSBtn.setForeground(Color.WHITE);
VCCCSBtn.setVisible(false);
addToPanel(visitedElementsPanel, VCCCSBtn, 0, 311, 25, 20);

img = ActiveState.btnsExited[ActiveState.SCONNECTOR];
VConnLineBtn = new JButton(new ImageIcon(img));
VConnLineBtn.setActionCommand("Connector");
VConnLineBtn.setBackground(new Color(179, 186, 241));
VConnLineBtn.setBorder(null);
VConnLineBtn.setForeground(Color.WHITE);
VConnLineBtn.setVisible(false);
addToPanel(visitedElementsPanel, VConnLineBtn, 0, 333, 25, 20);

img = ActiveState.btnsExited[ActiveState.SCONNODE];
VConnNodeBtn = new JButton(new ImageIcon(img));
VConnNodeBtn.setActionCommand("Connection Point");
VConnNodeBtn.setBackground(new Color(179, 186, 241));
VConnNodeBtn.setBorder(null);
VConnNodeBtn.setForeground(Color.WHITE);
VConnNodeBtn.setVisible(false);
addToPanel(visitedElementsPanel, VConnNodeBtn, 0, 355, 25, 20);

img = ActiveState.btnsExited[ActiveState.SGROUND];
VGroundBtn = new JButton(new ImageIcon(img));
VGroundBtn.setActionCommand("Ground");
VGroundBtn.setBackground(new Color(179, 186, 241));
VGroundBtn.setBorder(null);
VGroundBtn.setForeground(Color.WHITE);
VGroundBtn.setVisible(false);
addToPanel(visitedElementsPanel, VGroundBtn, 0, 377, 25, 20);

img = ActiveState.btnsExited[ActiveState.SCURRENTCOMP];
VCurrentMarkerBtn = new JButton(new ImageIcon(img));
VCurrentMarkerBtn.setActionCommand("Current Marker");
VCurrentMarkerBtn.setBackground(new Color(179, 186, 241));
```

```
    VCurrentMarkerBtn.setBorder(null);
    VCurrentMarkerBtn.setForeground(Color.WHITE);
    VCurrentMarkerBtn.setVisible(false);
    addToPanel(visitedElementsPanel, VCurrentMarkerBtn, 0, 399, 25, 20);

    img = ActiveState.btnsExited[ActiveState.VOLTAGEM];
    VVoltageMarkerBtn = new JButton(new ImageIcon(img));
    VVoltageMarkerBtn.setActionCommand("Voltage Marker");
    VVoltageMarkerBtn.setBackground(new Color(179, 186, 241));
    VVoltageMarkerBtn.setBorder(null);
    VVoltageMarkerBtn.setForeground(Color.WHITE);
    VVoltageMarkerBtn.setVisible(false);
    addToPanel(visitedElementsPanel, VVoltageMarkerBtn, 0, 421, 25, 20);

    img = ActiveState.btnsExited[ActiveState.SNODEL];
    VVoltageDiffMarkerBtn = new JButton(new ImageIcon(img));
    VVoltageDiffMarkerBtn.setActionCommand("Voltage Differential Marker");
    VVoltageDiffMarkerBtn.setBackground(new Color(179, 186, 241));
    VVoltageDiffMarkerBtn.setBorder(null);
    VVoltageDiffMarkerBtn.setForeground(Color.WHITE);
    VVoltageDiffMarkerBtn.setVisible(false);
    addToPanel(visitedElementsPanel, VVoltageDiffMarkerBtn, 0, 421, 25, 20);

    add(visitedElementsPanel, 634, 50, 25, 452);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawPanelLD() {
    panelLD = new JPanel();
    panelLD.setLayout(null);
//panelLD.setBackground(new Color(128,128,255));
    panelLD.setBackground(new Color(91, 141, 255));
    //panelLD.setBackground(Color.WHITE);
    Image img = getImage(getDocumentBase(), "ECS/images/logo.jpg");
    JLabel logo1 = new JLabel(new ImageIcon(img));
    img = getImage(getDocumentBase(), "ECS/images/logo2.jpg");
    JLabel logo2 = new JLabel(new ImageIcon(img));

    panelLD.add(logo1);
    panelLD.add(logo2);
    // add(logo1, 564, 50, 160, 40);
    //add(logo2, 564, 522, 160, 40);

    add(panelLD, 0, 5, 599, 34);
```

```
  }

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
  private void drawPanelR() {
    panelR = new JPanel();
    //panelR.setBackground(new Color(128,128,255));

    //panelR.setBorder(BorderFactory.createEtchedBorder
    //                   (Color.WHITE, Color.BLUE));
    // panelR.setBackground(new Color(91,141,255));

    Image img = getImage(getDocumentBase(),
                      "ECS/images/Electronic_Circuits_Simulator.jpg");
    title = new JLabel(new ImageIcon(img));

    add(title, 680, -7, 279, 45);
    //add(panelR, 600, 0, 285, 40);
  }

////////////////////////////////////////////////////////////////////////////
  private void drawPanelDL() {
    panelDL = new JPanel();
    panelDL.setBackground(Color.WHITE);

    add(panelDL, 30, 550, 600, 15);
  }

////////////////////////////////////////////////////////////////////////////
  private void drawPanelDR() {
    panelDR = new JPanel();
//panelDR.setBackground(new Color(128,128,255));
    panelDR.setBackground(Color.WHITE);

    add(panelDR, 600, 540, 290, 15);
  }

////////////////////////////////////////////////////////////////////////////

  private void drawStatusBar() {

    statusPanel = new JPanel();
```

```
    statusPanel.setLayout(null);

    Image img = getImage(getDocumentBase(), "ECS/images/statusbar.jpg");
    JLabel lbl = new JLabel(new ImageIcon(img));
    lbl.setBounds(0, 0, 589, 20);
    statusPanel.add(lbl);
    add(statusPanel, 35, 536, 589, 20);

  }

////////////////////////////////////////////////////////////////////////////

  private void uploadComponentsImgs() {
    ImgComponents.imageNormal =
       new Image[Array.getLength(ActiveState.StrCompGifs)];
    ImgComponents.imageActive =
       new Image[Array.getLength(ActiveState.StrCompGifs)];

    ActiveState.ImageNormal = new Image[Array.getLength(ActiveState.StrCompGifs)];
    ActiveState.ImageActive = new Image[Array.getLength(ActiveState.StrCompGifs)];
    ActiveState.btnsEntered = new Image[Array.getLength(ActiveState.strBtnsImgs)];
    ActiveState.btnsExited = new Image[Array.getLength(ActiveState.strBtnsImgs)];

    ActiveState.diffImages = new Image[Array.getLength(ActiveState.Images)];

    for (int i = 0; i < Array.getLength(ActiveState.StrCompGifs); i++) {
      ImgComponents.imageNormal[i] = this.getImage(getDocumentBase(),
         "ECS/images/" + ActiveState.StrCompGifs[i] + ".gif");
      ImgComponents.imageActive[i] = this.getImage(getDocumentBase(),
         "ECS/images/" + ActiveState.StrCompGifs[i] + "_active.gif");
      ActiveState.ImageNormal[i] = this.getImage(getDocumentBase(),
                                      "ECS/images/" +
                                      ActiveState.StrCompGifs[i] +
                                      ".gif");
      ActiveState.ImageActive[i] = this.getImage(getDocumentBase(),
                                      "ECS/images/" +
                                      ActiveState.StrCompGifs[i] +
                                      "_active.gif");

    }

    for (int i = 0; i < Array.getLength(ActiveState.strBtnsImgs); i++) {
      ActiveState.btnsExited[i] = this.getImage(getDocumentBase(),
                                     "ECS/images/" +
                                     ActiveState.strBtnsImgs[i] +
```

```
                                        ".jpg");
      ActiveState.btnsEntered[i] = this.getImage(getDocumentBase(),
                                        "ECS/images/" +
                                        ActiveState.strBtnsImgs[i] +
                                        "_active.jpg");
    }

    for (int i = 0; i < Array.getLength(ActiveState.Images); i++) {

      ActiveState.diffImages[i] = this.getImage(getDocumentBase(),
                                        "ECS/images/" +
                                        ActiveState.Images[i] + ".jpg");

    }
  }

////////////////////////////////////////////////////////////////////////////
  public void drawAllComps() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numbComp = 0; numbComp < Array.getLength(allComps[type]);
          numbComp++) {
        if (allComps[type][numbComp] != null) {
          allComps[type][numbComp].draw(gridCanvas, this);

        }
      }
    }

    //drawLabels();
  }

////////////////////////////////////////////////////////////////////////////
  /* public static void drawLabels() {
     for (int i = 0; i < txtLabels.size(); i++) {
       Labels txt = (Labels) txtLabels.get(i);
       txt.drawString(gridCanvas);
     }
   }*/

////////////////////////////////////////////////////////////////////////////
  private void clearAllComps() {

    for (int nCategory = ActiveState.RESISTOR; nCategory < componentsCount;
        nCategory++) {
      for (int numbComp = 0;
```

```
      numbComp < Array.getLength(allComps[nCategory]) - 1; numbComp++) {
   if (allComps[nCategory][numbComp].getComponentType() !=
      ActiveState.CONNECTOR
      &&
      allComps[nCategory][numbComp].getComponentType() !=
      ActiveState.GROUND
      &&
      allComps[nCategory][numbComp].getComponentType() !=
      ActiveState.NODEL
      &&
      allComps[nCategory][numbComp].getComponentType() !=
      ActiveState.NODER
      &&
      allComps[nCategory][numbComp].getComponentType() !=
      ActiveState.CONNODE
      &&
      allComps[nCategory][numbComp].getComponentType() !=
      ActiveState.CURRENTMARK
      &&
      allComps[nCategory][numbComp].getComponentType() !=
      ActiveState.VOLTAGEMARK) {
     gridCanvas.remove(allComps[nCategory][numbComp].txtName);
     gridCanvas.remove(allComps[nCategory][numbComp].txtValue);
   }
  }
}

for (int nCategory = ActiveState.RESISTOR; nCategory < componentsCount;
    nCategory++) {
  deletedComps[nCategory] = (Components[]) copyComponentsArray(allComps[
     nCategory]);
}

for (int nCategory = ActiveState.RESISTOR; nCategory < componentsCount;
    nCategory++) {
  allComps[nCategory] = (Components[]) cutArray(allComps[nCategory]);
}

drawComps = (DrawTool[]) cutArray(drawComps);

gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
gridCanvas.gridImage.setColor(canvasBackgroundColor);
gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
gridCanvas.gridImage.setColor(canvasForegroundColor);
if (ActiveState.showGrid) {
```

```
      gridCanvas.drawGridLayout(startPoint);
    }
    gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  private void rotateComponent(int dir) {

    for (int type = 0; type < componentsCount; type++) {
      for (int numbComp = 0; numbComp < Array.getLength(allComps[type]);
          numbComp++) {
        if (allComps[type][numbComp] != null) {

          if (allComps[type][numbComp].getActiveState()) {

            int temp = allComps[type][numbComp].getAlignment();
            int[] pos = allComps[type][numbComp].getPosition();

////////////
            switch (allComps[type][numbComp].getComponentType()) {
              case ActiveState.TRANSFORMERA:
              case ActiveState.TRANSFORMERB:
              case ActiveState.TRANSFORMERC:
              case ActiveState.TRANSFORMERD:
                switch (dir) {
                  case 0:
                    switch (temp) {
                      case 0:
                        temp++;
                        pos[0] += 50;
                        break;
                      case 1:
                        temp++;
                        pos[1] += 50;
                        break;
                      case 2:
                        temp++;
                        pos[0] -= 50;
                        break;
                      default:
                        temp = 0;
                        pos[1] -= 50;
                    }
                    break;
```

```
    case 1:
      switch (temp) {
        case 0:
          temp++;
          pos[0] -= 50;
          break;
        case 1:
          temp++;
          pos[1] += 50;
          break;
        case 2:
          temp++;
          pos[0] += 50;

          break;
        default:
          temp = 0;
          pos[1] -= 50;
      }
      break;
  }
  break;
default:
  switch (dir) {
    case 0:
      switch (temp) {
        case 0:
          temp++;
          pos[0] += 25;
          pos[1] -= 25;
          break;
        case 1:
          temp++;
          pos[0] += 25;
          pos[1] += 25;
          break;
        case 2:
          temp++;
          pos[0] -= 25;
          pos[1] += 25;
          break;
        default:
          temp = 0;
          pos[0] -= 25;
          pos[1] -= 25;
```

```
                }
                break;
              case 1:
                switch (temp) {
                  case 0:
                    temp++;
                    pos[0] -= 25;
                    pos[1] -= 25;
                    break;
                  case 1:
                    temp++;
                    pos[0] -= 25;
                    pos[1] += 25;
                    break;
                  case 2:
                    temp++;
                    pos[0] += 25;
                    pos[1] += 25;
                    break;
                  default:
                    temp = 0;
                    pos[0] += 25;
                    pos[1] -= 25;
                }
                break;
            }

          }
          allComps[type][numbComp].setAlignment(temp);
          allComps[type][numbComp].setNodes();

//////////
          }
        }
      }
    }

    //   redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void rotateGroup(int rotation) {

    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
```

```
      int[] node1 = allComps[type][numb].getNode(0).getPosition();
      int[] node2 = allComps[type][numb].getNode(1).getPosition();

      for (int cat = 0; cat < componentsCount; cat++) {
        for (int numbComp = 0; numbComp < Array.getLength(allComps[cat]) - 1;
            numbComp++) {
          if (!allComps[cat][numbComp].equals(allComps[type][numb])) {
            for (int m = 0; m < allComps[cat][numbComp].getNumNodes(); m++) {
              int[] node = allComps[cat][numbComp].getNode(m).getPosition();
              if ( (node1[0] == node[0]) && (node1[1] == node[1])) {

                NeighbourComp c = new NeighbourComp(cat, numbComp, 0, m);
                allComps[type][numb].NeighbourComps.add(c);
              }
              else if ( (node2[0] == node[0]) && (node2[1] == node[1])) {

                NeighbourComp c = new NeighbourComp(cat, numbComp, 1, m);
                allComps[type][numb].NeighbourComps.add(c);
              }

            }

          }
        }
      }
    }
////////////////////////////////////////////////
    for (int i = 0; i < componentsCount; i++) {
      for (int k = 0; k < Array.getLength(allComps[i]) - 1; k++) {
        for (int d = 0; d < allComps[i][k].NeighbourComps.size(); d++) {
          NeighbourComp comp = (NeighbourComp) allComps[i][k].NeighbourComps.
              elementAt(d);
          int compType = (int) comp.compType;
          int compNumber = (int) comp.compNumb;
          int nodeNumber1 = (int) comp.nodeNumb1;
          int nodeNumber2 = (int) comp.nodeNumb2;

        }
      }
    }
////////////////////////////////////////////
    rotateComponent(rotation);
```

```
boolean allRotated = false;

allComps[0][0].rotated = true;
while (!allRotated) {
  for (int type = 0; type < componentsCount; type++) {
    for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
        numbComp++) {
      if (allComps[type][numbComp].rotated &&
        !allComps[type][numbComp].rotatedFinished) {
       for (int i = 0; i < allComps[type][numbComp].NeighbourComps.size();
          i++) {
        NeighbourComp comp = (NeighbourComp) allComps[type][numbComp].
          NeighbourComps.elementAt(i);
        int compType = (int) comp.compType;
        int compNumber = (int) comp.compNumb;
        int nodeNumber1 = (int) comp.nodeNumb1;
        int nodeNumber2 = (int) comp.nodeNumb2;

        if (!allComps[compType][compNumber].rotated) {

          int align = allComps[compType][compNumber].getAlignment();
          int[] movedNode = allComps[type][numbComp].getNode(nodeNumber1).
            getPosition();

          if ( (align == 1 || align == 3) && nodeNumber2 == 0) {
            int[] pos = new int[2];
            pos[0] = movedNode[0];
            pos[1] = movedNode[1];
            allComps[compType][compNumber].setPosition(movedNode);
            allComps[compType][compNumber].setNodes();
            movedNode[1] -= 75;
          }
          else if ( (align == 1 || align == 3) && nodeNumber2 == 1) {
            movedNode[1] -= 75;
            allComps[compType][compNumber].setPosition(movedNode);
            allComps[compType][compNumber].setNodes();
            movedNode[1] += 75;
          }
          else if ( (align == 0 || align == 2) && nodeNumber2 == 0) {

            movedNode[0] -= 75;
            allComps[compType][compNumber].setPosition(movedNode);
            allComps[compType][compNumber].setNodes();
            movedNode[0] += 75;
          }
```

```
            else if ( (align == 0 || align == 2) && nodeNumber2 == 1) {

              movedNode[0] += 25;
              allComps[compType][compNumber].setPosition(movedNode);
              allComps[compType][compNumber].setNodes();
              movedNode[0] -= 25;
             }

//allComps[compType][compNumber].done=true;

            }

           }

          allComps[type][numbComp].rotatedFinished = true;
         }
        }
       }
       /*
        for(int i=0;i<componentsCount;i++)
        {
        for(int j=0;j<Array.getLength(allComps[i])-1;j++)
        {
         if(allComps[i][j].done)allComps[i][j].rotated=;
        }
        }*/

       allRotated = true;
       /*for(int i=0;i<componentsCount;i++)
        {
        for(int j=0;j<Array.getLength(allComps[i])-1;j++)
        {
        if(!allComps[i][j].rotated)allRotated=false;
        }
        }
        */
    }

    for (int i = 0; i < componentsCount; i++) {
      for (int j = 0; j < Array.getLength(allComps[i]) - 1; j++) {
        allComps[i][j].rotated = false;
        allComps[i][j].done = false;
        allComps[i][j].rotatedFinished = false;
      }
    }
```

```
    redrawGridCanvas();

  }

////////////////////////////////////////////////////////////////////////////

  private void showPopupMenu(int[] pos, MouseEvent me) {
    if (ActiveState.active) {

      switch (ActiveState.activeComp.getComponentType()) {
        case ActiveState.CONNODE:
          enterValue.setEnabled(false);
          singleElementRotateRight.setEnabled(false);
          singleElementRotateLeft.setEnabled(false);
          singleElementDelete.setEnabled(true);
          singleElementPopmenu.show(gridCanvas, pos[0], pos[1]);
          break;
        case ActiveState.GROUND:
          enterValue.setEnabled(false);
          singleElementRotateRight.setEnabled(true);
          singleElementRotateLeft.setEnabled(true);
          singleElementDelete.setEnabled(true);
          singleElementPopmenu.show(gridCanvas, pos[0], pos[1]);
          break;
        case ActiveState.CONNECTOR:
          deleteLine.setEnabled(true);
          connectionLinePopupMenu.show(gridCanvas, pos[0], pos[1]);

          break;
        default: {
          enterValue.setEnabled(true);
          singleElementRotateRight.setEnabled(true);
          singleElementRotateLeft.setEnabled(true);
          singleElementDelete.setEnabled(true);
          singleElementPopmenu.show(gridCanvas, pos[0], pos[1]);
        }
      }

    }
    else if (ActiveState.multiActive) {
      ;
      for (int numb = 0;
           numb < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; numb++) {
        if (allComps[ActiveState.CONNECTOR][numb].getActiveState()) {
```

```
      multiElementsRotateLeft.setEnabled(false);
      multiElementsRotateRight.setEnabled(false);
     }
    }
    if (ActiveState.disableGroup) {
     multiElementsGroup.setEnabled(false);
    }
    else {
     multiElementsGroup.setEnabled(true);

    }
    if (ActiveState.disableUngroup) {
     multiElementsUngroup.setEnabled(false);
    }
    else {
     multiElementsUngroup.setEnabled(true);

    }

    multiElementsPopmenu.show(gridCanvas, pos[0], pos[1]);
    //ActiveState.multiActive = false;
   }
   else if (SwingUtilities.isRightMouseButton(me)) {
    clearAll.setEnabled(true);
    if (ActiveState.DELETED) {
     undo.setEnabled(true);
     ActiveState.DELETED = false;
    }
    else {
     undo.setEnabled(false);
    }
    if (ActiveState.singleCopied || ActiveState.multipleCopied) {
     paste.setEnabled(true);
    }
    else {
     paste.setEnabled(false);
    }
    if (undoStack.top > 0) {
     undo.setEnabled(true);
    }
    else {
     undo.setEnabled(false);

    }
    gridPopupMenu.show(gridCanvas, pos[0], pos[1]);
```

```
    }

  }

////////////////////////////////////////////////////////////////////////////
  public void finalizeDepInput(Components comp) {
    currentComp = comp;
    switch (comp.getComponentType()) {
      case ActiveState.DEPVCVS:
      case ActiveState.DEPVCCS:
        if ( ( (DepComp) comp).controllers[0] != null &&
           ( (DepComp) comp).controllers[1] != null) {
          for (int numbComp = 0;
              numbComp < Array.getLength(allComps[ActiveState.NODEL]);
              numbComp++) {
            if ( ( (DepComp) comp).controllers[0].equals(allComps[ActiveState.
                NODEL][numbComp])) {

              ( (DepComp) comp).controllers[0].setControl(comp.getControl());
              ( (DepComp) comp).controllers[0].setComponentName(comp.getControl());
              allComps[ActiveState.NODEL][numbComp] = ( (DepComp) comp).
                  controllers[0];
              ( (DepComp) comp).controllers[1].setControl(comp.getControl());
              ( (DepComp) comp).controllers[1].setComponentName(comp.getControl());
              allComps[ActiveState.NODER][numbComp] = ( (DepComp) comp).
                  controllers[1];

            }
          }

        }

        else {
//int length=Array.getLength(allComps[ActiveState.NODEL]);
//allComps[ActiveState.NODEL][length-1]=ActiveState.createNew(ActiveState.NODEL);
//allComps[ActiveState.NODEL][length-1].setControl(comp.getControl());
//allComps[ActiveState.NODEL][length-1].setComponentName(comp.getControl());
//((DepComp) comp).controllers[0]=allComps[ActiveState.NODEL][length-1];
//length=Array.getLength(allComps[ActiveState.NODER]);
//allComps[ActiveState.NODER][length-1]=ActiveState.createNew(ActiveState.NODER);
//allComps[ActiveState.NODER][length-1].setControl(comp.getControl());
//allComps[ActiveState.NODER][length-1].setComponentName(comp.getControl());
//((DepComp) comp).controllers[1]=allComps[ActiveState.NODER][length-1];
//allComps[ActiveState.NODEL]=(Components[]) growArray(allComps[ActiveState.NODEL]);
//allComps[ActiveState.NODER]=(Components[]) growArray(allComps[ActiveState.NODER]);
```

```
        ActiveState.CurComp = ActiveState.createNew(ActiveState.NODEL);
        ( (NodeL) ActiveState.CurComp).setAsController();
        ActiveState.CurComp.setControl(comp.getControl());
        ActiveState.CurComp.setComponentName(comp.getControl());
       }

       break;
     case ActiveState.DEPCCVS:
     case ActiveState.DEPCCCS:
       if ( ( (DepComp) comp).controllers[0] != null) {
        for (int numbComp = 0;
             numbComp < Array.getLength(allComps[ActiveState.CURRENTCOMP]);
             numbComp++) {
          if ( ( (DepComp) comp).controllers[0].equals(allComps[ActiveState.
              CURRENTCOMP][numbComp])) {
            ( (DepComp) comp).controllers[0].setControl(comp.getControl());
            ( (DepComp) comp).controllers[0].setComponentName(comp.getControl());
            allComps[ActiveState.CURRENTCOMP][numbComp] = ( (DepComp) comp).
                controllers[0];
          }
        }
       }
       else {
        ActiveState.CurComp = ActiveState.createNew(ActiveState.CURRENTCOMP);
        ActiveState.CurComp.setControl(comp.getControl());
        ( (CurrentComp) ActiveState.CurComp).setAsController();
        ActiveState.CurComp.setComponentName(comp.getControl());
       }

       break;
     default:
    }
    redrawGridCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void showUserInput() {

    saveOriginalSettings();
    if (ActiveState.activeComp != null) {
      switch (ActiveState.activeComp.getComponentType()) {
        case ActiveState.RESISTOR:
        case ActiveState.INDUCTOR:
```

```
    case ActiveState.CAPACITOR:
      PassiveElementsInputFrame passiveDlg = new PassiveElementsInputFrame();
      passiveDlg.setComponent(ActiveState.activeComp, this);
      passiveDlg.setVisible(true);
      //passiveDlg.requestFocus();
      break;
    case ActiveState.TRANSFORMERA:
    case ActiveState.TRANSFORMERB:
    case ActiveState.TRANSFORMERC:
    case ActiveState.TRANSFORMERD:
      TransformerInputFrame transformerDlg = new TransformerInputFrame();
      transformerDlg.setComponent(ActiveState.activeComp, this);
      transformerDlg.setVisible(true);
      break;
    case ActiveState.DEPCCCS:
    case ActiveState.DEPCCVS:
    case ActiveState.DEPVCCS:
    case ActiveState.DEPVCVS:
      DepSourcesInputFrame depDlg = new DepSourcesInputFrame();
      depDlg.setComponent(ActiveState.activeComp, this);
      depDlg.setVisible(true);
      break;
    case ActiveState.CURRENTCOMP:
      CurrentInputFrame currentDlg = new CurrentInputFrame();
      currentDlg.setComponent(ActiveState.activeComp, this);
      currentDlg.setVisible(true);
      break;
    case ActiveState.DCVSOURCE:
    case ActiveState.DCCSOURCE:
      IndepDCSourcesInputFrame dcDlg = new IndepDCSourcesInputFrame();
      dcDlg.setComponent(ActiveState.activeComp, this);
      dcDlg.setVisible(true);
      break;
    case ActiveState.ACVSOURCE:
    case ActiveState.ACCSOURCE:
      IndepACSourcesInputFrame acDlg = new IndepACSourcesInputFrame();
      acDlg.setComponent(ActiveState.activeComp, this);
      acDlg.setVisible(true);
      break;
    default:

  }
}

if (activeComponent != null) {
```

```
switch (activeComponent.getComponentType()) {
  case ActiveState.RESISTOR:
  case ActiveState.INDUCTOR:
  case ActiveState.CAPACITOR:
    PassiveElementsInputFrame passiveDlg = new PassiveElementsInputFrame();
    passiveDlg.setComponent(activeComponent, this);
    passiveDlg.setVisible(true);
    //passiveDlg.requestFocus();
    break;
  case ActiveState.TRANSFORMERA:
  case ActiveState.TRANSFORMERB:
  case ActiveState.TRANSFORMERC:
  case ActiveState.TRANSFORMERD:
    TransformerInputFrame transformerDlg = new TransformerInputFrame();
    transformerDlg.setComponent(activeComponent, this);
    transformerDlg.setVisible(true);
    break;
  case ActiveState.DEPCCCS:
  case ActiveState.DEPCCVS:
  case ActiveState.DEPVCCS:
  case ActiveState.DEPVCVS:
    DepSourcesInputFrame depDlg = new DepSourcesInputFrame();
    depDlg.setComponent(activeComponent, this);
    depDlg.setVisible(true);
    break;
  case ActiveState.CURRENTCOMP:
    CurrentInputFrame currentDlg = new CurrentInputFrame();
    currentDlg.setComponent(activeComponent, this);
    currentDlg.setVisible(true);
    break;
  case ActiveState.DCVSOURCE:
  case ActiveState.DCCSOURCE:
    IndepDCSourcesInputFrame dcDlg = new IndepDCSourcesInputFrame();
    dcDlg.setComponent(activeComponent, this);
    dcDlg.setVisible(true);
    break;
  case ActiveState.ACVSOURCE:
  case ActiveState.ACCSOURCE:
    IndepACSourcesInputFrame acDlg = new IndepACSourcesInputFrame();
    acDlg.setComponent(activeComponent, this);
    acDlg.setVisible(true);
    break;
  default:
    //ErrorMessage errMsg=new ErrorMessage("Invalid Component Type");
    //errMsg.setVisible(true);
```

```
    }
   }
   /*    if (activeLabel != null) {
         for (int i = 0; i < txtLabels.size(); i++) {
          Labels txt = (Labels) txtLabels.get(i);
          if (activeLabel.getText().equals(txt.getText())) {
           InsertTextFrame txtFrame = new InsertTextFrame(this);
           txtFrame.setText(activeLabel.getText());
           txtFrame.show();
          }
         }
        }*/
  }

////////////////////////////////////////////////////////////////////////////
  private void deleteComponent(boolean tag) {

   if (ActiveState.active || ActiveState.multiActive) {
    if (tag) {

     for (int type = 0; type < componentsCount; type++) {
      deletedComps[type] = (Components[]) cutArray(deletedComps[type]);

     }
    }

    for (int type = 0; type < componentsCount; type++) {
     for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
         numbComp++) {

      if (allComps[type][numbComp].getActiveState()) {
       ActiveState.active = false;
       ActiveState.activeComp = null;
       ActiveState.DELETED = true;
       int length = Array.getLength(deletedComps[type]);
       deletedComps[type][length - 1] = allComps[type][numbComp];
       allComps[type][numbComp] = null;
       allComps[type] = (Components[]) shrinkArray(allComps[type],
          numbComp);
       deletedComps[type] = (Components[]) growArray(deletedComps[type]);
       deleteComponent(false);

      }
     }
    }
```

```
      redrawGridCanvas();
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void undoDeleteComponent() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numbComp = 0; numbComp < Array.getLength(deletedComps[type]);
           numbComp++) {
        if (deletedComps[type][numbComp] != null) {
          int length = Array.getLength(allComps[type]);
          allComps[type][length - 1] = deletedComps[type][numbComp];
          allComps[type] = (Components[]) growArray(allComps[type]);
//deletedComps[type]=(Components[]) shrinkArray(deletedComps[type],numbComp);
        }
      }
    }

    for (int type = 0; type < componentsCount; type++) {
      deletedComps[type] = (Components[]) cutArray(deletedComps[type]);
    }

  }

////////////////////////////////////////////////////////////////////////////
  private Object copyComponentsArray(Object a) {

    Class cl = a.getClass();
    if (!cl.isArray()) {
      return null;
    }
    Class componentType = a.getClass().getComponentType();
    int length = Array.getLength(a);
    Object newArray = Array.newInstance(componentType,
                                        length);
    System.arraycopy(a, 0, newArray, 0, length);
    return newArray;
  }

////////////////////////////////////////////////////////////////////////////

  private int[] calculateAppPoint(int[] point) {
    int ix, iy;
```

```
    int[] xy = new int[2];

    ix = point[0] / 25;
    iy = point[1] / 25;
    if (point[0] % 25 <= 12) {
      xy[0] = ix * 25;
    }
    else {
      xy[0] = ix * 25 + 25;

    }
    if (point[1] % 25 <= 12) {
      xy[1] = iy * 25;
    }
    else {
      xy[1] = iy * 25 + 25;

    }
    return xy;
  }

//*************************************************************************************/
/
  private void optimizeNetwork() {

    for (int connNumb = 0;
         connNumb < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1;
         connNumb++) {
      for (int l = connNumb + 1;
           l < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; l++) {
        MergeNet(allComps[ActiveState.CONNECTOR][l],
                 allComps[ActiveState.CONNECTOR][connNumb]);
      }

      for (int type = 0; type < componentsCount; type++) {
        for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
             numbComp++) {

          MergeNet(allComps[type][numbComp],
                   allComps[ActiveState.CONNECTOR][connNumb]);

        }
      }
    }
```

```
//////////////////////////////////////
   /*   if (probes) {
        for (int numb = 0;
        numb < Array.getLength(allComps[ActiveState.CURRENTCOMP]) - 1; numb++) {
         if (! ( (CurrentComp) allComps[ActiveState.CURRENTCOMP][numb]).
             isController()) {
           for (int type = 0; type < componentsCount - 1; type++) {
            for (int numbComp = 0;
                numbComp < Array.getLength(allComps[type]) - 1;
                numbComp++) {
              MergeNet(allComps[type][numbComp],
                    allComps[ActiveState.CURRENTCOMP][numb]);
            }
           }
         }
        }
      }*/
//////////////////////////////////////

   for (int type = 0; type < componentsCount; type++) {
     for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
       int numbNodes = allComps[type][numb].getNumNodes();
       for (int n = 0; n < numbNodes; n++) {
         int[] node = allComps[type][numb].getNode(n).getPosition();

       }

     }
   }

   for (int type = 0; type < componentsCount; type++) {
     for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
       int numbNodes = allComps[type][numb].getNumNodes();
       for (int n = 0; n < numbNodes; n++) {
         int[] node = allComps[type][numb].getNode(n).getEquivPos();

       }

     }

   }

   assignNodes();
  }
```

```
//*************************************************************************/
/

  public void assignNodes() {
    boolean Netfound = false;

    int index = 0;

    for (int count = 0;
        count < Array.getLength(allComps[ActiveState.GROUND]) - 1; count++) {
      int[] pos = allComps[ActiveState.GROUND][count].getNode(0).getEquivPos();
      Point point = new Point(pos[0], pos[1]);
      allComps[ActiveState.GROUND][count].getNode(0).strName = "0";
      netListNodes.addElement(new NetNode(point, "0", index));
    }
    index++;
    for (int type = 0; type < componentsCount - 1; type++) {
      if (type != ActiveState.GROUND) {
        for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
          int numbNodes = allComps[type][numb].getNumNodes();
          for (int n = 0; n < numbNodes; n++) {
            int[] node = allComps[type][numb].getNode(n).getEquivPos();
            Point p = new Point(node[0], node[1]);
            for (int i = 0; i < netListNodes.size(); i++) {

              if (p.equals( ( (NetNode) netListNodes.elementAt(i)).pos)) {

                if ( ( (NetNode) netListNodes.elementAt(i)).index == 0) {
                  allComps[type][numb].getNode(n).strName = "0";

                }
                else {
                  allComps[type][numb].getNode(n).strName = "N_" +
                      ( (NetNode) netListNodes.elementAt(i)).index;
                }
                Netfound = true;
                ( (NetNode) netListNodes.elementAt(i)).Increment();
              }
            }
            if (!Netfound) {
              netListNodes.addElement(new NetNode(p, "N_" + index, index));
              allComps[type][numb].getNode(n).strName = "N_" + index;
              index++;

            }
```

```
          Netfound = false;
        }

      }
    }
  }

  /*
   System.out.println("NETLIST TABLE");
   for(int k=0;k<Array.getLength(NetNames);k++)
   {
   System.out.println(NetNames[k]);
   }
   System.out.println("NET NODES");
   for(int type=0;type<componentsCount-1;type++)
   {
   for(int numb=0;numb<Array.getLength(allComps[type])-1;numb++)
   {
   int numbNodes=allComps[type][numb].getNumNodes();
   for(int n=0;n<numbNodes;n++)
   {
   System.out.println(allComps[type][numb].getNode(n).strName);
   int[] node=allComps[type][numb].getNode(n).getEquivPos();
   System.out.println("("+node[0]+","+node[1]+")");
   }
   System.out.println("<<<<<<<<<>>>>>>>>>");
   }
   }
   System.out.println(Array.getLength(NetNames));
   System.out.println(Array.getLength(NetPoints));
   */

//*************************************************************************/
/

  }

//*************************************************************************/
/
  private void MergeNet(Components comp, Components conn) {
    int[] node1 = conn.getNode(0).getEquivPos();
    int[] node2 = conn.getNode(1).getEquivPos();

    int numbNodes = comp.getNumNodes();
    for (int n = 0; n < numbNodes; n++) {
```

```
      int[] node = comp.getNode(n).getEquivPos();
      if (node[0] == node1[0] && node[1] == node1[1]) {
        comp.getNode(n).setEquivPos(node2);
      }
    }

  }

////////////////////////////////////////////////////////////////////////////
  public void saveSchematicToLocal() {
    //savePreviousPage();
    File openedFile = null;
    if (!ActiveState.pageSaved) {
      save = true;
      JFileChooser fc = new JFileChooser(new File("/"));
      fc.setSize(300, 400);
      fc.setEnabled(true);
      fc.addChoosableFileFilter(new MyFilter());

      //fc.addActionListener(new ActionListener() {
      // public void actionPerformed(ActionEvent ae) {
      // fc_action_performed(ae);
      //}
      //});
      int returnValue = fc.showSaveDialog(this.getContentPane());

      switch (returnValue) {
        case JFileChooser.APPROVE_OPTION:
          File loadedFile = fc.getSelectedFile();
          openedFile = null;
          if (loadedFile.getName().indexOf(".sch") == -1) {
            openedFile = new File(loadedFile.getPath() + ".sch");
          }
          else {
            openedFile = new File(loadedFile.getPath());

          }
          ActiveState.pageFile = openedFile;
          ActiveState.pageSaved = true;

          try {
            try {
              FileOutputStream FOS = new FileOutputStream(openedFile);
              ObjectOutputStream OOS = new ObjectOutputStream(FOS);
```

```
          WorkPlace page = new WorkPlace(canvasWidth, canvasHeight,
                                         canvasBackgroundColor,
                                         canvasForegroundColor, "page",
                                         pageTitle);
          page.saveComponents(allComps, drawComps);
          OOS.writeObject(page);

         }
         catch (NotSerializableException ne) {}
        }
        catch (Exception e) {
         System.out.println(e);
        }

        case JFileChooser.CANCEL_OPTION:
         System.out.println("CANCELED");
         break;
    }
    ;
  }
  else {

    openedFile = ActiveState.pageFile;
    try {
      try {
        FileOutputStream FOS = new FileOutputStream(openedFile);
        ObjectOutputStream OOS = new ObjectOutputStream(FOS);

        WorkPlace page = new WorkPlace(canvasWidth, canvasHeight,
                                       canvasBackgroundColor,
                                       canvasForegroundColor, "page",
                                       pageTitle);
        page.saveComponents(allComps, drawComps);
        OOS.writeObject(page);

      }
      catch (NotSerializableException ne) {}
    }
    catch (Exception e) {
      System.out.println(e);
    }

  }

}
```

```
////////////////////////////////////////////////////////////////////////////
  public void openSchematic() {
    try {
      try {
        FileInputStream FIS = new FileInputStream("schematic.sim");
        ObjectInputStream OIS = new ObjectInputStream(FIS);
        try {
          allComps = (Components[][]) OIS.readObject();
        }
        catch (ClassNotFoundException cnfe) {
          System.out.println(cnfe);
        }
      }
      catch (FileNotFoundException fnfe) {
        System.out.println(fnfe);
      }
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void splitLines(Components comp) {
///////SPLITING LINES ALGORITHM//////////////////////////////
    int[][] node = new int[2][2];
    int[] compn1 = comp.getNode(0).getPosition();
    int[] compn2 = comp.getNode(1).getPosition();
    int[] points = {
       compn1[0], compn1[1], compn2[0], compn2[1]};
    node[0][0] = points[0];
    node[0][1] = points[1];
    node[1][0] = points[2];
    node[1][1] = points[3];
    int[] pts = new int[4];
    ActiveState.deletedLines = (Components[]) cutArray(ActiveState.deletedLines);
    ActiveState.splitedLines = (Components[]) cutArray(ActiveState.splitedLines);

    //int maxnumb = Array.getLength(allComps[ActiveState.CONNECTOR]) - 2;
    for (int numb = 0;
        numb < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; numb++) {
      if (!allComps[ActiveState.CONNECTOR][numb].equals(comp)) {
        for (int i = 0; i < 2; i++) {
          int[] point1 = allComps[ActiveState.CONNECTOR][numb].getNode(0).
```

```
    getPosition();
int[] point2 = allComps[ActiveState.CONNECTOR][numb].getNode(1).
    getPosition();
pts[0] = point1[0];
pts[1] = point1[1];
pts[2] = point2[0];
pts[3] = point2[1];
int a = node[i][0] - pts[0];
int b = node[i][0] - pts[2];

double Y1 = node[i][1] - pts[1];
double X1 = node[i][0] - pts[0];
double Y2 = node[i][1] - pts[3];
double X2 = node[i][0] - pts[2];
double m1 = 0, m2 = 0;
if (X2 != 0) {
  m1 = Y2 / X2;
}
if (X1 != 0) {
  m2 = Y1 / X1;
}
double lineLength = Math.sqrt(Math.pow(pts[1] - pts[3], 2) +
                        Math.pow(pts[0] - pts[2], 2));
double d1 = Math.sqrt(Math.pow(node[i][0] - pts[0], 2) +
                    Math.pow(node[i][1] - pts[1], 2));
double d2 = Math.sqrt(Math.pow(node[i][0] - pts[2], 2) +
                    Math.pow(node[i][1] - pts[3], 2));

if ( (m1 == m2 || (X1 == 0 && X2 == 0)) && (lineLength == d1 + d2)) {
  int len = Array.getLength(ActiveState.splitedLines);
  ActiveState.splitedLines[len -
      1] = ActiveState.createNew(ActiveState.CONNECTOR);
  int[] p1 = {
      pts[0], pts[1], node[i][0], node[i][1]};
  int[] n1 = {
      pts[0], pts[1]};
  int[] n2 = {
      node[i][0], node[i][1]};

  ActiveState.splitedLines[len - 1].getNode(0).setPosition(n1);
  ActiveState.splitedLines[len - 1].getNode(1).setPosition(n2);

  ActiveState.splitedLines[len - 1].setPoints(p1);
  ActiveState.splitedLines[len - 1].setNodes();
  ActiveState.splitedLines[len - 1].setIndex(len - 1);
```

```
        ActiveState.splitedLines = (Components[]) growArray(ActiveState.
            splitedLines);
        ActiveState.splitedLines[len] = ActiveState.createNew(ActiveState.
            CONNECTOR);
        int[] p2 = {
            pts[2], pts[3], node[i][0], node[i][1]};
        int[] l1 = {
            pts[2], pts[3]};
        int[] l2 = {
            node[i][0], node[i][1]};

        ActiveState.splitedLines[len].getNode(0).setPosition(l1);
        ActiveState.splitedLines[len].getNode(1).setPosition(l2);
        ActiveState.splitedLines[len].setPoints(p2);

        ActiveState.splitedLines[len].setNodes();
        ActiveState.splitedLines[len].setIndex(len);
        ActiveState.splitedLines = (Components[]) growArray(ActiveState.
            splitedLines);
        ActiveState.deletedLines[Array.getLength(ActiveState.deletedLines) -
            1] =
            allComps[ActiveState.CONNECTOR][numb];
        ActiveState.deletedLines = (Components[]) growArray(ActiveState.
            deletedLines);
      }

     }
    }
   }
////////////////////////////////////////////////////////////////////////////

   Vector POINTS = new Vector();
   pts[0] = compn1[0];
   pts[1] = compn1[1];
   pts[2] = compn2[0];
   pts[3] = compn2[1];
   POINTS.addElement(compn1);
   for (int numb = 0;
       numb < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; numb++) {
    if (!allComps[ActiveState.CONNECTOR][numb].equals(comp)) {
     int[] n1 = allComps[ActiveState.CONNECTOR][numb].getNode(0).getPosition();
     int[] n2 = allComps[ActiveState.CONNECTOR][numb].getNode(1).getPosition();
     node[0][0] = n1[0];
     node[0][1] = n1[1];
     node[1][0] = n2[0];
```

```
node[1][1] = n2[1];

for (int i = 0; i < 2; i++) {
  int a = node[i][0] - pts[0];
  int b = node[i][0] - pts[2];

  double Y1 = node[i][1] - pts[1];
  double X1 = node[i][0] - pts[0];
  double Y2 = node[i][1] - pts[3];
  double X2 = node[i][0] - pts[2];
  double m1 = 0, m2 = 0;
  if (X2 != 0) {
    m1 = Y2 / X2;
  }
  if (X1 != 0) {
    m2 = Y1 / X1;
  }
  double lineLength = Math.sqrt(Math.pow(pts[1] - pts[3], 2) +
                          Math.pow(pts[0] - pts[2], 2));
  double d1 = Math.sqrt(Math.pow(node[i][0] - pts[0], 2) +
                    Math.pow(node[i][1] - pts[1], 2));
  double d2 = Math.sqrt(Math.pow(node[i][0] - pts[2], 2) +
                    Math.pow(node[i][1] - pts[3], 2));

  if ( (m1 == m2 || (X1 == 0 && X2 == 0)) && (lineLength == d1 + d2)) {
    int[] p = new int[2];
    p[0] = node[i][0];
    p[1] = node[i][1];
    POINTS.addElement(p);

    /*
        int len=Array.getLength(ActiveState.splitedLines);
        ActiveState.splitedLines[len-1]=ActiveState.createNew(ActiveState.CONNECTOR);
        int[] p1 = {pts[0], pts[1], node[i][0], node[i][1]};
        int[] k1={pts[0],pts[1]};
        int[] k2={node[i][0],node[i][1]};
        ActiveState.splitedLines[len-1].getNode(0).setPosition(k1);
        ActiveState.splitedLines[len-1].getNode(1).setPosition(k2);
        ActiveState.splitedLines[len - 1].setPoints(p1);
        ActiveState.splitedLines[len - 1].setNodes();
        ActiveState.splitedLines[len-1].setIndex(len-1);
        ActiveState.splitedLines = (Components[]) growArray(ActiveState.splitedLines);
        ActiveState.splitedLines[len] = ActiveState.createNew(ActiveState.CONNECTOR);
        int[] p2 = {pts[2], pts[3], node[i][0], node[i][1]};
        int[] l1={pts[2],pts[3]};
```

```
            int[] l2={node[i][0],node[i][1]};
            ActiveState.splitedLines[len].getNode(0).setPosition(l1);
            ActiveState.splitedLines[len].getNode(1).setPosition(l2);
            ActiveState.splitedLines[len].setPoints(p2);
          (ActiveState.deletedLines)-1]=comp;
            //allComps[ActiveState.CONNECTOR][comp.getComponentType()];
            ActiveState.deletedLines=(Components[])  growArray(ActiveState.deletedLines);
          */
        }

       }

     }
   }
   POINTS.addElement(compn2);
   if (POINTS.size() > 2) {
     for (int i = 0; i < POINTS.size() - 1; i++) {
       int n1[] = new int[2];
       int n2[] = new int[2];

       int[] p = (int[]) POINTS.get(i);

       n1 = (int[]) POINTS.get(i);
       n2 = (int[]) POINTS.get(i + 1);
       int len = Array.getLength(ActiveState.splitedLines);
       ActiveState.splitedLines[len -
          1] = ActiveState.createNew(ActiveState.CONNECTOR);
       ActiveState.splitedLines[len - 1].getNode(0).setPosition(n1);
       ActiveState.splitedLines[len - 1].getNode(1).setPosition(n2);
       int[] p1 = {
          n1[0], n1[1], n2[0], n2[1]};
       ActiveState.splitedLines[len - 1].setPoints(p1);
       ActiveState.splitedLines[len - 1].setNodes();
       ActiveState.splitedLines[len - 1].setIndex(len - 1);
       ActiveState.splitedLines = (Components[]) growArray(ActiveState.
          splitedLines);
     }
     ActiveState.deletedLines[Array.getLength(ActiveState.deletedLines) -
        1] = comp;
     ActiveState.deletedLines = (Components[]) growArray(ActiveState.
        deletedLines);

   }

//.....
```

```
    for (int numb = 0; numb < Array.getLength(ActiveState.deletedLines) - 1;
        numb++) {
      for (int n = 0; n < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1;
          n++) {
        if (ActiveState.deletedLines[numb].equals(allComps[ActiveState.
                                        CONNECTOR][n])) {
          allComps[ActiveState.CONNECTOR] = (Components[]) shrinkArray(
              allComps[ActiveState.CONNECTOR], n);
        }
      }
    }

    for (int numb = 0; numb < Array.getLength(ActiveState.splitedLines) - 1;
        numb++) {
      int len = Array.getLength(allComps[ActiveState.CONNECTOR]);
      allComps[ActiveState.CONNECTOR][len - 1] = ActiveState.splitedLines[numb];
      allComps[ActiveState.CONNECTOR] =
          (Components[]) growArray(allComps[ActiveState.CONNECTOR]);
    }

    for (int i = 0; i < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; i++) {

      int[] point1 = allComps[ActiveState.CONNECTOR][i].getNode(0).getPosition();
      int[] point2 = allComps[ActiveState.CONNECTOR][i].getNode(1).getPosition();

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void saveOriginalSettings() {

    for (int type = 0; type < componentsCount; type++) {
      recordComps[undoNumb][type] = (Components[]) cutArray(recordComps[
          undoNumb][type]);
    }

    for (int type = 0; type < componentsCount; type++) {
      for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
          numbComp++) {
        int length = Array.getLength(recordComps[undoNumb][type]);
        recordComps[undoNumb][type][length - 1] = ActiveState.createNew(type);

        if (allComps[type][numbComp].getComponentType() ==
```

```
        ActiveState.CONNECTOR) {
      int[] node1 = allComps[type][numbComp].getNode(0).getPosition();
      int[] node2 = allComps[type][numbComp].getNode(1).getPosition();

      recordComps[undoNumb][type][length - 1].getNode(0).setPosition(node1);
      recordComps[undoNumb][type][length - 1].getNode(1).setPosition(node2);
    }
    else {
      recordComps[undoNumb][type][length -
          1].setPosition(allComps[type][numbComp].getPosition());
      recordComps[undoNumb][type][length -
          1].setAlignment(allComps[type][numbComp].getAlignment());

    }
    recordComps[undoNumb][type][length - 1].setNodes();
    recordComps[undoNumb][type][length -
        1].setIndex(allComps[type][numbComp].getIndex());
    recordComps[undoNumb][type][length -
        1].setComponentName(allComps[type][numbComp].getComponentName());
    if (allComps[type][numbComp].isGrouped()) {
      recordComps[undoNumb][type][length -
          1].setGroupNumb(allComps[type][numbComp].getGroupNumb());
    }
    recordComps[undoNumb][type] = (Components[]) growArray(recordComps[
        undoNumb][type]);

   }
  }

  for (int type = 0; type < componentsCount; type++) {
   for (int numbComp = 0;
       numbComp < Array.getLength(ActiveState.origComps[type]) - 1;
       numbComp++) {

   }
  }
  undoStack.push(recordComps[undoNumb]);
  if (undoNumb < 9) {
   undoNumb++;
  }
  else {
   undoNumb = 0;
  }

}
```

```
////////////////////////////////////////////////////////////////////////////
  private void restoreOriginalSettings() {
    if (undoStack.top > 0) {

      for (int type = 0; type < componentsCount; type++) {
        allComps[type] = (Components[]) cutArray(allComps[type]);
//ActiveState.origComps[type]=(Components[]) cutArray(ActiveState.origComps[]);
      }

//Components[][] restoredComps=new Components[componentsCount][];
//for(int type=0;type<componetsCount;type++)
//{
//restoredComps=new
//}

      //      =undoStack.pop();

      allComps = undoStack.pop();
//for(int type=0;type<componentsCount;type++)
//{
//allComps[type]=(Components[]) copyComponentsArray(restoredComps);
//}

      redrawGridCanvas();
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void cutComps() {
    if (ActiveState.active) {
      saveOriginalSettings();
      ActiveState.copiedComp = ActiveState.activeComp;
      ActiveState.singleCopied = true;
      deleteComponent(true);
    }
    else if (ActiveState.multiActive) {
      saveOriginalSettings();
      ActiveState.undoEnabled = true;

      for (int type = 0; type < componentsCount; type++) {
        for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
          if (allComps[type][numb].getActiveState()) {

            int length = Array.getLength(ActiveState.copiedComps[type]);
```

```
          ActiveState.copiedComps[type][length - 1] = allComps[type][numb];
          ActiveState.copiedComps[type] = (Components[]) growArray(
             ActiveState.copiedComps[type]);

        }
       }
      }

      deleteComponent(false);

      ActiveState.multipleCopied = true;
      ActiveState.multiActive = false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void copyComps() {

    if (ActiveState.active) {
      ActiveState.copiedComp = ActiveState.activeComp;
      ActiveState.singleCopied = true;

    }
    else if (ActiveState.multiActive) {

      for (int type = 0; type < componentsCount; type++) {
        for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
          if (allComps[type][numb].getActiveState()) {

            int length = Array.getLength(ActiveState.copiedComps[type]);
            ActiveState.copiedComps[type][length - 1] = allComps[type][numb];
            ActiveState.copiedComps[type] = (Components[]) growArray(
               ActiveState.copiedComps[type]);

          }
        }
      }

      ActiveState.multipleCopied = true;
      //ActiveState.multiActive = false;
    }
    /*else {
      InfoMessage noSelection = new InfoMessage("No Component Selected");
      noSelection.show();
```

```
        }*/
    }

////////////////////////////////////////////////////////////////////////////
    private void pasteComps() {
      saveOriginalSettings();
      ActiveState.undoEnabled = true;

      if (ActiveState.singleCopied) {
        int type = ActiveState.copiedComp.getComponentType();
        int length = Array.getLength(allComps[type]);
        int index = length - 1;

        int[] pos = {
            xyClicked[0], xyClicked[1]};
        allComps[type][length - 1] = ActiveState.createNew(type);
        if (ActiveState.copiedComp.getComponentType() == ActiveState.CONNECTOR) {
          int[] n1 = ActiveState.copiedComp.getNode(0).getPosition();
          int[] n2 = ActiveState.copiedComp.getNode(1).getPosition();
          int xdiff = pos[0] - n1[0];
          int ydiff = pos[1] - n1[1];
          int[] newn1 = {
              n1[0] + xdiff, n1[1] + ydiff};
          int[] newn2 = {
              n2[0] + xdiff, n2[1] + ydiff};
          newn1 = calculateAppPoint(newn1);
          newn2 = calculateAppPoint(newn2);
          allComps[type][length - 1].getNode(0).setPosition(newn1);
          allComps[type][length - 1].getNode(1).setPosition(newn2);
        }
        else {
          pos = calculateAppPoint(pos);
          allComps[type][length - 1].setPosition(pos);
        }

        allComps[type][length -
            1].setAlignment(ActiveState.copiedComp.getAlignment());
        allComps[type][length - 1].setNodes();
        allComps[type][length - 1].setIndex(length - 1);
        allComps[type][length -
            1].setComponentName(ActiveState.copiedComp.getName() + index);

        allComps[type] = (Components[]) growArray(allComps[type]);

        ActiveState.copiedComp = null;
```

```
  ActiveState.singleCopied = false;

}
else if (ActiveState.multipleCopied) {

  boolean flag = true;
  int[] distance = new int[2];
  int[] pos = {
     xyClicked[0], xyClicked[1]};

  for (int type = 0; type < componentsCount; type++) {
    for (int numb = 0;
         numb < Array.getLength(ActiveState.copiedComps[type]) - 1; numb++)

    {

      int length = Array.getLength(allComps[type]);
      int index = length - 1;
      int[] compPos = ActiveState.copiedComps[type][numb].getPosition();

      if (flag) {
        if (ActiveState.copiedComps[type][numb].getComponentType() ==
            ActiveState.CONNECTOR) {
          distance = ActiveState.copiedComps[type][numb].getNode(0).
              getPosition();
        }
        else {
          distance = ActiveState.copiedComps[type][numb].getPosition();

        }
        flag = false;
      }
      int align = ActiveState.copiedComps[type][numb].getAlignment();
      int xdiff = pos[0] - distance[0];
      int ydiff = pos[1] - distance[1];
      int[] newPos = {
         compPos[0] + xdiff, compPos[1] + ydiff};
      allComps[type][length - 1] = ActiveState.createNew(type);

      allComps[type][length - 1].setIndex(length - 1);
      if (ActiveState.copiedComps[type][numb].getComponentType() ==
          ActiveState.CONNECTOR) {
        int[] n1 = ActiveState.copiedComps[type][numb].getNode(0).
            getPosition();
        int[] newn1 = {
```

```
            n1[0] + xdiff, n1[1] + ydiff};
          newn1 = calculateAppPoint(newn1);
          allComps[type][length - 1].getNode(0).setPosition(newn1);
          int[] n2 = ActiveState.copiedComps[type][numb].getNode(1).
             getPosition();
          int[] newn2 = {
             n2[0] + xdiff, n2[1] + ydiff};
          newn2 = calculateAppPoint(newn2);
          allComps[type][length - 1].getNode(1).setPosition(newn2);
        }
        else {
          allComps[type][length - 1].setPosition(newPos);
          allComps[type][length - 1].setAlignment(align);
        }

        allComps[type][length - 1].setNodes();
        allComps[type][length -
           1].setComponentName(ActiveState.copiedComps[type][numb].getName() +
                       index);
        allComps[type] = (Components[]) growArray(allComps[type]);

      }
    }

    ActiveState.multipleCopied = false;
    for (int type = 0; type < componentsCount; type++) {
      ActiveState.copiedComps[type] = (Components[]) cutArray(ActiveState.
         copiedComps[type]);
    }
  }
  for (int i = 0; i < Array.getLength(allComps[0]) - 1; i++) {

  }

  redrawGridCanvas();

 }

////////////////////////////////////////////////////////////////////////////
 private void runSimulation() {

  try {
    Socket simulateClient = new Socket("localhost", 3300);
    ObjectOutputStream OOS = new ObjectOutputStream(simulateClient.
       getOutputStream());
```

```
      OOS.writeObject(cirMLString);
      OOS.flush();
      ObjectInputStream OIS = new ObjectInputStream(simulateClient.
          getInputStream());
      try {
        outputResult = (String) OIS.readObject();

        graphicalResultStr = (String) OIS.readObject();

      }
      catch (ClassNotFoundException cnfe) {
        System.out.println(cnfe);
      }

    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    outputResultFrame = new SimulationResultFrame();
    outputResultFrame.result.setText(outputResult);
    outputResultFrame.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void updateStatusBar(MouseEvent me, boolean exit) {
    if (ActiveState.showStatusBar) {
      int[] pos = {
          me.getX(), me.getY()};

      Graphics g = statusPanel.getGraphics();
      Image img = getImage(getDocumentBase(), "ECS/images/statusbar.jpg");
      g.drawImage(img, 0, 0, statusPanel);

      if (!exit) {
        g.setColor(Color.BLUE);
        g.drawString("Status bar", 0, 14);
        g.setColor(Color.BLACK);
        g.drawString("Selected Items:" + ActiveState.numbOfSelectedItems, 120,
                     14);

        g.drawString("Page#" + ActiveState.currentPageNumber, 70, 14);

        g.drawString("Scale:100%", 220, 14);
```

```
      g.drawString("Current Position:[" + "X=" + pos[0] + ", Y=" + pos[1] +
                "]", 310,
                14);

      g.setColor(Color.WHITE);
      if (ActiveState.moving) {
        g.drawString("Mouse Moving...", 490, 14);
      }
      else if (ActiveState.pressing) {
        g.drawString("Mouse Pressing...", 490, 14);
      }
      else if (ActiveState.releasing) {
        g.drawString("Mouse Releasing...", 490, 14);
      }
      else if (ActiveState.dragging) {
        g.drawString("Mouse Dragging...", 490, 14);
      }
    }
  }
}

////////////////////////////////////////////////////////////////////////////
  private void activateElementsPanel() {
    passivePanel.setBackground(new Color(71, 127, 255));
    independentPanel.setBackground(new Color(71, 127, 255));
    dependentPanel.setBackground(new Color(71, 127, 255));
    connectionsPanel.setBackground(new Color(71, 127, 255));
    transformersPanel.setBackground(new Color(71, 127, 255));

    passivePanel.setFocusable(false);
    independentPanel.setFocusable(false);
    dependentPanel.setFocusable(false);
    connectionsPanel.setFocusable(false);
    transformersPanel.setFocusable(false);
    switch (ActiveState.activePanel) {
      case 0:
        passivePanel.setFocusable(true);
        passivePanel.setBackground(new Color(43, 108, 255));
        break;
      case 1:
        independentPanel.setFocusable(true);
        independentPanel.setBackground(new Color(43, 108, 255));
        break;
      case 2:
        dependentPanel.setFocusable(true);
```

```
      dependentPanel.setBackground(new Color(43, 108, 255));
      break;
    case 3:
      connectionsPanel.setFocusable(true);
      connectionsPanel.setBackground(new Color(43, 108, 255));
      break;
    case 4:
      transformersPanel.setFocusable(true);
      transformersPanel.setBackground(new Color(55, 117, 255));

      break;

  }

 }

////////////////////////////////////////////////////////////////////////////
  private void drawpagesExplorerToolbar() {

   pagesExplorerScrollBar = new JScrollPane();

   pagesExplorerToolbar = new JToolBar(JToolBar.VERTICAL);
   pagesExplorerToolbar.setFloatable(false);
   pagesExplorerToolbar.setBackground(new Color(178, 197, 251));
   pagesExplorerToolbar.setLayout(null);

   Image img = getImage(getDocumentBase(), "ECS/images/leftbar.jpg");
   JLabel lbl = new JLabel(new ImageIcon(img));

   img = ActiveState.btnsEntered[ActiveState.NEW1];
   new1Btn = new JButton(new ImageIcon(img));
   img = ActiveState.btnsExited[ActiveState.NEW2];
   new2Btn = new JButton(new ImageIcon(img));
   img = ActiveState.btnsExited[ActiveState.NEW3];
   new3Btn = new JButton(new ImageIcon(img));
   img = ActiveState.btnsExited[ActiveState.NEW4];
   new4Btn = new JButton(new ImageIcon(img));
   img = ActiveState.btnsExited[ActiveState.NEW5];
   new5Btn = new JButton(new ImageIcon(img));

   img = getImage(getDocumentBase(), "ECS/images/horizSep.jpg");
   JButton sep1 = new JButton(new ImageIcon(img));
   JButton sep2 = new JButton(new ImageIcon(img));
   JButton sep3 = new JButton(new ImageIcon(img));
```

```
new1Btn.setBackground(new Color(178, 197, 251));
new2Btn.setBackground(new Color(178, 197, 251));
new3Btn.setBackground(new Color(178, 197, 251));
new4Btn.setBackground(new Color(178, 197, 251));
new5Btn.setBackground(new Color(178, 197, 251));

sep1.setBackground(new Color(178, 197, 251));
sep2.setBackground(new Color(178, 197, 251));
sep3.setBackground(new Color(178, 197, 251));

new1Btn.setBorder(null);
new2Btn.setBorder(null);
new3Btn.setBorder(null);
new4Btn.setBorder(null);
new5Btn.setBorder(null);

new1Btn.setActionCommand("new1Btn");
new2Btn.setActionCommand("new2Btn");
new3Btn.setActionCommand("new3Btn");
new4Btn.setActionCommand("new4Btn");
new5Btn.setActionCommand("new5Btn");

new1Btn.setVisible(true);
new2Btn.setVisible(false);
new3Btn.setVisible(false);
new4Btn.setVisible(false);
new5Btn.setVisible(false);

sep1.setBorder(null);
sep2.setBorder(null);
sep3.setBorder(null);

sep1.setBounds(2, 5, 22, 5);
new1Btn.setBounds(3, 20, 22, 23);
new2Btn.setBounds(3, 55, 22, 23);
new3Btn.setBounds(3, 90, 22, 23);
new4Btn.setBounds(3, 125, 22, 23);
new5Btn.setBounds(3, 160, 22, 23);
sep2.setBounds(2, 10, 22, 5);
sep3.setBounds(2, 195, 22, 5);

pagesExplorerToolbar.add(sep1);
pagesExplorerToolbar.add(sep2);
//pagesExplorerToolbar.add(newBtn);
pagesExplorerToolbar.add(new1Btn);
```

```
    pagesExplorerToolbar.add(new2Btn);
    pagesExplorerToolbar.add(new3Btn);
    pagesExplorerToolbar.add(new4Btn);
    pagesExplorerToolbar.add(new5Btn);
    pagesExplorerToolbar.add(sep3);

    //pagesExplorerToolbar.setSize(89,320);
    //pagesExplorerScrollBar.getViewport().add(pagesExplorerToolbar, null);
    //pagesExplorerScrollBar.setBorder(null);
    pagesExplorerToolbar.setBounds(2, 35, 28, 200);
    //pagesExplorerScrollBar.getVerticalScrollBar().setLocation(0,0);
    this.getContentPane().add(pagesExplorerToolbar);
    //pagesExplorerScrollBar.getVerticalScrollBar().setLocation(0,0);

//pagesExplorerScrollBar.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_N
EVER);

//pagesExplorerScrollBar.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLB
AR_NEVER);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawCanvasToolbar() {

    canvasToolbar = new JToolBar();
    canvasToolbar.setFloatable(false);
    canvasToolbar.setBackground(new Color(178, 197, 251));
    canvasToolbar.setVisible(false);

    Image img = ActiveState.btnsExited[ActiveState.CANVASCOLORS];
    canvasColorsBtn = new JButton(new ImageIcon(img));
    canvasColorsBtn.setActionCommand("canvasColorsBtn");
    canvasColorsBtn.setBackground(new Color(55, 115, 247));
    canvasColorsBtn.setToolTipText("Canvas Colors");
    canvasColorsBtn.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.CANVASSIZE];
    canvasSizeBtn = new JButton(new ImageIcon(img));
    canvasSizeBtn.setActionCommand("canvasSizeBtn");
    canvasSizeBtn.setBackground(new Color(55, 115, 247));
    canvasSizeBtn.setToolTipText("Canvas Size");
    canvasSizeBtn.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.GOTO];
```

```
goToBtn = new JButton(new ImageIcon(img));
goToBtn.setActionCommand("goToBtn");
goToBtn.setBackground(new Color(55, 115, 247));
goToBtn.setToolTipText("Go To...");
goToBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ZOOMIN];
zoomInBtn = new JButton(new ImageIcon(img));
zoomInBtn.setActionCommand("zoomInBtn");
zoomInBtn.setBackground(new Color(55, 115, 247));
zoomInBtn.setToolTipText("Zooming in");
zoomInBtn.setBackground(new Color(179, 186, 241));
zoomInBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ZOOMOUT];
zoomOutBtn = new JButton(new ImageIcon(img));
zoomOutBtn.setActionCommand("zoomOutBtn");
zoomOutBtn.setBackground(new Color(55, 115, 247));
zoomOutBtn.setToolTipText("Zooming out");
zoomOutBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.HAND];
handBtn = new JButton(new ImageIcon(img));
handBtn.setActionCommand("handBtn");
handBtn.setBackground(new Color(55, 115, 247));
handBtn.setToolTipText("Scroll Hand");
handBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.DCURSOR];
dCursorBtn = new JButton(new ImageIcon(img));
dCursorBtn.setActionCommand("dCursorBtn");
dCursorBtn.setBackground(new Color(55, 115, 247));
dCursorBtn.setToolTipText("Default Cursor");
dCursorBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.FITZOOM];
fitZoomBtn = new JButton(new ImageIcon(img));
fitZoomBtn.setActionCommand("fitZoomBtn");
fitZoomBtn.setBackground(new Color(55, 115, 247));
fitZoomBtn.setToolTipText("Fit Zoom");
fitZoomBtn.setBorder(null);

img = ActiveState.diffImages[ActiveState.SEPARATOR];
JButton sep1 = new JButton(new ImageIcon(img));
JButton sep2 = new JButton(new ImageIcon(img));
```

```
JButton sep3 = new JButton(new ImageIcon(img));
JButton sep4 = new JButton(new ImageIcon(img));

sep1.setBackground(new Color(178, 197, 251));
sep2.setBackground(new Color(178, 197, 251));
sep3.setBackground(new Color(178, 197, 251));
sep4.setBackground(new Color(178, 197, 251));

sep1.setBorder(null);
sep2.setBorder(null);
sep3.setBorder(null);
sep4.setBorder(null);

canvasToolbar.add(sep1);
canvasToolbar.add(sep2);

canvasToolbar.addSeparator();

canvasToolbar.add(canvasColorsBtn);
canvasToolbar.addSeparator();
canvasToolbar.add(canvasSizeBtn);
canvasToolbar.addSeparator();
canvasToolbar.add(goToBtn);
//canvasToolbar.addSeparator();
//canvasToolbar.add(zoomInBtn);
//canvasToolbar.addSeparator();
//canvasToolbar.add(fitZoomBtn);
//canvasToolbar.addSeparator();
//canvasToolbar.add(zoomOutBtn);
canvasToolbar.addSeparator();
canvasToolbar.add(handBtn);
canvasToolbar.addSeparator();
canvasToolbar.add(dCursorBtn);
canvasToolbar.addSeparator();
canvasToolbar.addSeparator();
canvasToolbar.addSeparator();
canvasToolbar.addSeparator();
canvasToolbar.addSeparator();
canvasToolbar.addSeparator();
//canvasToolbar.addSeparator();
//canvasToolbar.addSeparator();
canvasToolbar.add(sep4);

canvasToolbar.setBounds(418, 35, 227, 25);
//canvasToolbar.setBounds(418, 35, 253, 25);
```

```
    this.getContentPane().add(canvasToolbar);

  }

////////////////////////////////////////////////////////////////////////////
  private void drawPaintingBar() {
    paintingToolbar = new JToolBar(JToolBar.VERTICAL);
    paintingToolbar.setFloatable(false);
    paintingToolbar.setBackground(new Color(178, 197, 251));

    String k = "shad";

    Image img = getImage(getDocumentBase(), "ECS/images/horizSep.jpg");
    JButton sep1 = new JButton(new ImageIcon(img));
    JButton sep2 = new JButton(new ImageIcon(img));
    JButton sep3 = new JButton(new ImageIcon(img));
    sep1.setBackground(new Color(178, 197, 251));
    sep2.setBackground(new Color(178, 197, 251));
    sep3.setBackground(new Color(178, 197, 251));
    sep1.setBorder(null);
    sep2.setBorder(null);
    sep3.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.LINE];
    lineBtn = new JButton(new ImageIcon(img));
    lineBtn.setActionCommand("lineBtn");
    lineBtn.setBackground(new Color(55, 115, 247));
    lineBtn.setToolTipText("Draw Line");
    lineBtn.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.TEXT];
    insertTextBtn = new JButton(new ImageIcon(img));
    insertTextBtn.setActionCommand("insertTextBtn");
    insertTextBtn.setBackground(new Color(55, 115, 247));
    insertTextBtn.setToolTipText("Insert Text");
    insertTextBtn.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.FILLCOLOR];
    fillColorBtn = new JButton(new ImageIcon(img));
    fillColorBtn.setActionCommand("fillColorBtn");
    fillColorBtn.setBackground(new Color(55, 115, 247));
    fillColorBtn.setToolTipText("Fill Color");
    fillColorBtn.setBorder(null);

    img = ActiveState.btnsExited[ActiveState.LINECOLOR];
```

```
lineColorBtn = new JButton(new ImageIcon(img));
lineColorBtn.setActionCommand("lineColorBtn");
lineColorBtn.setBackground(new Color(55, 115, 247));
lineColorBtn.setToolTipText("Line Color");
lineColorBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.TEXTCOLOR];
textColorBtn = new JButton(new ImageIcon(img));
textColorBtn.setActionCommand("textColorBtn");
textColorBtn.setBackground(new Color(55, 115, 247));
textColorBtn.setToolTipText("Text Color");
textColorBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.DCURSOR];
dftCursorBtn = new JButton(new ImageIcon(img));
dftCursorBtn.setActionCommand("dftCursorBtn");
dftCursorBtn.setBackground(new Color(55, 115, 247));
dftCursorBtn.setToolTipText("Default Cursor");
dftCursorBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.SQUARE];
rectangleBtn = new JButton(new ImageIcon(img));
rectangleBtn.setActionCommand("rectangleBtn");
rectangleBtn.setBackground(new Color(55, 115, 247));
rectangleBtn.setToolTipText("Draw Rectangle");
rectangleBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ROUNDRECTANGLE];
roundRectangleBtn = new JButton(new ImageIcon(img));
roundRectangleBtn.setActionCommand("roundRectangleBtn");
roundRectangleBtn.setBackground(new Color(55, 115, 247));
roundRectangleBtn.setToolTipText("Draw Round Rectangle");
roundRectangleBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.CIRCLE];
circleBtn = new JButton(new ImageIcon(img));
circleBtn.setActionCommand("circleBtn");
circleBtn.setBackground(new Color(55, 115, 247));
circleBtn.setToolTipText("Draw Circle");
circleBtn.setBorder(null);

img = ActiveState.btnsExited[ActiveState.ARC];
arcBtn = new JButton(new ImageIcon(img));
arcBtn.setActionCommand("arcBtn");
arcBtn.setBackground(new Color(55, 115, 247));
```

```
    arcBtn.setToolTipText("Draw Arc");
    arcBtn.setBorder(null);

    paintingToolbar.add(sep1);
    paintingToolbar.add(sep2);
    paintingToolbar.addSeparator();
    paintingToolbar.add(fillColorBtn);
    paintingToolbar.addSeparator();
    paintingToolbar.add(lineColorBtn);
    paintingToolbar.addSeparator();
    paintingToolbar.add(textColorBtn);
    paintingToolbar.addSeparator();
    paintingToolbar.add(insertTextBtn);
    paintingToolbar.addSeparator();
    paintingToolbar.add(dftCursorBtn);
    paintingToolbar.addSeparator();
    paintingToolbar.add(lineBtn);
    paintingToolbar.addSeparator();
    paintingToolbar.add(rectangleBtn);
    paintingToolbar.addSeparator();
    paintingToolbar.add(roundRectangleBtn);
    paintingToolbar.addSeparator();
    paintingToolbar.add(circleBtn);
    paintingToolbar.addSeparator();
//leftToolBar2.add(arcBtn);
//leftToolBar2.addSeparator();
    paintingToolbar.add(sep3);

    add(paintingToolbar, 2, 250, 28, 285);

  }

////////////////////////////////////////////////////////////////////////////
  /*private void drawServerFileManagerToolbar()
   {
     serverFileManagerToolbar = new JToolBar();
     serverFileManagerToolbar.setFloatable(false);
     serverFileManagerToolbar.setBackground(new Color(178, 197, 251));
     Image img = getImage(getDocumentBase(), "ECS/images/horizSep.jpg");
     JButton sep1 = new JButton(new ImageIcon(img));
     JButton sep2 = new JButton(new ImageIcon(img));
     sep1.setBackground(new Color(178, 197, 251));
     sep2.setBackground(new Color(178, 197, 251));
     sep1.setBorder(null);
     sep2.setBorder(null);
```

```
img = ActiveState.btnsExited[ActiveState.CONNECTFM];
connectFMBtn = new JButton(new ImageIcon(img));
connectFMBtn.setActionCommand("Connect To Server");
connectFMBtn.setBackground(new Color(55, 115, 247));
connectFMBtn.setToolTipText("Draw Line");
connectFMBtn.setBorder(null);
img = ActiveState.btnsExited[ActiveState.DISCONNECTFM];
disconnectFMBtn = new JButton(new ImageIcon(img));
disconnectFMBtn.setActionCommand("Disconnect From Server");
disconnectFMBtn.setBackground(new Color(55, 115, 247));
disconnectFMBtn.setToolTipText("Disconnect From Serve");
disconnectFMBtn.setBorder(null);
img = ActiveState.btnsExited[ActiveState.UPLOADFM];
uploadFMBtn = new JButton(new ImageIcon(img));
uploadFMBtn.setActionCommand("Upload Schematic File");
uploadFMBtn.setBackground(new Color(55, 115, 247));
uploadFMBtn.setToolTipText("Upload Schematic File");
uploadFMBtn.setBorder(null);
img = ActiveState.btnsExited[ActiveState.DOWNLOADFM];
downloadFMBtn = new JButton(new ImageIcon(img));
downloadFMBtn.setActionCommand("lineBtn");
downloadFMBtn.setBackground(new Color(55, 115, 247));
downloadFMBtn.setToolTipText("Draw Line");
downloadFMBtn.setBorder(null);
img = ActiveState.btnsExited[ActiveState.MKDIRECTORYFM];
mkDirectoryFMBtn = new JButton(new ImageIcon(img));
mkDirectoryFMBtn.setActionCommand("lineBtn");
mkDirectoryFMBtn.setBackground(new Color(55, 115, 247));
mkDirectoryFMBtn.setToolTipText("Draw Line");
mkDirectoryFMBtn.setBorder(null);
img = ActiveState.btnsExited[ActiveState.RMDIRECTORYFM];
rmDirectoryFMBtn = new JButton(new ImageIcon(img));
rmDirectoryFMBtn.setActionCommand("lineBtn");
rmDirectoryFMBtn.setBackground(new Color(55, 115, 247));
rmDirectoryFMBtn.setToolTipText("Draw Line");
rmDirectoryFMBtn.setBorder(null);
img = ActiveState.btnsExited[ActiveState.RENAMEFM];
renameFMBtn = new JButton(new ImageIcon(img));
renameFMBtn.setActionCommand("lineBtn");
renameFMBtn.setBackground(new Color(55, 115, 247));
renameFMBtn.setToolTipText("Draw Line");
renameFMBtn.setBorder(null);
img = ActiveState.btnsExited[ActiveState.DELETEFM];
deleteFMBtn = new JButton(new ImageIcon(img));
deleteFMBtn.setActionCommand("lineBtn");
```

```
    deleteFMBtn.setBackground(new Color(55, 115, 247));
    deleteFMBtn.setToolTipText("Draw Line");
    deleteFMBtn.setBorder(null);
    serverFileManagerToolbar.add(sep1);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(connectFMBtn);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(disconnectFMBtn);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(uploadFMBtn);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(downloadFMBtn);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(mkDirectoryFMBtn);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(rmDirectoryFMBtn);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(renameFMBtn);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(deleteFMBtn);
    serverFileManagerToolbar.addSeparator();
    serverFileManagerToolbar.add(sep2);
    add(serverFileManagerToolbar, 320, 65, 280, 25);
  }
  */

////////////////////////////////////////////////////////////////////////////
  public void redrawGridCanvas() {
    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasForegroundColor);
    if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
    }
    drawAllComps();
    drawDrawingComps();

    if (!pageTitle.equals(new String(""))) {

      drawPageTitle();
    }
    if (ActiveState.IsNetlistNodesShown) {
      showNetListNodes();
    }
```

```
    if (ActiveState.IsNodesVoltagesShown) {
      showNodesVoltages();
    }
    if (ActiveState.IsNodesCurrentsShown) {
      showNodesCurrents();

    }

    if (ActiveState.IsNetlistNodesShown) {
      showNetListNodes();
    }
    if (ActiveState.IsNodesVoltagesShown) {
      showNodesVoltages();
    }
    if (ActiveState.IsNodesCurrentsShown) {
      showNodesCurrents();

    }
    gridCanvas.redraw();
    //savePreviousPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void deActivatePages() {
    new1Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW1]));
    new2Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW2]));
    new3Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW3]));
    new4Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW4]));
    new5Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW5]));

  }

////////////////////////////////////////////////////////////////////////////

  private void addBtnToPanel() {

    JButton shadiBtn = new JButton("SHADI");
    shadiBtn.setActionCommand("createNetlistBtn");
    shadiBtn.setBackground(new Color(179, 186, 241));
    shadiBtn.setBorder(null);
    shadiBtn.setForeground(Color.WHITE);
    shadiBtn.setFont(myFont);
    addToPanel(visitedElementsPanel, shadiBtn, 0, 0, 25, 20);
```

```
  }

////////////////////////////////////////////////////////////////////////////
  private JButton getVisitedActionButton(String com) {

    if (com.equals("Resistor")) {
      return VResistorBtn;
    }
    else if (com.equals("Capacitor")) {
      return VCapacitorBtn;
    }
    else if (com.equals("Inductor")) {
      return VInductorBtn;
    }
    else if (com.equals("PPTransformer")) {
      return VPPTransformerBtn;
    }
    else if (com.equals("PNTransformer")) {
      return VPNTransformerBtn;
    }
    else if (com.equals("NPTransformer")) {
      return VNPTransformerBtn;
    }
    else if (com.equals("NNTransformer")) {
      return VNNTransformerBtn;
    }
    else if (com.equals("DC Voltage Source")) {
      return VDCVoltageBtn;
    }
    else if (com.equals("DC Current Source")) {
      return VDCCurrentBtn;
    }
    else if (com.equals("AC Voltage Source")) {
      return VACVoltageBtn;
    }
    else if (com.equals("AC Current Source")) {
      return VACCurrentBtn;
    }
    else if (com.equals("Voltage Controlled Voltage Source")) {
      return VVCVSBtn;
    }
    else if (com.equals("Voltage Controlled Current Source")) {
      return VVCCSBtn;
    }
    else if (com.equals("Current Controlled Voltage Source")) {
```

```
      return VCCVSBtn;
    }
    else if (com.equals("Current Controlled Current Source")) {
      return VCCCSBtn;
    }
    else if (com.equals("Connector")) {

      return VConnLineBtn;
    }
    else if (com.equals("Connection Point")) {
      return VConnNodeBtn;
    }
    else if (com.equals("Ground")) {
      return VGroundBtn;
    }
    else if (com.equals("Current Marker")) {
      return VCurrentMarkerBtn;
    }
    else if (com.equals("Voltage Marker")) {
      return VVoltageMarkerBtn;
    }
    else if (com.equals("Voltage Differential Marker")) {
      return VVoltageDiffMarkerBtn;
    }
    else {
      return null;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void drawLine() {
    gridCanvas.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasForegroundColor);
    if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
    }
    drawAllComps();
    drawDrawingComps();
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();
    }
```

```
    if (ActiveState.IsNetlistNodesShown) {
      showNetListNodes();
    }
    if (ActiveState.IsNodesVoltagesShown) {
      showNodesVoltages();
    }
    if (ActiveState.IsNodesCurrentsShown) {
      showNodesCurrents();

    }
    gridCanvas.gridImage.setColor(drawingLineColor);
    gridCanvas.gridImage.drawLine(xyold[0], xyold[1], xynew[0], xynew[1]);
    gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  private void drawRectangle(boolean round) {
    gridCanvas.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasForegroundColor);
    if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
    }
    drawAllComps();
    drawDrawingComps();
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();
    }
    if (ActiveState.IsNetlistNodesShown) {
      showNetListNodes();
    }
    if (ActiveState.IsNodesVoltagesShown) {
      showNodesVoltages();
    }
    if (ActiveState.IsNodesCurrentsShown) {
      showNodesCurrents();

    }
    gridCanvas.gridImage.setColor(drawingFillColor);
    if (!round) {
      gridCanvas.gridImage.fillRect(xyold[0], xyold[1], xynew[0] - xyold[0],
                                    xynew[1] - xyold[1]);
```

```
    gridCanvas.gridImage.setColor(drawingLineColor);
    gridCanvas.gridImage.drawRect(xyold[0], xyold[1], xynew[0] - xyold[0],
                                  xynew[1] - xyold[1]);
   }
   else {
    gridCanvas.gridImage.fillRoundRect(xyold[0], xyold[1], xynew[0] - xyold[0],
                                       xynew[1] - xyold[1], 20, 20);
    gridCanvas.gridImage.setColor(drawingLineColor);
    gridCanvas.gridImage.drawRoundRect(xyold[0], xyold[1], xynew[0] - xyold[0],
                                       xynew[1] - xyold[1], 20, 20);
   }
   gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  private void drawCircle() {
   gridCanvas.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
   gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
   gridCanvas.gridImage.setColor(canvasBackgroundColor);
   gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
   gridCanvas.gridImage.setColor(canvasForegroundColor);
   if (ActiveState.showGrid) {
    gridCanvas.drawGridLayout(startPoint);
   }
   drawAllComps();
   drawDrawingComps();
   if (!pageTitle.equals(new String(""))) {
    drawPageTitle();
   }
   if (ActiveState.IsNetlistNodesShown) {
    showNetListNodes();
   }
   if (ActiveState.IsNodesVoltagesShown) {
    showNodesVoltages();
   }
   if (ActiveState.IsNodesCurrentsShown) {
    showNodesCurrents();

   }
   gridCanvas.gridImage.setColor(drawingFillColor);
   gridCanvas.gridImage.fillOval(xyold[0], xyold[1], xynew[0] - xyold[0],
                                 xynew[1] - xyold[1]);
   gridCanvas.gridImage.setColor(drawingLineColor);
   gridCanvas.gridImage.drawOval(xyold[0], xyold[1], xynew[0] - xyold[0],
```

```
                        xynew[1] - xyold[1]);
    gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  private void drawArc() {
    gridCanvas.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasForegroundColor);
    if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
    }
    drawAllComps();
    drawDrawingComps();
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();
    }
    if (ActiveState.IsNetlistNodesShown) {
      showNetListNodes();
    }
    if (ActiveState.IsNodesVoltagesShown) {
      showNodesVoltages();
    }
    if (ActiveState.IsNodesCurrentsShown) {
      showNodesCurrents();

    }
    gridCanvas.gridImage.setColor(drawingFillColor);
    gridCanvas.gridImage.fillArc(xyold[0], xyold[1], xynew[0] - xyold[0],
                        xynew[1] - xyold[1], 0, 180);
    gridCanvas.gridImage.setColor(drawingLineColor);
    gridCanvas.gridImage.drawArc(xyold[0], xyold[1], xynew[0] - xyold[0],
                        xynew[1] - xyold[1], 0, 180);
    gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  private void drawDrawingComps() {
    for (int count = 0; count < Array.getLength(drawComps) - 1; count++) {
      drawComps[count].drawComponent(gridCanvas);
    }
```

```
  }

////////////////////////////////////////////////////////////////////////////
  public void updateCanvas(int width, int height, Color backColor,
                           Color foreColor) {
    canvasWidth = width;
    canvasHeight = height;
    canvasBackgroundColor = backColor;
    canvasForegroundColor = foreColor;
    gridCanvas.resizeCanvas(canvasWidth, canvasHeight);

    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void deActivaePages() {
    new1Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW1]));
    new2Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW2]));
    new3Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW3]));
    new4Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW4]));
    new5Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW5]));

  }

////////////////////////////////////////////////////////////////////////////
  private void clearGridCanvas() {
    gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasBackgroundColor);
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
    gridCanvas.gridImage.setColor(canvasForegroundColor);
    if (ActiveState.showGrid) {
      gridCanvas.drawGridLayout(startPoint);
    }
    gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  private void mouseMoved_drawingCompsActions(MouseEvent me) {
    int[] xyPos = {
        me.getX(), me.getY()};
    for (int numb = 0; numb < Array.getLength(drawComps) - 1; numb++) {

      if (drawComps[numb].mouseOverArea(xyPos)) {
        gridCanvas.setCursor(new Cursor(Cursor.MOVE_CURSOR));
```

```
    }
   }

  }

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

  private void mouseClicked_ToolbarsActions(MouseEvent me) {
    /*if (ActiveState.zoomInCmd) {
      double xScale = ActiveState.currentXScale;
      double yScale = ActiveState.currentYScale;
      int[] stPoint = startPoint;
      Graphics2D G = (Graphics2D) gridCanvas.gridImage;
      xScale = (xScale + 0.1) / xScale;
      yScale = (yScale + 0.1) / yScale;
      ActiveState.currentXScale = xScale;
      ActiveState.currentYScale = yScale;
      G.scale(xScale, yScale);
      gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], canvasWidth,
                                     canvasHeight);
      if (ActiveState.showGrid) {
        gridCanvas.drawGridLayout(startPoint);
      }
      drawAllComps();
      drawDrawingComps();
      if (!pageTitle.equals("")) {
        drawPageTitle();
      }
      gridCanvas.redraw();
      System.out.println(xScale + "," + yScale);
      ActiveState.zoomInCmd = false;
      gridCanvas.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
         }*/
    /* if (ActiveState.zoomOutCmd) {
       double xScale = ActiveState.currentXScale;
       double yScale = ActiveState.currentYScale;
       int[] stPoint = startPoint;
       Graphics2D G = (Graphics2D) gridCanvas.gridImage;
       xScale = (xScale - 0.1) / xScale;
       yScale = (yScale - 0.1) / yScale;
       ActiveState.currentXScale = xScale;
       ActiveState.currentYScale = yScale;
       G.scale(xScale, yScale);
         gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 2500, 2500);
```

```
      if (ActiveState.showGrid) {
        gridCanvas.drawGridLayout(startPoint);
      }
      drawAllComps();
      drawDrawingComps();
      if (!pageTitle.equals("")) {
        drawPageTitle();
      }
      gridCanvas.redraw();
      System.out.println(xScale + "," + yScale);
      ActiveState.zoomOutCmd = false;
      gridCanvas.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
     }*/
    if (ActiveState.insertTextCmd) {
      InsertTextFrame textFrame = new InsertTextFrame(this, -1);
      textFrame.setLocation(clickedPos[0], clickedPos[1]);
      textFrame.show();
      ActiveState.insertTextCmd = false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void mouseDragged_ToolbarsActions(MouseEvent me) {
    int[] xy = {
        me.getX(), me.getY()};

    if (ActiveState.handCmd) {

      int hValue = mainScroll.getHorizontalScrollBar().getValue();
      int vValue = mainScroll.getVerticalScrollBar().getValue();
      if (xy[0] < xyold[0]) {
        mainScroll.getHorizontalScrollBar().setValue(hValue + 25);
      }
      else {
        mainScroll.getHorizontalScrollBar().setValue(hValue - 25);

      }
      if (xy[1] < xyold[1]) {
        mainScroll.getVerticalScrollBar().setValue(vValue + 25);
      }
      else {
        mainScroll.getVerticalScrollBar().setValue(vValue - 25);

      }
```

```
      xyold[0] = xy[0];
      xyold[1] = xy[1];
      startPoint[0] = mainScroll.getHorizontalScrollBar().getValue();
      startPoint[1] = mainScroll.getVerticalScrollBar().getValue();

    }
    else if (ActiveState.lineCmd) {
      drawLine();
    }
    else if (ActiveState.rectangleCmd) {
      drawRectangle(false);
    }
    else if (ActiveState.circleCmd) {
      drawCircle();
    }
    else if (ActiveState.roundRectangleCmd) {
      drawRectangle(true);
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void mouseReleased_ToolbarsActions(MouseEvent me) {

    if (ActiveState.lineCmd) {
      int size = Array.getLength(drawComps);
      drawComps[size - 1] = new Line(xyPressed, xyReleased);
      drawComps[size - 1].setStartPoint(xyPressed);
      drawComps[size - 1].setEndPoint(xyReleased);
      drawComps[size - 1].setLineColor(drawingLineColor);
      drawComps[size - 1].drawComponent(gridCanvas);
      drawComps = (DrawTool[]) growArray(drawComps);
      gridCanvas.redraw();
      ActiveState.lineCmd = false;

    }
    if (ActiveState.rectangleCmd) {
      int size = Array.getLength(drawComps);
      drawComps[size -
          1] = new ECS.DrawingTools.Rectangle(xyold[0], xyold[1],
                                      xynew[0] - xyold[0],
                                      xynew[1] - xyold[1]);
      drawComps[size - 1].setFillColor(drawingFillColor);
      drawComps[size - 1].setLineColor(drawingLineColor);
      drawComps[size - 1].drawComponent(gridCanvas);
```

```
      drawComps = (DrawTool[]) growArray(drawComps);
      gridCanvas.redraw();
      ActiveState.rectangleCmd = false;
    }
    else if (ActiveState.circleCmd) {
      int size = Array.getLength(drawComps);
      drawComps[size -
          1] = new Circle(xyold[0], xyold[1], xynew[0] - xyold[0],
                  xynew[1] - xyold[1]);
      drawComps[size - 1].setFillColor(drawingFillColor);
      drawComps[size - 1].setLineColor(drawingLineColor);
      drawComps[size - 1].drawComponent(gridCanvas);
      drawComps = (DrawTool[]) growArray(drawComps);
      gridCanvas.redraw();
      ActiveState.circleCmd = false;

    }
    else if (ActiveState.roundRectangleCmd) {
      int size = Array.getLength(drawComps);
      drawComps[size -
          1] = new ECS.DrawingTools.Rectangle(xyold[0], xyold[1],
                                  xynew[0] - xyold[0],
                                  xynew[1] - xyold[1]);
      drawComps[size - 1].setRoundType(true);
      drawComps[size - 1].setFillColor(drawingFillColor);
      drawComps[size - 1].setLineColor(drawingLineColor);
      drawComps[size - 1].drawComponent(gridCanvas);
      drawComps = (DrawTool[]) growArray(drawComps);
      gridCanvas.redraw();
      ActiveState.roundRectangleCmd = false;

    }

  }

////////////////////////////////////////////////////////////////////////////
  private boolean checkNetListErrors() {

    boolean groundNotExist = true;
    boolean floatingNode = false;
    boolean noSchematic = false;
    boolean connectionLinesOnly = false;
    boolean groundsOnly = false;
    boolean connectionNodesOnly = false;
    boolean groundNodesExist = false;
```

```
    boolean shortCircuitExist = false;

    boolean depControllerNotExist = false;
// Check Ground if it exists
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getComponentType() != ActiveState.CONNECTOR &&
            allComps[type][numb].getComponentType() != ActiveState.VOLTAGEMARK &&
            allComps[type][numb].getComponentType() != ActiveState.CURRENTMARK &&
            allComps[type][numb].getComponentType() != ActiveState.NODEL &&
            allComps[type][numb].getComponentType() != ActiveState.NODER) {

          int numbNodes = allComps[type][numb].getNumNodes();
          for (int numbN = 0; numbN < numbNodes; numbN++) {
            if (allComps[type][numb].getNode(numbN).strName.equals("0")) {
              groundNotExist = false;
            }
          }
        }
      }
    }
// check Node floating..
    for (int i = 0; i < netListNodes.size(); i++) {
      if ( ( (NetNode) netListNodes.elementAt(i)).hit == 1) {
        //floatingNode = true;
      }
    }

// check schematic existance
    if (checkSchematicEmpty()) {
      noSchematic = true;
// check only connection lines
    }
    if (checkOnlyComponentType(ActiveState.CONNECTOR)) {
      connectionLinesOnly = true;
// check only grounds
    }
    if (checkOnlyComponentType(ActiveState.GROUND)) {
      groundsOnly = true;
// check only Connection Nodes
    }
    if (checkOnlyComponentType(ActiveState.CONNODE)) {
      connectionNodesOnly = true;
    }
```

```
// check short circuit
    if (checkShortCircuit()) {
      shortCircuitExist = true;
    }

    if (noSchematic) {
      ErrorMessage NoSchematicError = new ErrorMessage("Schematic Not Exist");
      NoSchematicError.show();
    }
    else if (connectionLinesOnly) {
      AlertMessage onlyConnectionLinesAlert = new AlertMessage(
          "Only Connection Lines Found");
      onlyConnectionLinesAlert.show();
    }
    else if (groundsOnly) {
      AlertMessage onlyGroundsAlert = new AlertMessage("Only Grounds Found");
      onlyGroundsAlert.show();
    }
    else if (connectionNodesOnly) {
      AlertMessage onlyConnectionNodesAlert = new AlertMessage(
          "Only Connection Nodes Found");
      onlyConnectionNodesAlert.show();
    }
    else if (floatingNode) {
      AlertMessage floatingNodeAlert = new AlertMessage("Node Floating Error");
      floatingNodeAlert.show();
    }
    else if (groundNotExist) {
      AlertMessage groundNotExistAlert = new AlertMessage(
          "Schematic is missing a Ground node");
      groundNotExistAlert.show();
    }
    else if (shortCircuitExist) {
      AlertMessage shortCircuitExistAlert = new AlertMessage(
          "Short Circuit Fround");
      shortCircuitExistAlert.show();
    }
    else if (checkDependentControllers()) {
      // check dependent controllers
      depControllerNotExist = true;
    }

    if (groundNotExist || connectionLinesOnly || groundsOnly ||
        noSchematic || depControllerNotExist || shortCircuitExist) {
```

```
      return false;
    }
    else {
      return true;
    }
  }

////////////////////////////////////////////////////////////////////////////
  private boolean generateNetList() {
    netListNodes.removeAllElements();
    resetNodesEquivalentPositions();
    optimizeNetwork();

    if (checkNetListErrors() && generateProbesVector()) {
      cirMLString = parseNETLISTToCIRML();

      netlistPanelString = netlistPanelString + netListString + "\n";
      netlistPanel.netText.setText(netlistPanelString);

      return true;
    }
    else {
      return false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void savePreviousPage() {
    switch (ActiveState.currentPageNumber) {
      case 1:
        page1.saveComponents(allComps, drawComps);
        page1.saveCanvasWidth(canvasWidth);
        page1.saveCanvasHeight(canvasHeight);
        page1.saveCanvasBackgroundColor(canvasBackgroundColor);
        page1.saveCanvasForegroundColor(canvasForegroundColor);
        page1.savePageTitle(pageTitle);
        page1.pageSaved = ActiveState.pageSaved;
        page1.pageFile = ActiveState.pageFile;
        break;
      case 2:
        page2.saveComponents(allComps, drawComps);
        page2.saveCanvasWidth(canvasWidth);
        page2.saveCanvasHeight(canvasHeight);
        page2.saveCanvasBackgroundColor(canvasBackgroundColor);
```

```
        page2.saveCanvasForegroundColor(canvasForegroundColor);
        page2.savePageTitle(pageTitle);
        page2.pageSaved = ActiveState.pageSaved;
        page2.pageFile = ActiveState.pageFile;
        break;
      case 3:

//page3.deleteComponents();
        page3.saveComponents(allComps, drawComps);
        page3.saveCanvasWidth(canvasWidth);
        page3.saveCanvasHeight(canvasHeight);
        page3.saveCanvasBackgroundColor(canvasBackgroundColor);
        page3.saveCanvasForegroundColor(canvasForegroundColor);
        page3.savePageTitle(pageTitle);
        page3.pageSaved = ActiveState.pageSaved;
        page3.pageFile = ActiveState.pageFile;
        break;
      case 4:

//page4.deleteComponents();
        page4.saveComponents(allComps, drawComps);
        page4.saveCanvasWidth(canvasWidth);
        page4.saveCanvasHeight(canvasHeight);
        page4.saveCanvasBackgroundColor(canvasBackgroundColor);
        page4.saveCanvasForegroundColor(canvasForegroundColor);
        page4.savePageTitle(pageTitle);
        page4.pageSaved = ActiveState.pageSaved;
        page4.pageFile = ActiveState.pageFile;
        break;
      case 5:

//page5.deleteComponents();
        page5.saveComponents(allComps, drawComps);
        page5.saveCanvasWidth(canvasWidth);
        page5.saveCanvasHeight(canvasHeight);
        page5.saveCanvasBackgroundColor(canvasBackgroundColor);
        page5.saveCanvasForegroundColor(canvasForegroundColor);
        page5.savePageTitle(pageTitle);
        page5.pageSaved = ActiveState.pageSaved;
        page5.pageFile = ActiveState.pageFile;
        break;
    }
    ;
    pageTitle = "";
    ActiveState.pageFile = null;
```

```
    ActiveState.pageSaved = false;
  }

////////////////////////////////////////////////////////////////////////////
  private void createNewPage() {

    savePreviousPage();
    /*pagesExplorerToolbar.resize(30,10);
        int pageNumb=ActiveState.numberOfPages+1;
        JButton newBtn=new JButton("2");
        newBtn.setBackground(new Color(91, 141, 255));
        newBtn.setForeground(Color.WHITE);
        newBtn.setBorder(null);
        newBtn.setVisible(true);
        newBtn.setActionCommand(new Integer(pageNumb).toString());
        newBtn.setBounds(3 , 20+35*ActiveState.numberOfPages, 22, 23);
        pagesExplorerToolbar.addSeparator();
        pagesExplorerToolbar.add(newBtn);
        pagesExplorerToolbar.repaint();
        ActiveState.numberOfPages++;
        ActiveState.currentPageNumber=ActiveState.numberOfPages;*/

    deActivatePages();
    switch (ActiveState.numberOfPages) {
      case 1:
        new2Btn.setVisible(true);
        new2Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW2]));
        ActiveState.currentPageNumber = 2;
        allComps = page2.getCircuitComponents();
        drawComps = page2.getDrawingComponents();
        goToPageMenu.add(page2MenuItem);
        page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        break;
      case 2:
        new3Btn.setVisible(true);
        new3Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW3]));
        ActiveState.currentPageNumber = 3;
        allComps = page3.getCircuitComponents();
        drawComps = page3.getDrawingComponents();
        goToPageMenu.add(page3MenuItem);
        page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
```

```
      break;
    case 3:
      new4Btn.setVisible(true);
      new4Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW4]));
      ActiveState.currentPageNumber = 4;
      allComps = page4.getCircuitComponents();
      drawComps = page4.getDrawingComponents();
      goToPageMenu.add(page4MenuItem);
      page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      break;
    case 4:
      new5Btn.setVisible(true);
      new5Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW5]));
      ActiveState.currentPageNumber = 5;
      allComps = page5.getCircuitComponents();
      drawComps = page5.getDrawingComponents();
      goToPageMenu.add(page5MenuItem);
      page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      break;
  }
  if (ActiveState.numberOfPages < 5) {
    ActiveState.numberOfPages++;

  }

  resetCanvasParameters();
  clearAllComps();

 }

////////////////////////////////////////////////////////////////////////////
 private void deleteCurrentSchematic() {
  QuestionMessage qMsg = new QuestionMessage(
```

```
      "Do you want to save the current schematic page", this);
    qMsg.show();
  }

////////////////////////////////////////////////////////////////////////////
  public void resetCanvasParameters() {
    canvasWidth = 2500;
    canvasHeight = 2500;
    canvasBackgroundColor = Color.WHITE;
    canvasForegroundColor = Color.BLUE;
    pageTitle = "";
  }

////////////////////////////////////////////////////////////////////////////
  public void removeCurrentPage() {

    deActivatePages();

    switch (ActiveState.currentPageNumber) {
      case 1:
        swapPages(page1, page2);
        swapPages(page2, page3);
        swapPages(page3, page4);
        swapPages(page4, page5);
        page5.resetPage();
        break;
      case 2:

        swapPages(page2, page3);
        swapPages(page3, page4);
        swapPages(page4, page5);
        break;
      case 3:
        swapPages(page3, page4);
        swapPages(page4, page5);
        break;
      case 4:
        swapPages(page4, page5);
        break;
    }
    ;

    switch (ActiveState.numberOfPages) {
      case 5:
        new5Btn.setVisible(false);
```

```
      new4Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW4]));
      page5.resetPage();
      break;
    case 4:
      new4Btn.setVisible(false);
      new3Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW3]));
      page4.resetPage();
      break;
    case 3:
      new3Btn.setVisible(false);
      new2Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW2]));
      page3.resetPage();
      break;
    case 2:
      new2Btn.setVisible(false);
      new1Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW1]));
      page2.resetPage();
      break;
    case 1:
      new1Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW1]));
      page1.resetPage();
      break;
    default:
      new1Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW1]));
      page1.resetPage();
   }

   if (ActiveState.numberOfPages > 0) {
    ActiveState.numberOfPages--;
   }
   ActiveState.currentPageNumber = ActiveState.numberOfPages;

   loadCurrentPage();
  }

////////////////////////////////////////////////////////////////////////////
  private void loadCurrentPage() {

   switch (ActiveState.currentPageNumber) {
    case 5:
      allComps = page5.getCircuitComponents();
      drawComps = page5.getDrawingComponents();
      updateCanvas(page5.canvasWidth, page5.canvasHeight,
                page5.canvasBackgroundColor,
                page5.canvasForegroundColor);
```

```
  pageTitle = page5.getPageTitle();
  ActiveState.pageSaved = page5.pageSaved;
  ActiveState.pageFile = page5.pageFile;
  break;
 case 4:
  allComps = page4.getCircuitComponents();
  drawComps = page4.getDrawingComponents();
  updateCanvas(page4.canvasWidth, page4.canvasHeight,
            page4.canvasBackgroundColor,
            page4.canvasForegroundColor);
  pageTitle = page4.getPageTitle();
  ActiveState.pageSaved = page4.pageSaved;
  ActiveState.pageFile = page4.pageFile;
  break;
 case 3:
  allComps = page3.getCircuitComponents();
  drawComps = page3.getDrawingComponents();
  updateCanvas(page3.canvasWidth, page3.canvasHeight,
            page3.canvasBackgroundColor,
            page3.canvasForegroundColor);
  pageTitle = page3.getPageTitle();
  ActiveState.pageSaved = page3.pageSaved;
  ActiveState.pageFile = page3.pageFile;
  break;
 case 2:
  allComps = page2.getCircuitComponents();
  drawComps = page2.getDrawingComponents();
  updateCanvas(page2.canvasWidth, page2.canvasHeight,
            page2.canvasBackgroundColor,
            page2.canvasForegroundColor);
  pageTitle = page2.getPageTitle();
  ActiveState.pageSaved = page2.pageSaved;
  ActiveState.pageFile = page2.pageFile;
  break;
 case 1:

  allComps = page1.getCircuitComponents();
  drawComps = page1.getDrawingComponents();
  updateCanvas(page1.canvasWidth, page1.canvasHeight,
            page1.canvasBackgroundColor,
            page1.canvasForegroundColor);
  pageTitle = page1.getPageTitle();
  ActiveState.pageSaved = page1.pageSaved;
  ActiveState.pageFile = page1.pageFile;
  break;
```

```
      case 0:
        page1.resetPage();
        allComps = page1.getCircuitComponents();
        drawComps = page1.getDrawingComponents();
        updateCanvas(page1.canvasWidth, page1.canvasHeight,
                     page1.canvasBackgroundColor,
                     page1.canvasForegroundColor);
        pageTitle = page1.getPageTitle();
        ActiveState.pageSaved = page1.pageSaved;
        ActiveState.pageFile = page1.pageFile;
        break;
    }
    ;
  }

////////////////////////////////////////////////////////////////////////////
  public void swapPages(WorkPlace p1, WorkPlace p2) {
    p1.saveCanvasWidth(p2.getCanvasWidth());
    p1.saveCanvasHeight(p2.getCanvasHeight());
    p1.saveCanvasBackgroundColor(p2.getCanvasBackgroundColor());
    p1.saveCanvasForegroundColor(p2.getCanvasForegroundColor());
    p1.saveComponents(p2.getCircuitComponents(), p2.getDrawingComponents());
    p1.savePageTitle(p2.getPageTitle());

  }

////////////////////////////////////////////////////////////////////////////
  public void mouseDragged_drawCompsActions(MouseEvent me) {
    int[] xy = {
       me.getX(), me.getY()};
    if (ActiveState.drawingCompActive != null) {

      if (ActiveState.drawingCompActive.getName().equals("Line")) {
        int[] p1 = ActiveState.drawingCompActive.getStartPoint();
        int[] p2 = ActiveState.drawingCompActive.getEndPoint();
        int xdiff = xy[0] - xyPressed[0];
        int ydiff = xy[1] - xyPressed[1];
        p1[0] += xdiff;
        p1[1] += ydiff;
        p2[0] += xdiff;
        p2[1] += ydiff;
        ActiveState.drawingCompActive.setStartPoint(p1);
        ActiveState.drawingCompActive.setEndPoint(p2);
        xyPressed = xy;
      }
```

```
      else {
        int[] oldPos = ActiveState.drawingCompActive.getPosition();
        int xdiff = xy[0] - oldPos[0];
        int ydiff = xy[1] - oldPos[1];
        int[] p = ActiveState.drawingCompActive.getPosition();
        p[0] += xdiff;
        p[1] += ydiff;
        ActiveState.drawingCompActive.setPosition(p);
      }
      redrawGridCanvas();
    }

  }

////////////////////////////////////////////////////////////////////////////
  public void mouseReleased_drawCompsActions(MouseEvent me) {

  }

////////////////////////////////////////////////////////////////////////////
  public void mouseClicked_drawCompsActions(MouseEvent me) {
    ActiveState.drawingCompActive = null;

    int[] pos = {
       me.getX(), me.getY()};
    for (int numb = 0; numb < Array.getLength(drawComps) - 1; numb++) {
      if (drawComps[numb].occupiedArea(pos)) {
        ActiveState.drawingCompActive = drawComps[numb];
      }
    }

    int[] xypos = {
       me.getX(), me.getY()};
    if (me.getClickCount() >= 2) {
      for (int numb = 0; numb < Array.getLength(drawComps) - 1; numb++) {
        if (drawComps[numb].occupiedArea(xypos)) {
          InsertTextFrame textFrame = new InsertTextFrame(this, numb);
          textFrame.setLocation(clickedPos[0], clickedPos[1]);
          textFrame.txtArea.setText(drawComps[numb].getText());
          textFrame.show();

        }

      }
```

```
    }

    redrawGridCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void showHelpWindow() {
    try {
      getAppletContext().showDocument(new URL(getDocumentBase(),
                                    "Tutorial/help.chm"));

    }
    catch (IOException ioe) {

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void openLocalSchematic() {
    load = true;
    JFileChooser fc = new JFileChooser(new File("/"));
    fc.setSize(300, 400);
    fc.setEnabled(true);
    fc.addChoosableFileFilter(new MyFilter());
    int returnValue = fc.showOpenDialog(this.getContentPane());

    switch (returnValue) {
      case JFileChooser.APPROVE_OPTION:

        File loadedFile = fc.getSelectedFile();
        //try {
        try {
          FileInputStream FIS = new FileInputStream(loadedFile);
          ObjectInputStream OIS = new ObjectInputStream(FIS);

          WorkPlace page = (WorkPlace) OIS.readObject();

          allComps = page.getCircuitComponents();
          drawComps = page.getDrawingComponents();
          updateCanvas(page.canvasWidth, page.canvasHeight,
                  page.canvasBackgroundColor,
                  page.canvasForegroundColor);
          pageTitle = page.getPageTitle();
```

```
        redrawGridCanvas();
        //page.resetPage();
        //allComps=(Components[][]) OIS.readObject();
       }
       catch (Exception ne) {
        ne.printStackTrace();
       }

       // }
       //catch (NotSerializableException ne) {
       // System.out.print(ne);
       //}

       redrawGridCanvas();
       load = false;

       break;
     case JFileChooser.CANCEL_OPTION:
       System.out.println("CANCELED");
       break;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void selectAllComps() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        allComps[type][numb].setActiveState(true);
      }
    }

    for (int numb = 0; numb < Array.getLength(drawComps) - 1; numb++) {
      drawComps[numb].setActiveState(true);
    }
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void deActivateAllComps() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        allComps[type][numb].setActiveState(false);
      }
```

```
    }
    for (int numb = 0; numb < drawComps.length - 1; numb++) {
      drawComps[numb].setActiveState(false);
    }
    redrawGridCanvas();
    ActiveState.active = false;
    ActiveState.activeComp = null;
    ActiveState.multiActive = false;
    ActiveState.drawingCompActive = null;
  }

////////////////////////////////////////////////////////////////////////////
  private String parseNETLISTToCIRML() {

    String cirString = "";
    generateProbesVector();
    cirString = "*CIRML format (.CIR) file\n\n";
    cirString = cirString + "*Analysis directives:\n";
    cirString = cirString + ".TRAN " + "0 " + endTime + " " + startTime + " " +
       stepTime + "\n";
    cirString = cirString + ".PROBE/CSDF " + probesString + "\n";
    cirString += "\n" + generateCIRMLString();
    cirString += "\n.END";

    if (cirString == "") {
      netlistPanelString = netlistPanelString +
          "\n\nCIRML format has been created.....\n\n";
    }
    else {
      netlistPanelString = netlistPanelString +
          "\n\nCIRML format has been updated.....\n\n";
    }
    netlistPanelString += cirString;
    netlistPanel.netText.setText(netlistPanelString);

    return cirString;
  }

////////////////////////////////////////////////////////////////////////////
  private String generateNETLISTString() {

    netListString = "";
    netListString = "*Schematic Netlist";
```

```
    for (int type = 0; type < componentsCount - 8; type++) {
      for (int count = 0; count < Array.getLength(allComps[type]) - 1; count++) {
        netListString += "\n" + allComps[type][count].getNetListString();
      }
    }

    return netListString;
  }

////////////////////////////////////////////////////////////////////////////
  private String generateCIRMLString() {
    String CIRMLString = "";
    CIRMLString = "*Schematic Netlist";

    for (int type = 0; type < componentsCount - 8; type++) {
      for (int count = 0; count < Array.getLength(allComps[type]) - 1; count++) {
        CIRMLString += "\n" + allComps[type][count].getCIRMLString();
      }
    }

    return CIRMLString;

  }

////////////////////////////////////////////////////////////////////////////
  public void setSimulationSettings(String start, String end, String step) {
    startTime = start;
    endTime = end;
    stepTime = step;
  }

////////////////////////////////////////////////////////////////////////////
  private void showNetListNodes() {

    for (int i = 0; i < netListNodes.size(); i++) {
      Point p = ( (NetNode) netListNodes.elementAt(i)).pos;
      gridCanvas.gridImage.setColor(Color.blue);
      gridCanvas.gridImage.fillRect(p.x - 15, p.y - 20, 30, 15);
      gridCanvas.gridImage.setColor(Color.WHITE);
      String netName = ( (NetNode) netListNodes.elementAt(i)).netName;
      gridCanvas.gridImage.drawString(netName, p.x - 10, p.y - 10);

    }

    /*
```

```
      for (int type = 0; type < componentsCount; type++) {
       for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
       if (allComps[type][numb].getComponentType() != ActiveState.CONNECTOR) {
           int numbNodes = allComps[type][numb].getNumNodes();
           for (int i = 0; i < numbNodes; i++) {
             int[] pos = allComps[type][numb].getNode(i).getPosition();
             gridCanvas.gridImage.setColor(Color.BLUE);
       gridCanvas.gridImage.fillOval(pos[0] - 20, pos[1] - 20, 30, 15);
             gridCanvas.gridImage.setColor(Color.WHITE);
       gridCanvas.gridImage.drawString(allComps[type][numb].getNode(i).
       strName, pos[0] - 10, pos[1] - 10);
             System.out.println(allComps[type][numb].getNode(i).strName);
           }
         }
        }
       }*/
    //gridCanvas.redraw();
    gridCanvas.gridImage.setColor(canvasForegroundColor);
  }

////////////////////////////////////////////////////////////////////////////
  private void mainScroll_mouseDragged(MouseEvent me) {
    int[] xy = {
       me.getX(), me.getY()};

    if (ActiveState.handCmd) {
      gridCanvas.setLocation(startPoint[0] - 5, startPoint[1] - 5);
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void mainScroll_mouseReleased(MouseEvent me) {

  }

////////////////////////////////////////////////////////////////////////////

  private void mainScroll_keyPressed(KeyEvent ke) {

  }

////////////////////////////////////////////////////////////////////////////
  private void drawGraphicalResults() {
    if (!graphicalResultStr.equals(new String(""))) {
```

```
    generateNetList();

    numbOfGraphs = probesVector.size();
    drawInitialGraph();

    for (int i = 1; i <= numbOfGraphs; i++) {
      double[] index = getResultVector(":" + (new Integer(i)).toString() +
                                       " ");
      drawVector(index, getGraphColor(i));
    }
   }

   graphicalResultStr = "";
  }

////////////////////////////////////////////////////////////////////////////
  private void drawInitialGraph() {
   if (probesVector.size() > 0) { //DRAW Y FUNCTIONS
    graphicalResult.drawGridArea(graphicalResult.picImage);
    int step = 0;
    for (int i = 0; i < probesVector.size(); i++) {
      graphicalResult.picImage.setColor(getGraphColor(i + 1));
      graphicalResult.picImage.fillRect(50 + step, 187, 5, 5);
      graphicalResult.picImage.setColor(Color.BLACK);
      graphicalResult.picImage.drawString( (String) probesVector.get(i),
                                          60 + step,
                                          195);

      step += 60;

    }
    //
    // DRAW Y & X AXISIS
    graphicalResult.picImage.setColor(Color.BLACK);
    graphicalResult.picImage.drawString(startTime + "S", 35, 185);
    graphicalResult.picImage.drawString(endTime + "S", 315, 185);

    double[] mins = new double[probesVector.size()];
    double[] maxs = new double[probesVector.size()];

    for (int i = 1; i <= probesVector.size(); i++) {
      double[] index = getResultVector(":" + (new Integer(i)).toString() +
                                       " ");
      mins[i - 1] = findMin(index);
      maxs[i - 1] = findMax(index);
```

```
      }
      minVI = findMin(mins);
      maxVI = findMax(maxs);
////////////////////////////////////////////////

      double minabs = Math.abs(minVI);
      double maxValue = Math.max(maxVI, minabs);
      maxValue = Math.ceil(maxValue);
      int scale = (new Double(maxValue)).intValue();

//////////////////////////////////////

      if (minVI < 0) {

        graphicalResult.picImage.drawString( (new Integer(scale)).
                                    toString(), 27, 25);
        graphicalResult.picImage.drawString("0", 25, 100);

        graphicalResult.picImage.drawString("-" + (new Integer(scale)).toString(),
                                    23, 175);

      }
      else {

        graphicalResult.picImage.drawString( (new Integer(scale)).toString(),
                                    27,
                                    25);
        graphicalResult.picImage.drawString("0", 25, 180);
      }

    }
  }

////////////////////////////////////////////////////////////////////////////
  private double[] getResultVector(String vecStr) {
    int c = graphicalResultStr.indexOf(vecStr);
    int i = 0;
    while (c != -1) {
      i++;
      c = graphicalResultStr.indexOf(vecStr, c + 5);
    }

    index = new double[i];
```

```
c = graphicalResultStr.indexOf(vecStr);
i = 0;
while (c != -1) {

  int length = 10;
  String k = graphicalResultStr.substring(c - length, c);

  if (k.indexOf("E-") == -1) {
    length--;
  }
  if (k.indexOf("-") == -1) {
    length--;

  }
  if (k.indexOf("-") != -1) {
    if (k.indexOf("-") == k.indexOf("E-") + 1) {
      length--;

    }
  }

  k = graphicalResultStr.substring(c - length, c);
  if (k.indexOf("E-") != -1) {
    if (k.indexOf("-") != k.indexOf("E-") + 1) {
      String pow = k.substring(8, 10);
      String numb = k.substring(0, 6);

      Double n = new Double(numb);

      Integer x = new Integer(pow);

      index[i] = (double) (n.doubleValue() / Math.pow(10, x.intValue()));

    }
    else {

      String pow = k.substring(7, 9);
      String numb = k.substring(0, 5);

      Double n = new Double(numb);

      Integer x = new Integer(pow);
      index[i] = (double) (n.doubleValue() / Math.pow(10, x.intValue()));

    }
```

```
    }
    else {
     if (k.indexOf("-") != -1) {
       String pow = k.substring(7, 9);

       String numb = k.substring(0, 6);

       Double n = new Double(numb);

       Integer x = new Integer(pow);

       index[i] = (double) (n.doubleValue() * Math.pow(10, x.intValue()));

     }
     else {
       String pow = k.substring(6, 8);
       String numb = k.substring(0, 5);

       Double n = new Double(numb);

       Integer x = new Integer(pow);

       index[i] = (double) (n.floatValue() * Math.pow(10, x.intValue()));

     }
    }

    c = graphicalResultStr.indexOf(vecStr, c + 5);
    i++;
   }

   return index;
  }

////////////////////////////////////////////////////////////////////////////
  private boolean checkDependentControllers() {
   for (int type = 0; type < componentsCount; type++) {
    for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
        numbComp++) {
     switch (allComps[type][numbComp].getComponentType()) {
       case ActiveState.DEPVCVS:
       case ActiveState.DEPVCCS:
         if ( ( (DepComp) allComps[type][numbComp]).controllers[0] == null &&
            ( (DepComp) allComps[type][numbComp]).controllers[1] == null) {
           String depName = ( (DepComp) allComps[type][numbComp]).
```

```
            getComponentName();
          AlertMessage controllerNotFound = new AlertMessage(
            "Voltage Controller Not Found for" + depName);
          controllerNotFound.show();
          return true;
        }
        break;

      case ActiveState.DEPCCVS:
      case ActiveState.DEPCCCS:
        if ( ( (DepComp) allComps[type][numbComp]).controllers[0] == null) {
          String depName = ( (DepComp) allComps[type][numbComp]).
            getComponentName();
          AlertMessage controllerNotFound = new AlertMessage(
            "Current Controller Not Found for:" + depName);
          controllerNotFound.show();
          return true;
        }
        break;

    }
    ;

  }

  }

  return false;

 }

////////////////////////////////////////////////////////////////////////////
 private boolean generateProbesVector() {
  probesString = "";
  probesVector.removeAllElements();

  for (int type = 0; type < componentsCount; type++) {
    for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
         numbComp++) {
      switch (allComps[type][numbComp].getComponentType()) {
        case ActiveState.NODEL:
          if (! ( (NodeL) allComps[type][numbComp]).isController()) {

            for (int cat = 0; cat < componentsCount; cat++) {
              for (int numb = 0; numb < Array.getLength(allComps[cat]) - 1;
```

```
  numb++) {
if (allComps[cat][numb].getComponentType() !=
    ActiveState.NODEL &&
    allComps[cat][numb].getComponentType() !=
    ActiveState.NODER &&
    allComps[cat][numb].getComponentType() !=
    ActiveState.CURRENTMARK
    &&
    allComps[cat][numb].getComponentType() !=
    ActiveState.GROUND
    &&
    allComps[cat][numb].getComponentType() !=
    ActiveState.CONNECTOR
    &&
    allComps[cat][numb].getComponentType() !=
    ActiveState.VOLTAGEMARK)

{
  int[] node1 = allComps[cat][numb].getNode(0).getPosition();
  int[] node2 = allComps[cat][numb].getNode(1).getPosition();
  int[] nodeL = allComps[ActiveState.NODEL][numbComp].getNode(
     0).getPosition();
  int[] nodeR = allComps[ActiveState.NODER][numbComp].getNode(

     0).getPosition();

  if ( (nodeL[0] == node1[0] && nodeL[1] == node1[1] &&
       nodeR[0] == node2[0] && nodeR[1] == node2[1])
      ||
      (nodeL[0] == node2[0] && nodeL[1] == node2[1] &&
       nodeR[0] == node1[0] && nodeR[1] == node1[1])) {
       /*if (allComps[cat][numb].getNode(0).strName.equals("0")) {
           probesString += "V(" +
     allComps[cat][numb].getNode(1).strName + ",0)";
        probesVector.add(new String("V(" +
     allComps[cat][numb].getNode(1).strName + ",0)"));
       }
     else if (allComps[cat][numb].getNode(1).strName.equals(
              "0")) {
           probesString += "V(" +
     allComps[cat][numb].getNode(0).strName + ",0)";
        probesVector.add(new String("V(" +
     allComps[cat][numb].getNode(0).strName + ",0)"));
       }
     */
```

```
          //else {
          probesString += "V(" + allComps[cat][numb].getName() +
              "_" + allComps[cat][numb].getComponentName() + ") ";
          probesVector.add(new String("V(" +
                                  allComps[cat][numb].
                                  getComponentName() +
                                  ") "));
        }

      }

    }
  }

 }

 break;
case ActiveState.CURRENTMARK:

 for (int cat = 0; cat < componentsCount; cat++) {
   for (int numb = 0; numb < Array.getLength(allComps[cat]) - 1;
       numb++) {
     if (allComps[cat][numb].getComponentType() !=
         ActiveState.NODEL &&
         allComps[cat][numb].getComponentType() !=
         ActiveState.NODER
         &&
         allComps[cat][numb].getComponentType() !=
         ActiveState.GROUND
         &&
         allComps[cat][numb].getComponentType() !=
         ActiveState.CONNECTOR
         &&
         allComps[cat][numb].getComponentType() !=
         ActiveState.VOLTAGEMARK
         &&
         allComps[cat][numb].getComponentType() !=
         ActiveState.CURRENTMARK
         )

     {
       int[] node1 = allComps[cat][numb].getNode(0).getPosition();
       int[] node2 = allComps[cat][numb].getNode(1).getPosition();
       int[] node = allComps[ActiveState.CURRENTMARK][numbComp].
           getNode(
```

```
          0).getPosition();
        //int[] nodeR = allComps[ActiveState.CURRENTCOMP][numbComp].getNode(
        //  0).getEquivPos();

        if ( (node[0] == node1[0] && node[1] == node1[1])
           ||
           (node[0] == node2[0] && node[1] == node2[1])) {

          probesString += "I(" +
             allComps[cat][numb].getName() + "_" +
             allComps[cat][numb].getComponentName() + ")";
          probesVector.add(new String("I(" +
                                allComps[cat][numb].
                                getComponentName() + ")"));

        }

      }

    }
  }

  break;

case ActiveState.VOLTAGEMARK:
  for (int cat = 0; cat < componentsCount; cat++) {
    for (int numb = 0; numb < Array.getLength(allComps[cat]) - 1;
         numb++) {
      if (allComps[cat][numb].getComponentType() !=
          ActiveState.NODEL &&
          allComps[cat][numb].getComponentType() !=
          ActiveState.NODER &&
          allComps[cat][numb].getComponentType() !=
          ActiveState.GROUND
          &&
          allComps[cat][numb].getComponentType() !=
          ActiveState.CONNECTOR
          &&
          allComps[cat][numb].getComponentType() !=
          ActiveState.VOLTAGEMARK
          &&
          allComps[cat][numb].getComponentType() !=
          ActiveState.CURRENTMARK
          )
```

```
          {
            int[] node1 = allComps[cat][numb].getNode(0).getPosition();
            int[] node2 = allComps[cat][numb].getNode(1).getPosition();
            int[] node = allComps[ActiveState.VOLTAGEMARK][numbComp].
                getNode(0).getPosition();
            // getNode(
            //0).getPosition();
            //int[] nodeR = allComps[ActiveState.CURRENTCOMP][numbComp].getNode(
            //  0).getEquivPos();

            if ( (node[0] == node1[0] && node[1] == node1[1])) {
              probesString += "N(" +
                  allComps[cat][numb].getNode(0).strName + ")";
              probesVector.add("N(" +
                            allComps[cat][numb].getNode(0).strName +
                            ")");

            }
            else if (node[0] == node2[0] && node[1] == node2[1]) {
              probesString += "N(" +
                  allComps[cat][numb].getNode(1).strName + ")";
              probesVector.add("N(" +
                            allComps[cat][numb].getNode(1).strName +
                            ")");

            }

          }

        }

        break;
    }
    ;

  }

}

if (probesVector.size() > 5) {
  AlertMessage probesSucceeded = new AlertMessage(
      "Number of Probes Succeeded");
  probesSucceeded.show();
```

```
      return false;
    }
    else {
      return true;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void drawVector(double[] index, Color graphColor) {
//Graphics g=graphicalResult.getGraphics();
    double length = 0, xstepScale;
    if (index.length > 300) {
      xstepScale = 1;
    }
    else {
      length = index.length;
      double xstep = 300 / length;
      xstep = Math.ceil(xstep);
      xstepScale = (new Double(xstep)).intValue();

    }

    int xAxis = 40;

    if (minVI < 0) {
      double minabs = Math.abs(minVI);
      double maxValue = Math.max(maxVI, minabs);
      maxValue = Math.ceil(maxValue);
      int scale = (new Double(maxValue)).intValue();

      int prevPixel = 0;
      int prevxAxis = xAxis;
      for (int i = 0; i < index.length; i++) {
        if (xAxis <= 340) {
          int pixel = parseValueToPixels(index[i], scale);
          if (i == 0) {
            prevPixel = pixel;
            prevxAxis = 40;
          }

          //graphicalResult.picImage.drawLine(prevxAxis,105-prevPixel,xAxis,105-pixel);
          graphicalResult.picImage.setColor(graphColor);
          graphicalResult.picImage.drawLine(prevxAxis, 100 - prevPixel, xAxis,
                                            100 - pixel);
```

```
        graphicalResult.picImage.setColor(Color.BLACK);

        prevxAxis = xAxis;
        xAxis += xstepScale;
        prevPixel = pixel;
       }
      }

    }
    else {

      double minabs = Math.abs(minVI);
      double maxValue = Math.max(maxVI, minabs);
      maxValue = Math.ceil(maxValue);
      int prevPixel = 0, prevxAxis = xAxis;
      //Double d=new Double(maxValue/6);

      int scale = (new Double(maxValue / 2)).intValue();

      for (int i = 0; i < index.length; i++) {
       if (xAxis <= 340) {
        int pixel = parseValueToPixels(index[i], scale);
        if (i == 0) {
         prevPixel = pixel;
         prevxAxis = 40;
        }

        graphicalResult.picImage.setColor(graphColor);
        graphicalResult.picImage.drawLine(prevxAxis, 175 - (prevPixel), xAxis,
                                  175 - (pixel));
        graphicalResult.picImage.setColor(Color.BLACK);

        prevxAxis = xAxis;
        xAxis += xstepScale;
        prevPixel = pixel;
       }
      }

    }

  }

////////////////////////////////////////////////////////////////////////////
  private double findMax(double[] a) {
    double max = a[0];
```

```
    for (int i = 1; i < a.length; i++) {
      max = Math.max(max, a[i]);

    }
    return max;
  }

////////////////////////////////////////////////////////////////////////////
  private double findMin(double[] a) {
    double min = a[0];
    for (int i = 1; i < a.length; i++) {
      min = Math.min(min, a[i]);

    }
    return min;

  }

////////////////////////////////////////////////////////////////////////////
  private int parseValueToPixels(double a, int scale) {

    double pixel = (a * 75) / scale;
    return (new Double(pixel)).intValue();
  }

////////////////////////////////////////////////////////////////////////////
  private Color getGraphColor(int i) {
    Color c = Color.BLACK;
    switch (i) {
      case 1:
        c = Color.BLUE;
        break;
      case 2:
        c = Color.RED;
        break;
      case 3:
        c = new Color(118, 159, 0);
        break;
      case 4:
        c = new Color(137, 39, 39);
        break;
    }
    ;
    return c;
  }
```

```
////////////////////////////////////////////////////////////////////////////
  private void showNodesVoltages() {
    if (!outputResult.equals(new String(""))) {
      int numbNodes = 0;

      for (int i = 0; i < netListNodes.size(); i++) {
        if ( ( (NetNode) (netListNodes.get(i))).netName != "0") {
          numbNodes++;

        }
      }
      double[] nodesVoltages = new double[numbNodes];

      for (int k = 0; k < numbNodes; k++) {
        int n = k + 1;
        String node = "N_" + n + ")";
        int nlength = node.length();
        int index = outputResult.indexOf(node);

        if (outputResult.substring(index + nlength + 4) != "-") {
          String nodeVoltage = outputResult.substring(index + nlength + 3,
              index + nlength + 10);
          nodesVoltages[k] = (new Double(nodeVoltage)).doubleValue();

        }
        else {

          String nodeVoltage = outputResult.substring(index + 4, index + 11);
          nodesVoltages[k] = (new Double(nodeVoltage)).doubleValue();
        }
      }

      /*for(int i=0;i<nodesVoltages.length;i++)
          {
          System.out.println(nodesVoltages[i]);
          }
       */
////////////////////
      for (int i = 1; i < netListNodes.size(); i++) {
        Point p = ( (NetNode) netListNodes.elementAt(i)).pos;
        String nodeV = new Double(nodesVoltages[i - 1]).toString();
        gridCanvas.gridImage.setColor(new Color(128, 0, 64));
        gridCanvas.gridImage.fillRect(p.x - 15, p.y, 50, 15);
        gridCanvas.gridImage.setColor(Color.WHITE);
```

```
        gridCanvas.gridImage.drawString(new Double(nodesVoltages[i - 1]).
                                        toString(),
                                        p.x - 10, p.y + 10);
      }
      Point p = ( (NetNode) netListNodes.elementAt(0)).pos;
      gridCanvas.gridImage.setColor(new Color(128, 0, 64));
      gridCanvas.gridImage.fillRect(p.x - 15, p.y, 50, 15);
      gridCanvas.gridImage.setColor(Color.WHITE);
      gridCanvas.gridImage.drawString("0V", p.x - 10, p.y + 10);

      //gridCanvas.redraw();
////////////////////
    }
  }

//////////////////////////////////////////////////////////////////////////////
  private void showNodesCurrents() {
    if (!outputResult.equals(new String(""))) {
      String[] voltagesName = new String[1];
      int numbVoltages = 0;
      for (int type = 0; type < componentsCount; type++) {
        for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
          switch (allComps[type][numb].getComponentType()) {
            case ActiveState.DCVSOURCE:
            case ActiveState.DCCSOURCE:
            case ActiveState.ACVSOURCE:
            case ActiveState.ACCSOURCE:
            case ActiveState.DEPVCCS:
            case ActiveState.DEPVCVS:
            case ActiveState.DEPCCCS:
            case ActiveState.DEPCCVS:
              voltagesName[numbVoltages] = allComps[type][numb].
                 getComponentName();
              voltagesName = (String[]) growArray(voltagesName);

              numbVoltages++;
              break;
          }
          ;
        }
      }

      for (int i = 0; i < voltagesName.length - 1; i++) {
        int index = outputResult.indexOf(voltagesName[i]);
        index = outputResult.indexOf(voltagesName[i], index + 5);
```

```
      int vlength = voltagesName[i].length();
      int length = 10;
      String voltage = outputResult.substring(index + vlength + 4,
                                      index + vlength + length + 4);
      //System.out.println(outputResult.substring(index + vlength + 4,
      //                                index + vlength + 5));

      //System.out.println(outputResult.substring(index + vlength + 11,
      //                                index + vlength + 12));
      //System.out.println(voltage);
      if (!outputResult.substring(index + vlength + 4, index + vlength + 5).
         equals("-")) {
        length--;
      }
      if (!outputResult.substring(index + vlength + 11, index + vlength + 12).
         equals("-")) {
        length--;
      }
      voltage = outputResult.substring(index + vlength + 4,
                               index + vlength + length + 4);

      for (int type = 0; type < componentsCount; type++) {
        for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
          if (allComps[type][numb].getComponentName() == voltagesName[i]) {
            int[] pos = allComps[type][numb].getNode(0).getPosition();
            gridCanvas.gridImage.setColor(Color.RED);
            gridCanvas.gridImage.fillRect(pos[0] - 15, pos[1] + 20, 60, 15);
            gridCanvas.gridImage.setColor(Color.WHITE);
            gridCanvas.gridImage.drawString(voltage, pos[0] - 10, pos[1] + 30);

          }
          //gridCanvas.redraw();
        }
      }

    }
  }
 }

////////////////////////////////////////////////////////////////////////////
  private int getNumberOfActiveComponents() {
    int numbOfActiveComponents = 0;

    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
```

```
        if (allComps[type][numb].getActiveState()) {
          numbOfActiveComponents++;

        }
       }
     }
    return numbOfActiveComponents;
  }

////////////////////////////////////////////////////////////////////////////
  private void drawMenubar() {
    menuBar = new JMenuBar();
    menuBar.setBounds(0, 0, 400, 10);
    menuBar.setBackground(new Color(178, 197, 251));

    toolsBarMenu = new JMenu("       Toolbars");
    toolsBarMenu.setBackground(new Color(214, 223, 247));

    fileMenu = new JMenu("File");
    fileMenu.setBackground(new Color(178, 197, 251));

    editMenu = new JMenu("Edit");
    editMenu.setBackground(new Color(178, 197, 251));

    viewMenu = new JMenu("View");
    viewMenu.setBackground(new Color(178, 197, 251));

    optionsMenu = new JMenu("Settings");
    optionsMenu.setBackground(new Color(178, 197, 251));

    navigateMenu = new JMenu("Navigate");
    navigateMenu.setBackground(new Color(178, 197, 251));

    drawMenu = new JMenu("Draw");
    drawMenu.setBackground(new Color(178, 197, 251));

    analysisMenu = new JMenu("Simulation & Analysis");
    analysisMenu.setBackground(new Color(178, 197, 251));

    helpMenu = new JMenu("Help");
    helpMenu.setBackground(new Color(178, 197, 251));

    canvasPropertiesMenu = new JMenu("Canvas Properties");
    canvasPropertiesMenu.setBackground(new Color(214, 223, 247));
```

```
zoomingMenu = new JMenu("        Zooming");
zoomingMenu.setBackground(new Color(214, 223, 247));

markersMenu = new JMenu("Markers");
markersMenu.setBackground(new Color(178, 197, 251));

componentsMenu = new JMenu("Component");
componentsMenu.setBackground(new Color(178, 197, 251));

rotateFlipMenu = new JMenu("       Rotate or Flip");
rotateFlipMenu.setBackground(new Color(214, 223, 247));

groupMenu = new JMenu("       Grouping");
groupMenu.setBackground(new Color(214, 223, 247));

alignMenu = new JMenu("       Aligning");
alignMenu.setBackground(new Color(214, 223, 247));

goToPageMenu = new JMenu("       Go To Page");
goToPageMenu.setBackground(new Color(214, 223, 247));

newMenuItem = new JMenuItem("New...       Ctrl+N",
                    new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUNEW]));
newMenuItem.setBackground(new Color(214, 223, 247));
newMenuItem.setActionCommand("newMenuItem");

openMenuItem = new JMenuItem("Open...     Ctrl+O",
                    new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUOPEN]));
openMenuItem.setBackground(new Color(214, 223, 247));
openMenuItem.setActionCommand("openMenuItem");

saveMenuItem = new JMenuItem("Save        Ctrl+S",
                    new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUSAVE]));
saveMenuItem.setBackground(new Color(214, 223, 247));
saveMenuItem.setActionCommand("saveMenuItem");

saveAsMenuItem = new JMenuItem("       Save As...");
saveAsMenuItem.setBackground(new Color(214, 223, 247));
saveAsMenuItem.setActionCommand("saveAsMenuItem");

closePageMenuItem = new JMenuItem("Close",
                        new ImageIcon(ActiveState.btnsExited[
```

```
   ActiveState.DELETEPAGEMENU]));
closePageMenuItem.setBackground(new Color(214, 223, 247));
closePageMenuItem.setActionCommand("closePageMenuItem");

cutMenuItem = new JMenuItem("Cut              Ctrl+X",
                    new ImageIcon(ActiveState.btnsExited[
                            ActiveState.MENUCUT]));
cutMenuItem.setBackground(new Color(214, 223, 247));
cutMenuItem.setActionCommand("cutMenuItem");

copyMenuItem = new JMenuItem("Copy           Ctrl+C",
                     new ImageIcon(ActiveState.btnsExited[
                             ActiveState.MENUCOPY]));
copyMenuItem.setBackground(new Color(214, 223, 247));
copyMenuItem.setActionCommand("copyMenuItem");

pasteMenuItem = new JMenuItem("Paste         Ctrl+V",
                      new ImageIcon(ActiveState.btnsExited[
                              ActiveState.MENUPASTE]));
pasteMenuItem.setBackground(new Color(214, 223, 247));
pasteMenuItem.setActionCommand("pasteMenuItem");

deleteMenuItem = new JMenuItem("Delete        Del",
                       new ImageIcon(ActiveState.btnsExited[
                               ActiveState.MENUREMOVE]));
deleteMenuItem.setBackground(new Color(214, 223, 247));
deleteMenuItem.setActionCommand("deleteMenuItem");

selectAllMenuItem = new JMenuItem("Select All    Ctrl+A",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUSELECTALL]));
selectAllMenuItem.setBackground(new Color(214, 223, 247));
selectAllMenuItem.setActionCommand("selectAllMenuItem");

undoMenuItem = new JMenuItem("Undo           Ctrl+Z",
                     new ImageIcon(ActiveState.btnsExited[
                             ActiveState.MENUUNDO]));
undoMenuItem.setBackground(new Color(214, 223, 247));
undoMenuItem.setActionCommand("undoMenuItem");

redoMenuItem = new JMenuItem("Redo           Ctrl+W",
                     new ImageIcon(ActiveState.btnsExited[
                             ActiveState.MENUREDO]));
redoMenuItem.setBackground(new Color(214, 223, 247));
redoMenuItem.setActionCommand("redoMenuItem");
```

```
zoomInMenuItem = new JMenuItem("Zoom In       Shift+I",
                        new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUZOOMIN]));
zoomInMenuItem.setBackground(new Color(214, 223, 247));
zoomInMenuItem.setActionCommand("zoomInMenuItem");

fitZoomMenuItem = new JMenuItem("Fit Zoom      Shift+F",
                        new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUFITZOOM]));
fitZoomMenuItem.setBackground(new Color(214, 223, 247));
fitZoomMenuItem.setActionCommand("fitZoomMenuItem");

zoomOutMenuItem = new JMenuItem("Zoom Out    Shift+O",
                        new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUZOOMOUT]));
zoomOutMenuItem.setBackground(new Color(214, 223, 247));
zoomOutMenuItem.setActionCommand("zoomOutMenuItem");

scrollHandMenuItem = new JMenuItem("Scroll Hand",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.HAND]));
scrollHandMenuItem.setBackground(new Color(214, 223, 247));
scrollHandMenuItem.setActionCommand("scrollHandMenuItem");

defaultCursorMenuItem = new JMenuItem("Default Cursor",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.DCURSOR]));
defaultCursorMenuItem.setBackground(new Color(214, 223, 247));
defaultCursorMenuItem.setActionCommand("defaultCursorMenuItem");

helpMenuItem = new JMenuItem("Help                F1",
                        new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUHELP]));
helpMenuItem.setBackground(new Color(214, 223, 247));
helpMenuItem.setActionCommand("helpMenuItem");

refreshMenuItem = new JMenuItem("Refresh",
                        new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUREFRESH]));
refreshMenuItem.setBackground(new Color(214, 223, 247));
refreshMenuItem.setActionCommand("refreshMenuItem");

standardBarMenuItem = new JMenuItem("Standard",
                        new ImageIcon(ActiveState.btnsExited[
```

```
   ActiveState.MENUCHECKED]));
standardBarMenuItem.setBackground(new Color(214, 223, 247));
standardBarMenuItem.setActionCommand("standardBarMenuItem");

canvasBarMenuItem = new JMenuItem("Canvas",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUNOTCHECKED]));
canvasBarMenuItem.setBackground(new Color(214, 223, 247));
canvasBarMenuItem.setActionCommand("canvasBarMenuItem");

componentBarMenuItem = new JMenuItem("Component",
                          new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUNOTCHECKED]));
componentBarMenuItem.setBackground(new Color(214, 223, 247));
componentBarMenuItem.setActionCommand("componentBarMenuItem");

pageExplorerBarMenuItem = new JMenuItem("Page Explorer",
                            new ImageIcon(ActiveState.
   btnsExited[ActiveState.MENUCHECKED]));
pageExplorerBarMenuItem.setBackground(new Color(214, 223, 247));
pageExplorerBarMenuItem.setActionCommand("pageExplorerBarMenuItem");

statusBarMenuItem = new JMenuItem("Statusbar",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCHECKED]));
statusBarMenuItem.setBackground(new Color(214, 223, 247));
statusBarMenuItem.setActionCommand("statusBarMenuItem");

visitedElementsBarMenuItem = new JMenuItem("Visited Elements",
                               new
                               ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCHECKED]));
visitedElementsBarMenuItem.setBackground(new Color(214, 223, 247));
visitedElementsBarMenuItem.setActionCommand("visitedElementsBarMenuItem");

paintingBarMenuItem = new JMenuItem("Painting",
                          new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCHECKED]));
paintingBarMenuItem.setBackground(new Color(214, 223, 247));
paintingBarMenuItem.setActionCommand("paintingBarMenuItem");

acknowledgmentMenuItem = new JMenuItem("      Acknowledgment");
acknowledgmentMenuItem.setBackground(new Color(214, 223, 247));
acknowledgmentMenuItem.setActionCommand("acknowledgmentMenuItem");
```

```
contactUsMenuItem = new JMenuItem("       Contact us");
contactUsMenuItem.setBackground(new Color(214, 223, 247));
contactUsMenuItem.setActionCommand("contactUsMenuItem");

exitMenuItem = new JMenuItem("       Exit");
exitMenuItem.setBackground(new Color(214, 223, 247));
exitMenuItem.setActionCommand("exitMenuItem");

simulationBarMenuItem = new JMenuItem("Simulation",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUNOTCHECKED]));
simulationBarMenuItem.setBackground(new Color(214, 223, 247));
simulationBarMenuItem.setActionCommand("simulationBarMenuItem");

canvasColorsMenuItem = new JMenuItem("Canvas Colors",
                       new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCANVASCOLORS]));
canvasColorsMenuItem.setBackground(new Color(214, 223, 247));
canvasColorsMenuItem.setActionCommand("canvasColorsMenuItem");

canvasSizeMenuItem = new JMenuItem("Canvas Size",
                      new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCANVASSIZE]));
canvasSizeMenuItem.setBackground(new Color(214, 223, 247));
canvasSizeMenuItem.setActionCommand("canvasSizeMenuItem");

voltageMarkerMenuItem = new JMenuItem("Voltage Marker",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUVOLTAGEMARKER]));
voltageMarkerMenuItem.setBackground(new Color(214, 223, 247));
voltageMarkerMenuItem.setActionCommand("voltageMarkerMenuItem");

diffVoltageMarkerMenuItem = new JMenuItem("Voltage Differential Marker",
                            new
                            ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUDIFFVOLTAGEMARKER]));
diffVoltageMarkerMenuItem.setBackground(new Color(214, 223, 247));
diffVoltageMarkerMenuItem.setActionCommand("diffVoltageMarkerMenuItem");

currentMarkerMenuItem = new JMenuItem("Current Marker",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCURRENTMARKER]));
currentMarkerMenuItem.setBackground(new Color(214, 223, 247));
currentMarkerMenuItem.setActionCommand("currentMarkerMenuItem");
```

```
hideAllMarkersMenuItem = new JMenuItem("       Hide All");
hideAllMarkersMenuItem.setBackground(new Color(214, 223, 247));
hideAllMarkersMenuItem.setActionCommand("hideAllMarkersMenuItem");

showAllMarkersMenuItem = new JMenuItem("       Show All");
showAllMarkersMenuItem.setBackground(new Color(214, 223, 247));
showAllMarkersMenuItem.setActionCommand("showAllMarkersMenuItem");

lastPreviousMenuItem = new JMenuItem("First Page",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENULASTBACKWARD]));
lastPreviousMenuItem.setBackground(new Color(214, 223, 247));
lastPreviousMenuItem.setActionCommand("lastPreviousMenuItem");

previousMenuItem = new JMenuItem("Previous Page",
                       new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUBACKWARD]));
previousMenuItem.setBackground(new Color(214, 223, 247));
previousMenuItem.setActionCommand("previousMenuItem");

nextMenuItem = new JMenuItem("Next Page",
                     new ImageIcon(ActiveState.btnsExited[
                             ActiveState.MENUFORWARD]));
nextMenuItem.setBackground(new Color(214, 223, 247));
nextMenuItem.setActionCommand("nextMenuItem");

lastNextMenuItem = new JMenuItem("Last Page",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENULASTFORWARD]));
lastNextMenuItem.setBackground(new Color(214, 223, 247));
lastNextMenuItem.setActionCommand("lastNextMenuItem");

goToMenuItem = new JMenuItem("Go To ...",
                     new ImageIcon(ActiveState.btnsExited[
                             ActiveState.MENUGOTO]));
goToMenuItem.setBackground(new Color(214, 223, 247));
goToMenuItem.setActionCommand("goToMenuItem");

editPageInfoMenuItem = new JMenuItem("       Edit Page Info...");
editPageInfoMenuItem.setBackground(new Color(214, 223, 247));
editPageInfoMenuItem.setActionCommand("editPageInfoMenuItem");

drawTextMenuItem = new JMenuItem("Text",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUTEXT]));
```

```
drawTextMenuItem.setBackground(new Color(214, 223, 247));
drawTextMenuItem.setActionCommand("drawTextMenuItem");

drawLineMenuItem = new JMenuItem("Line",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENULINE]));
drawLineMenuItem.setBackground(new Color(214, 223, 247));
drawLineMenuItem.setActionCommand("drawLineMenuItem");

drawCircleMenuItem = new JMenuItem("Circle",
                       new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCIRCLE]));
drawCircleMenuItem.setBackground(new Color(214, 223, 247));
drawCircleMenuItem.setActionCommand("drawCircleMenuItem");

drawRectangleMenuItem = new JMenuItem("Rectangle",
                          new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENURECTANGLE]));
drawRectangleMenuItem.setBackground(new Color(214, 223, 247));
drawRectangleMenuItem.setActionCommand("drawRectangleMenuItem");

drawRoundRectangleMenuItem = new JMenuItem("Round Rectangle",
                             new ImageIcon(ActiveState.
   btnsExited[ActiveState.MENUROUNDRECTANGLE]));
drawRoundRectangleMenuItem.setBackground(new Color(214, 223, 247));
drawRoundRectangleMenuItem.setActionCommand("drawRoundRectangleMenuItem");

displayPropertiesMenuItem = new JMenuItem("Display Properties     F3",
                            new
                            ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUDISPLAYPROPERTIES]));
displayPropertiesMenuItem.setBackground(new Color(214, 223, 247));
displayPropertiesMenuItem.setActionCommand("displayPropertiesMenuItem");

editPropertiesMenuItem = new JMenuItem("Edit Attributes           F2",
                         new
                         ImageIcon(ActiveState.btnsExited[
   ActiveState.
   MENUEDITPROPERTIES]));
editPropertiesMenuItem.setBackground(new Color(214, 223, 247));
editPropertiesMenuItem.setActionCommand("editPropertiesMenuItem");

rotateRightMenuItem = new JMenuItem("Rotate Right      Ctrl+R",
                      new
                      ImageIcon(ActiveState.btnsExited[
```

```
                                ActiveState.
                                MENUROTATERIGHT]));
rotateRightMenuItem.setBackground(new Color(214, 223, 247));
rotateRightMenuItem.setActionCommand("rotateRightMenuItem");

rotateLeftMenuItem = new JMenuItem("Rotate Left        Ctrl+L",
                        new
                        ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUROTATELEFT]));
rotateLeftMenuItem.setBackground(new Color(214, 223, 247));
rotateLeftMenuItem.setActionCommand("rotateLeftMenuItem");

flipVerticalMenuItem = new JMenuItem("Flip Vertical       Ctrl+F",
                         new
                         ImageIcon(ActiveState.btnsExited[
   ActiveState.
   MENUFLIPVERTICAL]));
flipVerticalMenuItem.setBackground(new Color(214, 223, 247));
flipVerticalMenuItem.setActionCommand("flipVerticalMenuItem");

flipHorizontalMenuItem = new JMenuItem("Flip Horizontal    Ctrl+H",
                           new
                           ImageIcon(ActiveState.btnsExited[
   ActiveState.
   MENUFLIPHORIZONTAL]));
flipHorizontalMenuItem.setBackground(new Color(214, 223, 247));
flipHorizontalMenuItem.setActionCommand("flipHorizontalMenuItem");

rotateTextMenuItem = new JMenuItem("Rotate Text",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUROTATETEXT]));
rotateTextMenuItem.setBackground(new Color(214, 223, 247));
rotateTextMenuItem.setActionCommand("rotateTextMenuItem");

alignRightMenuItem = new JMenuItem("Align Right     Ctrl+Shift+R",
                        new
                        ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUALIGNRIGHT]));
alignRightMenuItem.setBackground(new Color(214, 223, 247));
alignRightMenuItem.setActionCommand("alignrightMenuItem");

alignLeftMenuItem = new JMenuItem("Align Left       Ctrl+Shift+L",
                       new
```

```
                        ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUALIGNLEFT]));
alignLeftMenuItem.setBackground(new Color(214, 223, 247));
alignLeftMenuItem.setActionCommand("alignLeftMenuItem");

alignTopMenuItem = new JMenuItem("Align Top        Ctrl+Shift+T",
                        new
                        ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUALIGNTOP]));
alignTopMenuItem.setBackground(new Color(214, 223, 247));
alignTopMenuItem.setActionCommand("alignTopMenuItem");

alignBottomMenuItem = new JMenuItem("Align Bottom  Ctrl+Shift+B",
                         new
                         ImageIcon(ActiveState.btnsExited[
                                 ActiveState.
                                 MENUALIGNBOTTOM]));
alignBottomMenuItem.setBackground(new Color(214, 223, 247));
alignBottomMenuItem.setActionCommand("alignBottomMenuItem");

groupMenuItem = new JMenuItem("Group         Shift+G",
                       new ImageIcon(ActiveState.btnsExited[
                                 ActiveState.MENUGROUP]));
groupMenuItem.setBackground(new Color(214, 223, 247));
groupMenuItem.setActionCommand("groupMenuItem");

ungroupMenuItem = new JMenuItem("Ungroup    Shift+U",
                        new ImageIcon(ActiveState.btnsExited[
                                 ActiveState.MENUUNGROUP]));
ungroupMenuItem.setBackground(new Color(214, 223, 247));
ungroupMenuItem.setActionCommand("ungroupMenuItem");

regroupMenuItem = new JMenuItem("Regroup    Shift+R",
                        new ImageIcon(ActiveState.btnsExited[
                                 ActiveState.MENUREGROUP]));
regroupMenuItem.setBackground(new Color(214, 223, 247));
regroupMenuItem.setActionCommand("regroupMenuItem");

createNetlistMenuItem = new JMenuItem(
   "Create Netlist                    F5",
   new
   ImageIcon(ActiveState.btnsExited[
          ActiveState.
```

```
            MENUCREATENETLIST]));
createNetlistMenuItem.setBackground(new Color(214, 223, 247));
createNetlistMenuItem.setActionCommand("createNetlistMenuItem");

editNetlistMenuItem = new JMenuItem(
   "Edit Netlist                          F6",
   new ImageIcon(ActiveState.btnsExited[ActiveState.MENUEDITNETLIST]));
editNetlistMenuItem.setBackground(new Color(214, 223, 247));
editNetlistMenuItem.setActionCommand("editNetlistMenuItem");

simulationSettingsMenuItem = new JMenuItem(
   "Simulation Settings            F7",
   new ImageIcon(ActiveState.btnsExited[ActiveState.MENUSIMULATIONSETTINGS]));
simulationSettingsMenuItem.setBackground(new Color(214, 223, 247));
simulationSettingsMenuItem.setActionCommand("simulationSettingsMenuItem");

runMenuItem = new JMenuItem(
   "Run                                   F9",
   new
   ImageIcon(ActiveState.btnsExited[ActiveState.
            MENURUN]));
runMenuItem.setBackground(new Color(214, 223, 247));
runMenuItem.setActionCommand("runMenuItem");

showNodesNetNamesMenuItem = new JMenuItem("Show Netlist Nodes",
                              new
                              ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUSHOWNODES]));
showNodesNetNamesMenuItem.setBackground(new Color(214, 223, 247));
showNodesNetNamesMenuItem.setActionCommand("showNodesNetNamesMenuItem");

showNodesVoltagesMenuItem = new JMenuItem("Show Nodes Voltages",
                              new
                              ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUSHOWVOLTAGES]));
showNodesVoltagesMenuItem.setBackground(new Color(214, 223, 247));
showNodesVoltagesMenuItem.setActionCommand("showNodesVoltagesMenuItem");

showNodesCurrentsMenuItem = new JMenuItem("Show Nodes Currents",
                              new
                              ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUSHOWCURENTS]));
showNodesCurrentsMenuItem.setBackground(new Color(214, 223, 247));
showNodesCurrentsMenuItem.setActionCommand("showNodesCurrentsMenuItem");
```

```
viewCIRMLMenuItem = new JMenuItem("View CIRML                    F8",
                        new
                        ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUVIEWCIRML]));
viewCIRMLMenuItem.setBackground(new Color(214, 223, 247));
viewCIRMLMenuItem.setActionCommand("viewCIRMLMenuItem");

viewSimulationResultMenuItem = new JMenuItem(
   "View Simulation Results    F10",
   new ImageIcon(ActiveState.btnsExited[ActiveState.
            MENUVIEWSIMULATIONRESULTS]));
viewSimulationResultMenuItem.setBackground(new Color(214, 223, 247));
viewSimulationResultMenuItem.setActionCommand(
   "viewSimulationResultMenuItem");

drawResultMenuItem = new JMenuItem("Draw Result",
                         new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUDRAWRESULT]));
drawResultMenuItem.setBackground(new Color(214, 223, 247));
drawResultMenuItem.setActionCommand("drawResultMenuItem");

showGridMenuItem = new JMenuItem("GRID",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCHECKED]));
showGridMenuItem.setBackground(new Color(214, 223, 247));
showGridMenuItem.setActionCommand("showGridMenuItem");

showConnectionPointsMenuItem = new JMenuItem("Connection Points",
                                new
                                ImageIcon(ActiveState.
   btnsExited[
   ActiveState.MENUCHECKED]));
showConnectionPointsMenuItem.setBackground(new Color(214, 223, 247));
showConnectionPointsMenuItem.setActionCommand(
   "showConnectionPointsMenuItem");

clearAllMenuItem = new JMenuItem("Clear All",
                        new ImageIcon(ActiveState.btnsExited[
   ActiveState.MENUCLEARALL]));
clearAllMenuItem.setBackground(new Color(214, 223, 247));
clearAllMenuItem.setActionCommand("clearAllMenuItem");

copyPageMenuItem = new JMenuItem("       Copy Page");
copyPageMenuItem.setBackground(new Color(214, 223, 247));
```

```
copyPageMenuItem.setActionCommand("copyPageMenuItem");

pastePageMenuItem = new JMenuItem("       Paste Page");
pastePageMenuItem.setBackground(new Color(214, 223, 247));
pastePageMenuItem.setActionCommand("pastePageMenuItem");

selectPageMenuItem = new JMenuItem("       Select Page");
selectPageMenuItem.setBackground(new Color(214, 223, 247));
selectPageMenuItem.setActionCommand("selectPageMenuItem");

findMenuItem = new JMenuItem("Find              Ctrl+F",
                    new ImageIcon(ActiveState.btnsExited[
                                ActiveState.MENUFIND]));
findMenuItem.setBackground(new Color(214, 223, 247));
findMenuItem.setActionCommand("findMenuItem");

replaceMenuItem = new JMenuItem("       Replace...");
replaceMenuItem.setBackground(new Color(214, 223, 247));
replaceMenuItem.setActionCommand("replaceMenuItem");

duplicateMenuItem = new JMenuItem("       Duplicate    Ctrl+D");
duplicateMenuItem.setBackground(new Color(214, 223, 247));
duplicateMenuItem.setActionCommand("duplicateMenuItem");

page1MenuItem = new JMenuItem("Page1",
                    new ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUCHECKED]));
page1MenuItem.setBackground(new Color(214, 223, 247));
page1MenuItem.setActionCommand("page1MenuItem");

page2MenuItem = new JMenuItem("Page2",
                    new ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUCHECKED]));
page2MenuItem.setBackground(new Color(214, 223, 247));
page2MenuItem.setActionCommand("page2MenuItem");

page3MenuItem = new JMenuItem("Page3",
                    new ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUCHECKED]));
page3MenuItem.setBackground(new Color(214, 223, 247));
page3MenuItem.setActionCommand("page3MenuItem");
```

```
    page4MenuItem = new JMenuItem("Page4",
                        new ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUCHECKED]));
    page4MenuItem.setBackground(new Color(214, 223, 247));
    page4MenuItem.setActionCommand("page4MenuItem");

    page5MenuItem = new JMenuItem("Page5",
                        new ImageIcon(ActiveState.btnsExited[
                                ActiveState.
                                MENUCHECKED]));
    page5MenuItem.setBackground(new Color(214, 223, 247));
    page5MenuItem.setActionCommand("page5MenuItem");

    redrawMenuItem = new JMenuItem("       Redraw");
    redrawMenuItem.setBackground(new Color(214, 223, 247));
    redrawMenuItem.setActionCommand("redrawMenuItem");

    fillColorMenuItem = new JMenuItem("Fill Color ",
                        new ImageIcon(ActiveState.btnsExited[
       ActiveState.MENUFILLCOLOR]));
    fillColorMenuItem.setBackground(new Color(214, 223, 247));
    fillColorMenuItem.setActionCommand("fillColorMenuItem");

    lineColorMenuItem = new JMenuItem("Line Color",
                        new ImageIcon(ActiveState.btnsExited[
       ActiveState.MENULINECOLOR]));
    lineColorMenuItem.setBackground(new Color(214, 223, 247));
    lineColorMenuItem.setActionCommand("lineColorMenuItem");

    textColorMenuItem = new JMenuItem("Text Color",
                        new ImageIcon(ActiveState.btnsExited[
       ActiveState.MENUTEXTCOLOR]));
    textColorMenuItem.setBackground(new Color(214, 223, 247));
    textColorMenuItem.setActionCommand("textColorMenuItem");

    //textColorMenuItem=new JMenuItem("Text Color",new
ImageIcon(ActiveState.btnsExited[ActiveState.MENUTEXTCOLOR]));
    //textColorMenuItem.setBackground(new Color(214,223,247));
    //textColorMenuItem.setActionCommand("textColorMenuItem");

    menuBar.add(fileMenu);
    menuBar.add(editMenu);
    menuBar.add(drawMenu);
    menuBar.add(navigateMenu);
```

```
menuBar.add(viewMenu);
menuBar.add(componentsMenu);
menuBar.add(analysisMenu);
menuBar.add(markersMenu);
menuBar.add(optionsMenu);
menuBar.add(helpMenu);

fileMenu.add(newMenuItem);
fileMenu.add(openMenuItem);
fileMenu.add(closePageMenuItem);
fileMenu.addSeparator();
fileMenu.add(saveMenuItem);
fileMenu.add(saveAsMenuItem);
fileMenu.addSeparator();
fileMenu.add(exitMenuItem);

editMenu.add(undoMenuItem);
//editMenu.add(redoMenuItem);
editMenu.addSeparator();
editMenu.add(cutMenuItem);
editMenu.add(copyMenuItem);
editMenu.add(pasteMenuItem);
editMenu.addSeparator();
editMenu.add(deleteMenuItem);
editMenu.addSeparator();
editMenu.add(clearAllMenuItem);
editMenu.add(selectAllMenuItem);
editMenu.add(duplicateMenuItem);
editMenu.addSeparator();
//editMenu.add(findMenuItem);
//editMenu.add(replaceMenuItem);

viewMenu.add(toolsBarMenu);
viewMenu.addSeparator();
//viewMenu.add(zoomingMenu);
viewMenu.addSeparator();
viewMenu.add(showGridMenuItem);
viewMenu.add(showConnectionPointsMenuItem);
viewMenu.addSeparator();
viewMenu.add(redrawMenuItem);

//analysisMenu.add(editNetlistMenuItem);
analysisMenu.add(createNetlistMenuItem);
analysisMenu.add(showNodesNetNamesMenuItem);
analysisMenu.addSeparator();
```

```
analysisMenu.add(simulationSettingsMenuItem);
analysisMenu.add(runMenuItem);
analysisMenu.add(showNodesVoltagesMenuItem);
analysisMenu.add(showNodesCurrentsMenuItem);
analysisMenu.addSeparator();
analysisMenu.add(showNodesVoltagesMenuItem);
analysisMenu.add(showNodesCurrentsMenuItem);
analysisMenu.addSeparator();
analysisMenu.add(viewCIRMLMenuItem);
analysisMenu.add(viewSimulationResultMenuItem);
analysisMenu.add(drawResultMenuItem);

toolsBarMenu.add(standardBarMenuItem);
toolsBarMenu.add(simulationBarMenuItem);
toolsBarMenu.add(canvasBarMenuItem);
toolsBarMenu.add(componentBarMenuItem);
toolsBarMenu.add(paintingBarMenuItem);

toolsBarMenu.add(pageExplorerBarMenuItem);
toolsBarMenu.add(visitedElementsBarMenuItem);
toolsBarMenu.add(statusBarMenuItem);

optionsMenu.add(canvasColorsMenuItem);
optionsMenu.add(canvasSizeMenuItem);
optionsMenu.addSeparator();
optionsMenu.add(fillColorMenuItem);
optionsMenu.add(lineColorMenuItem);
optionsMenu.add(textColorMenuItem);

helpMenu.add(helpMenuItem);
helpMenu.add(acknowledgmentMenuItem);
helpMenu.add(contactUsMenuItem);

markersMenu.add(voltageMarkerMenuItem);
markersMenu.add(diffVoltageMarkerMenuItem);
markersMenu.add(currentMarkerMenuItem);
markersMenu.addSeparator();
markersMenu.add(hideAllMarkersMenuItem);
markersMenu.add(showAllMarkersMenuItem);

navigateMenu.add(lastPreviousMenuItem);
navigateMenu.add(previousMenuItem);
navigateMenu.add(nextMenuItem);
navigateMenu.add(lastNextMenuItem);
navigateMenu.addSeparator();
```

```
navigateMenu.add(refreshMenuItem);
navigateMenu.addSeparator();
navigateMenu.add(selectPageMenuItem);
navigateMenu.add(copyPageMenuItem);
navigateMenu.add(pastePageMenuItem);
navigateMenu.addSeparator();
navigateMenu.add(goToMenuItem);
navigateMenu.add(goToPageMenu);
navigateMenu.addSeparator();
navigateMenu.add(editPageInfoMenuItem);

drawMenu.add(drawTextMenuItem);
drawMenu.add(drawLineMenuItem);
drawMenu.add(drawCircleMenuItem);
drawMenu.add(drawRectangleMenuItem);
drawMenu.add(drawRoundRectangleMenuItem);

componentsMenu.add(displayPropertiesMenuItem);
componentsMenu.add(editPropertiesMenuItem);
componentsMenu.addSeparator();
componentsMenu.add(rotateFlipMenu);
componentsMenu.add(groupMenu);
componentsMenu.add(alignMenu);

//rotateFlipMenu.add(rotateTextMenuItem);
rotateFlipMenu.add(rotateRightMenuItem);
rotateFlipMenu.add(rotateLeftMenuItem);
rotateFlipMenu.addSeparator();
rotateFlipMenu.add(flipVerticalMenuItem);
rotateFlipMenu.add(flipHorizontalMenuItem);

groupMenu.add(groupMenuItem);
groupMenu.add(ungroupMenuItem);
groupMenu.add(regroupMenuItem);

alignMenu.add(alignTopMenuItem);
alignMenu.add(alignRightMenuItem);
alignMenu.add(alignLeftMenuItem);
alignMenu.add(alignBottomMenuItem);

goToPageMenu.add(page1MenuItem);

//zoomingMenu.add(zoomInMenuItem);
//zoomingMenu.add(fitZoomMenuItem);
//zoomingMenu.add(zoomOutMenuItem);
```

```
    setJMenuBar(menuBar);

  }

////////////////////////////////////////////////////////////////////////////
  private void undoActions() {
    restoreOriginalSettings();
  }

////////////////////////////////////////////////////////////////////////////
  private void redoActions() {}

////////////////////////////////////////////////////////////////////////////
  private void deleteComps() {
    saveOriginalSettings();
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getActiveState()) {
          if (allComps[type][numb].getComponentType() == ActiveState.NODEL) {
            allComps[ActiveState.NODER][numb].setActiveState(true);
          }
          else if (allComps[type][numb].getComponentType() == ActiveState.NODER) {
            allComps[ActiveState.NODEL][numb].setActiveState(true);

          }
        }
      }

    }
    deleteComponent(true);
  }

////////////////////////////////////////////////////////////////////////////
  private void clearCanvas() {

    clearAllComps();
    for (int i = 0; i < Array.getLength(allComps); i++) {
      empty[i] = 0;
    }
    drawAllComps();
    drawDrawingComps();
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();
```

```
    }
    if (ActiveState.IsNetlistNodesShown) {
      showNetListNodes();
    }
    if (ActiveState.IsNodesVoltagesShown) {
      showNodesVoltages();
    }
    if (ActiveState.IsNodesCurrentsShown) {
      showNodesCurrents();

    }
  }

////////////////////////////////////////////////////////////////////////////
  private void duplicateComp() {
    copyComps();
    pasteComps();
  }

////////////////////////////////////////////////////////////////////////////
  private void find() {}

////////////////////////////////////////////////////////////////////////////
  private void replace() {}

////////////////////////////////////////////////////////////////////////////

  public void setNameANDValueVisible(boolean name, boolean value) {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {

        if (allComps[type][numb].getComponentType() != ActiveState.CONNECTOR &&
            allComps[type][numb].getComponentType() != ActiveState.GROUND) {
          if (allComps[type][numb].getActiveState()) {
            allComps[type][numb].setNameVisible(name);
            allComps[type][numb].setValueVisible(value);
            if (name & value) {
              allComps[type][numb].setNameANDValueDisplayFormat(3);
            }
            else if (name) {
              allComps[type][numb].setNameANDValueDisplayFormat(2);
            }
            else if (value) {
              allComps[type][numb].setNameANDValueDisplayFormat(1);
            }
```

```
          else {
            allComps[type][numb].setNameANDValueDisplayFormat(0);
          }
        }

      }

    }

  }

////////////////////////////////////////////////////////////////////////////
  public int getNameANDValueDisplayFormat() {
    int displayFormat = 4;
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getComponentType() != ActiveState.CONNECTOR &&
            allComps[type][numb].getComponentType() != ActiveState.GROUND) {
          if (allComps[type][numb].getActiveState()) {
            displayFormat = allComps[type][numb].getNameANDValueDisplayFormat();
          }
        }

      }
    }

    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getComponentType() != ActiveState.CONNECTOR &&
            allComps[type][numb].getComponentType() != ActiveState.GROUND) {
          if (allComps[type][numb].getActiveState()) {
            if (allComps[type][numb].getNameANDValueDisplayFormat() !=
                displayFormat) {
              displayFormat = 4;

            }
          }
        }

      }
    }

    return displayFormat;
```

```
  }

////////////////////////////////////////////////////////////////////////////
  public void rotateComp(int deg) {
    switch (deg) {
      case -90:
        rotateComponent(1);
        break;
      case -180:
        rotateComponent(1);
        rotateComponent(1);
        break;
      case -270:
        rotateComponent(1);
        rotateComponent(1);
        rotateComponent(1);
        break;
      case 90:
        rotateComponent(0);
        break;
      case 180:
        rotateComponent(0);
        rotateComponent(0);
        break;
      case 270:
        rotateComponent(0);
        rotateComponent(0);
        rotateComponent(0);
        break;
    }
    ;
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void alignRight() {
    int xMax = 0;
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getActiveState()) {
          if (allComps[type][numb].getComponentType() == ActiveState.GROUND ||
              allComps[type][numb].getComponentType() == ActiveState.NODEL ||
              allComps[type][numb].getComponentType() == ActiveState.NODER ||
              allComps[type][numb].getComponentType() == ActiveState.CONNODE ||
              allComps[type][numb].getComponentType() ==
```

```
        ActiveState.VOLTAGEMARK ||
        allComps[type][numb].getComponentType() ==
        ActiveState.CURRENTMARK) {
       int[] xyNode1 = allComps[type][numb].getNode(0).getPosition();
       if (xyNode1[0] > xMax) {
        xMax = xyNode1[0];
       }
      }
      else {
       int[] xyNode1 = allComps[type][numb].getNode(1).getPosition();
       int[] xyNode2 = allComps[type][numb].getNode(0).getPosition();
       if (xyNode1[0] > xMax) {
        xMax = xyNode1[0];
       }
       if (xyNode2[0] > xMax) {
        xMax = xyNode2[0];
       }
      }
    }

  }
}

for (int type = 0; type < componentsCount; type++) {
  for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
    if (allComps[type][numb].getActiveState()) {
      if (allComps[type][numb].getComponentType() == ActiveState.CONNECTOR) {
       int[] node1 = allComps[type][numb].getNode(0).getPosition();
       int[] node2 = allComps[type][numb].getNode(1).getPosition();
       if (node1[0] > node2[0]) {
        int xdiff = xMax - node1[0];
        node1[0] = xMax;
        node2[0] += xdiff;

       }
       else {
        int xdiff = xMax - node2[0];
        node2[0] = xMax;
        node1[0] += xdiff;
       }
       allComps[type][numb].getNode(0).setPosition(node1);
       allComps[type][numb].getNode(1).setPosition(node2);
       allComps[type][numb].setNodes();

      }
```

```
        else if (allComps[type][numb].getComponentType() ==
                ActiveState.GROUND ||
                allComps[type][numb].getComponentType() == ActiveState.NODEL ||
                allComps[type][numb].getComponentType() == ActiveState.NODER ||
                allComps[type][numb].getComponentType() ==
                ActiveState.CONNODE ||
                allComps[type][numb].getComponentType() ==
                ActiveState.VOLTAGEMARK ||
                allComps[type][numb].getComponentType() ==
                ActiveState.CURRENTMARK) {
          int[] node = allComps[type][numb].getNode(0).getPosition();
          node[0] = xMax;
          allComps[type][numb].setPosition(node);
          allComps[type][numb].setNodes();

        }
        else {
          int[] pos = allComps[type][numb].getPosition();
          pos[0] = xMax - 25;
          allComps[type][numb].setPosition(pos);
          allComps[type][numb].setNodes();
        }
       }

     }
    }

    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void alignLeft() {
    int xMin = 2500;
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getActiveState()) {
          if (allComps[type][numb].getComponentType() == ActiveState.GROUND ||
              allComps[type][numb].getComponentType() == ActiveState.NODEL ||
              allComps[type][numb].getComponentType() == ActiveState.NODER ||
              allComps[type][numb].getComponentType() == ActiveState.CONNODE ||
              allComps[type][numb].getComponentType() ==
              ActiveState.VOLTAGEMARK ||
              allComps[type][numb].getComponentType() ==
              ActiveState.CURRENTMARK) {
            int[] xyNode1 = allComps[type][numb].getNode(0).getPosition();
```

```
      if (xyNode1[0] < xMin) {
        xMin = xyNode1[0];
      }
    }
    else {
      int[] xyNode1 = allComps[type][numb].getNode(1).getPosition();
      int[] xyNode2 = allComps[type][numb].getNode(0).getPosition();
      if (xyNode1[0] < xMin) {
        xMin = xyNode1[0];
      }
      if (xyNode2[0] < xMin) {
        xMin = xyNode2[0];
      }
    }
  }

 }
}

for (int type = 0; type < componentsCount; type++) {
 for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
  if (allComps[type][numb].getActiveState()) {
    if (allComps[type][numb].getComponentType() == ActiveState.CONNECTOR) {
      int[] node1 = allComps[type][numb].getNode(0).getPosition();
      int[] node2 = allComps[type][numb].getNode(1).getPosition();
      if (node1[0] < node2[0]) {
        int xdiff = node1[0] - xMin;
        node1[0] = xMin;
        node2[0] -= xdiff;

      }
      else {
        int xdiff = node2[0] - xMin;
        node2[0] = xMin;
        node1[0] -= xdiff;
      }
      allComps[type][numb].getNode(0).setPosition(node1);
      allComps[type][numb].getNode(1).setPosition(node2);
      allComps[type][numb].setNodes();

    }
    else if (allComps[type][numb].getComponentType() ==
          ActiveState.GROUND ||
          allComps[type][numb].getComponentType() == ActiveState.NODEL ||
          allComps[type][numb].getComponentType() == ActiveState.NODER ||
```

```
            allComps[type][numb].getComponentType() ==
            ActiveState.CONNODE ||
            allComps[type][numb].getComponentType() ==
            ActiveState.VOLTAGEMARK ||
            allComps[type][numb].getComponentType() ==
            ActiveState.CURRENTMARK) {
          int[] node = allComps[type][numb].getNode(0).getPosition();
          node[0] = xMin;
          allComps[type][numb].setPosition(node);
          allComps[type][numb].setNodes();

        }
        else {
          int[] pos = allComps[type][numb].getPosition();
          pos[0] = xMin + 25;
          allComps[type][numb].setPosition(pos);
          allComps[type][numb].setNodes();
        }
       }

     }
    }

    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void alignTop() {
    int yMin = 2500;
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getActiveState()) {
          if (allComps[type][numb].getComponentType() == ActiveState.GROUND ||
              allComps[type][numb].getComponentType() == ActiveState.NODEL ||
              allComps[type][numb].getComponentType() == ActiveState.NODER ||
              allComps[type][numb].getComponentType() == ActiveState.CONNODE ||
              allComps[type][numb].getComponentType() ==
              ActiveState.VOLTAGEMARK ||
              allComps[type][numb].getComponentType() ==
              ActiveState.CURRENTMARK) {
            int[] xyNode1 = allComps[type][numb].getNode(0).getPosition();
            if (xyNode1[1] < yMin) {
              yMin = xyNode1[1];
            }
          }
```

```
    else {
      int[] xyNode1 = allComps[type][numb].getNode(1).getPosition();
      int[] xyNode2 = allComps[type][numb].getNode(0).getPosition();
      if (xyNode1[1] < yMin) {
        yMin = xyNode1[1];
      }
      if (xyNode2[1] < yMin) {
        yMin = xyNode2[1];
      }
    }
   }

  }
}

for (int type = 0; type < componentsCount; type++) {
  for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
    if (allComps[type][numb].getActiveState()) {
      if (allComps[type][numb].getComponentType() == ActiveState.CONNECTOR) {
        int[] node1 = allComps[type][numb].getNode(0).getPosition();
        int[] node2 = allComps[type][numb].getNode(1).getPosition();
        if (node1[1] < node2[1]) {
          int ydiff = node1[1] - yMin;
          node1[1] = yMin;
          node2[1] -= ydiff;

        }
        else {
          int ydiff = node2[1] - yMin;
          node2[1] = yMin;
          node1[1] -= ydiff;
        }
        allComps[type][numb].getNode(0).setPosition(node1);
        allComps[type][numb].getNode(1).setPosition(node2);
        allComps[type][numb].setNodes();

      }
      else if (allComps[type][numb].getComponentType() ==
              ActiveState.GROUND ||
              allComps[type][numb].getComponentType() == ActiveState.NODEL ||
              allComps[type][numb].getComponentType() == ActiveState.NODER ||
              allComps[type][numb].getComponentType() ==
              ActiveState.CONNODE ||
              allComps[type][numb].getComponentType() ==
              ActiveState.VOLTAGEMARK ||
```

```
            allComps[type][numb].getComponentType() ==
            ActiveState.CURRENTMARK) {
          int[] node = allComps[type][numb].getNode(0).getPosition();
          node[1] = yMin;
          allComps[type][numb].setPosition(node);
          allComps[type][numb].setNodes();

        }
        else {
          int[] pos = allComps[type][numb].getPosition();
          pos[1] = yMin;
          allComps[type][numb].setPosition(pos);
          allComps[type][numb].setNodes();
        }
       }

     }
   }

   redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void alignBottom() {
    int yMax = 0;
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getActiveState()) {
          if (allComps[type][numb].getComponentType() == ActiveState.GROUND ||
              allComps[type][numb].getComponentType() == ActiveState.NODEL ||
              allComps[type][numb].getComponentType() == ActiveState.NODER ||
              allComps[type][numb].getComponentType() == ActiveState.CONNODE ||
              allComps[type][numb].getComponentType() ==
              ActiveState.VOLTAGEMARK ||
              allComps[type][numb].getComponentType() ==
              ActiveState.CURRENTMARK) {
            int[] xyNode1 = allComps[type][numb].getNode(0).getPosition();
            if (xyNode1[1] > yMax) {
              yMax = xyNode1[1];
            }
          }
          else {
            int[] xyNode1 = allComps[type][numb].getNode(1).getPosition();
            int[] xyNode2 = allComps[type][numb].getNode(0).getPosition();
            if (xyNode1[1] > yMax) {
```

```
        yMax = xyNode1[1];
       }
       if (xyNode2[1] > yMax) {
        yMax = xyNode2[1];
       }
      }
     }

   }
}

for (int type = 0; type < componentsCount; type++) {
  for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
    if (allComps[type][numb].getActiveState()) {
      if (allComps[type][numb].getComponentType() == ActiveState.CONNECTOR) {
       int[] node1 = allComps[type][numb].getNode(0).getPosition();
       int[] node2 = allComps[type][numb].getNode(1).getPosition();
       if (node1[1] > node2[1]) {
        int ydiff = node1[1] - yMax;
        node1[1] = yMax;
        node2[1] -= ydiff;

       }
       else {
        int ydiff = node2[1] - yMax;
        node2[1] = yMax;
        node1[1] -= ydiff;
       }
       allComps[type][numb].getNode(0).setPosition(node1);
       allComps[type][numb].getNode(1).setPosition(node2);
       allComps[type][numb].setNodes();

      }
      else if (allComps[type][numb].getComponentType() ==
            ActiveState.GROUND ||
            allComps[type][numb].getComponentType() == ActiveState.NODEL ||
            allComps[type][numb].getComponentType() == ActiveState.NODER ||
            allComps[type][numb].getComponentType() ==
            ActiveState.CONNODE ||
            allComps[type][numb].getComponentType() ==
            ActiveState.VOLTAGEMARK ||
            allComps[type][numb].getComponentType() ==
            ActiveState.CURRENTMARK) {
       int[] node = allComps[type][numb].getNode(0).getPosition();
       node[1] = yMax;
```

```
          allComps[type][numb].setPosition(node);
          allComps[type][numb].setNodes();

        }
        else {
          int[] pos = allComps[type][numb].getPosition();
          pos[1] = yMax;
          allComps[type][numb].setPosition(pos);
          allComps[type][numb].setNodes();
        }
      }

    }
  }

  redrawGridCanvas();
 }

////////////////////////////////////////////////////////////////////////////
 private void goToNextPage() {
  switch (ActiveState.currentPageNumber) {
    case 1:
      activatePage2();
      break;
    case 2:
      activatePage3();
      break;
    case 3:
      activatePage4();
      break;
    case 4:
      activatePage5();
      break;

  }
  ;
 }

////////////////////////////////////////////////////////////////////////////
 private void goToPreviousPage() {
  switch (ActiveState.currentPageNumber) {
    case 2:
      activatePage1();
      break;
    case 3:
```

```
        activatePage2();
        break;
      case 4:
        activatePage3();
        break;
      case 5:
        activatePage4();
        break;
    }
    ;
  }

////////////////////////////////////////////////////////////////////////////
  private void goToFirstPage() {
    activatePage1();

  }

////////////////////////////////////////////////////////////////////////////
  private void goToLastPage() {
    activatePage5();
  }

////////////////////////////////////////////////////////////////////////////
  public void activatePage1() {
    ActiveState.deActivatePages();
    ActiveState.page1Active = true;
    new1Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW1]));
    new2Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW2]));
    new3Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW3]));
    new4Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW4]));
    new5Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW5]));

    savePreviousPage();

    ActiveState.currentPageNumber = 1;

    loadCurrentPage();

    page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUCHECKED]));
    switch (ActiveState.numberOfPages) {
      case 2:
        page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
```

```
      break;
    case 3:
      page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      break;
    case 4:
      page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      break;
    case 5:
      page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      page5MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                            MENUNOTCHECKED]));
      break;
   }
   ;
   if (!pageTitle.equals(new String(""))) {

     drawPageTitle();

   }

  }

////////////////////////////////////////////////////////////////////////////
  public void activatePage2() {
    ActiveState.deActivatePages();
    ActiveState.page2Active = true;
    new2Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW2]));
    new1Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW1]));
    new3Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW3]));
    new4Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW4]));
    new5Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW5]));
```

```
  savePreviousPage();
  ActiveState.currentPageNumber = 2;

  loadCurrentPage();

  page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUCHECKED]));
  switch (ActiveState.numberOfPages) {
    case 2:
      page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      break;
    case 3:
      page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      break;
    case 4:
      page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      break;
    case 5:
      page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      page5MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                              MENUNOTCHECKED]));
      break;
  }
  ;
  if (!pageTitle.equals(new String(""))) {
    drawPageTitle();

  }

}
```

```
////////////////////////////////////////////////////////////////////////////
  public void activatePage3() {
    ActiveState.deActivatePages();
    ActiveState.page3Active = true;
    new3Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW3]));
    new1Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW1]));
    new2Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW2]));
    new4Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW4]));
    new5Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW5]));

    savePreviousPage();
    ActiveState.currentPageNumber = 3;

    loadCurrentPage();

    page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUCHECKED]));
    switch (ActiveState.numberOfPages) {
      case 3:
        page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        break;
      case 4:
        page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
      case 5:
        page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        page5MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                    MENUNOTCHECKED]));
        break;
    }
    ;
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();
```

```
    }

  }

////////////////////////////////////////////////////////////////////////////
  public void activatePage4() {
    ActiveState.deActivatePages();
    ActiveState.page4Active = true;
    new4Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW4]));
    new1Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW1]));
    new2Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW2]));
    new3Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW3]));
    new5Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW5]));

    savePreviousPage();
    ActiveState.currentPageNumber = 4;

    loadCurrentPage();

    page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUCHECKED]));
    switch (ActiveState.numberOfPages) {
      case 4:
        page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUNOTCHECKED]));
        page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUNOTCHECKED]));
        page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUNOTCHECKED]));
      case 5:
        page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUNOTCHECKED]));
        page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUNOTCHECKED]));
        page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUNOTCHECKED]));
        page5MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                                MENUNOTCHECKED]));
        break;
    }
    ;
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();
```

```
    }

  }

////////////////////////////////////////////////////////////////////////////
  public void activatePage5() {
    ActiveState.deActivatePages();
    ActiveState.page5Active = true;
    new5Btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.NEW5]));
    new1Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW1]));
    new2Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW2]));
    new3Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW3]));
    new4Btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.NEW4]));

    savePreviousPage();
    ActiveState.currentPageNumber = 5;

    loadCurrentPage();

    page5MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                          MENUCHECKED]));
    page1MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                          MENUNOTCHECKED]));
    page2MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                          MENUNOTCHECKED]));
    page3MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                          MENUNOTCHECKED]));
    page4MenuItem.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.
                          MENUNOTCHECKED]));
    if (!pageTitle.equals(new String(""))) {
      drawPageTitle();

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void refreshCurrentPage() {
    switch (ActiveState.currentPageNumber) {
      case 1:
        activatePage1();
        break;
      case 2:
        activatePage2();
        break;
```

```
      case 3:
        activatePage3();
        break;
      case 4:
        activatePage4();
        break;
      case 5:
        activatePage5();
        break;
    }
    ;

  }

////////////////////////////////////////////////////////////////////////////
  private void reLocateGUI(int level) {

    switch (level) {
      case 3:
        mainScroll.setLocation(33, 95);
        paintingToolbar.setLocation(2, 310);
        pagesExplorerToolbar.setLocation(2, 100);
        statusPanel.setLocation(35, 596);
        graphicalResult.setLocation(385, 620);
        netlistPanel.setLocation(5, 620);
        visitedElementsPanel.setLocation(634, 110);
        elementsPanel.setLocation(660, 70);
        toolsPanel.setLocation(765, 70);
        fileManagerPanel.setLocation(765, 320);
        simulationPanel.setLocation(765, 620);

        break;
      case 2:
        mainScroll.setLocation(33, 65);
        paintingToolbar.setLocation(2, 280);
        pagesExplorerToolbar.setLocation(2, 70);
        statusPanel.setLocation(35, 566);
        graphicalResult.setLocation(385, 590);
        netlistPanel.setLocation(5, 590);
        visitedElementsPanel.setLocation(634, 80);
        elementsPanel.setLocation(660, 70);
        toolsPanel.setLocation(765, 40);
        fileManagerPanel.setLocation(765, 290);
        simulationPanel.setLocation(765, 590);
```

```
        break;
      case 1:
      case 0:
        mainScroll.setLocation(33, 35);
        paintingToolbar.setLocation(2, 250);
        pagesExplorerToolbar.setLocation(2, 40);
        statusPanel.setLocation(35, 536);
        graphicalResult.setLocation(385, 590);
        netlistPanel.setLocation(5, 590);
        visitedElementsPanel.setLocation(634, 50);
        elementsPanel.setLocation(660, 40);
        toolsPanel.setLocation(765, 40);
        fileManagerPanel.setLocation(765, 290);
        simulationPanel.setLocation(765, 590);

        break;

    }
    ;
    ActiveState.toolBarLevel = level;
  }

////////////////////////////////////////////////////////////////////////////
  private void setNodesVisible(boolean vis) {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {

        allComps[type][numb].setNodevisible(vis);

      }
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void drawPageTitle() {
    gridCanvas.gridImage.setColor(new Color(234, 239, 255));
    gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 250, 40);
    gridCanvas.gridImage.setColor(Color.BLACK);
    gridCanvas.gridImage.drawString("Page Title:  ", startPoint[0] + 10,
                                    startPoint[1] + 15);
    gridCanvas.gridImage.setColor(Color.BLUE);
    gridCanvas.gridImage.drawRect(startPoint[0], startPoint[1], 250, 40);
    gridCanvas.gridImage.drawString(pageTitle, startPoint[0] + 70,
                                    startPoint[1] + 15);
```

```
    Date currentDate = new Date();
    gridCanvas.gridImage.setColor(Color.BLACK);
    gridCanvas.gridImage.drawString(currentDate.toString(), startPoint[0] + 10,
                                    startPoint[1] + 35);
    gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  public void goToPosition(int x, int y) {
    mainScroll.getVerticalScrollBar().setValue(y);
    mainScroll.getHorizontalScrollBar().setValue(x);
    startPoint[0] = x;
    startPoint[1] = y;
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void countNumberofSelectedItems() {
    ActiveState.numbOfSelectedItems = 0;
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getActiveState()) {
          ActiveState.numbOfSelectedItems++;

        }
      }
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void flipVertical() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getActiveState()) {

          if (allComps[type][numb].getAlignment() == 1 ||
              allComps[type][numb].getAlignment() == 3) {
            int[] position = allComps[type][numb].getPosition();
            int oldx = position[0];
            int oldy = position[1];
            int[] oldPos = {
                oldx, oldy};
```

```
          rotateComp(180);

          allComps[type][numb].setPosition(oldPos);
          allComps[type][numb].setNodes();

        }

       }

     }

   }

   redrawGridCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void flipHorizontal() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getActiveState()) {

          if (allComps[type][numb].getAlignment() == 0 ||
              allComps[type][numb].getAlignment() == 2) {
            int[] position = allComps[type][numb].getPosition();
            int oldx = position[0];
            int oldy = position[1];
            int[] oldPos = {
                oldx, oldy};

            rotateComp(180);

            allComps[type][numb].setPosition(oldPos);
            allComps[type][numb].setNodes();

          }

        }

      }

    }

    redrawGridCanvas();
```

```
  }

////////////////////////////////////////////////////////////////////////////
  private void groupComps() {
    int maxX = 0, maxY = 0, minX = 1200, minY = 1200;
    int groupNumb = 0;
    if (ActiveState.multiActive) {
      int length = Array.getLength(groupedComps);

      groupedComps[length - 1] = new Components[componentsCount][];
      for (int type = 0; type < componentsCount; type++) {
        groupedComps[length - 1][type] = ActiveState.createNewElements(type);
      }

      for (int type = 0; type < componentsCount; type++) {
        for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
            numbComp++) {
          if (allComps[type][numbComp].getActiveState()) {
            allComps[type][numbComp].setGroupNumb(length - 1);
            int len = Array.getLength(groupedComps[length - 1][type]);
            groupedComps[length - 1][type][len - 1] = allComps[type][numbComp];
            groupedComps[length -
                1][type] = (Components[]) growArray(groupedComps[length -
                1][type]);
            groupNumb = allComps[type][numbComp].getGroupNumb();
          }
        }
      }
      groupedComps = (Components[][][]) growArray(groupedComps);
      ActiveState.grouped = true;

      for (int cat = 0; cat < componentsCount; cat++) {
        for (int numb = 0; numb < Array.getLength(allComps[cat]) - 1; numb++) {

          if (allComps[cat][numb].isGrouped() &&
              allComps[cat][numb].getGroupNumb() == groupNumb) {

            allComps[cat][numb].setActiveState(true);
            for (int numNode = 0; numNode < allComps[cat][numb].getNumNodes();
                numNode++) {
              int[] POS = allComps[cat][numb].getNode(numNode).getPosition();
              if (POS[0] > maxX) {
                maxX = POS[0];
              }
```

```
        if (POS[0] < minX) {
          minX = POS[0];
        }
        if (POS[1] > maxY) {
          maxY = POS[1];
        }
        if (POS[1] < minY) {
          minY = POS[1];

        }
      }

    }

  }
}

gridCanvas.gridImage.clearRect(startPoint[0], startPoint[1], 650, 500);
gridCanvas.gridImage.setColor(canvasBackgroundColor);
gridCanvas.gridImage.fillRect(startPoint[0], startPoint[1], 650, 500);
gridCanvas.gridImage.setColor(canvasForegroundColor);

gridCanvas.gridImage.setColor(new Color(245, 245, 248));
gridCanvas.gridImage.fillRect(minX - 25, minY - 25, maxX - minX + 50,
                        maxY - minY + 50);
gridCanvas.gridImage.setColor(Color.BLUE);
gridCanvas.gridImage.drawRect(minX - 25, minY - 25, maxX - minX + 50,
                        maxY - minY + 50);
if (ActiveState.showGrid) {
  gridCanvas.drawGridLayout(startPoint);
}
drawAllComps();
drawDrawingComps();
if (!pageTitle.equals(new String(""))) {
  drawPageTitle();
}
if (ActiveState.IsNetlistNodesShown) {
  showNetListNodes();
}
if (ActiveState.IsNodesVoltagesShown) {
  showNodesVoltages();
}
if (ActiveState.IsNodesCurrentsShown) {
  showNodesCurrents();
```

```
      }
      gridCanvas.redraw();

    }

    rotateGroup = true;

  }

////////////////////////////////////////////////////////////////////////////
  private void ungroupComps() {
    if (ActiveState.multiActive) {
      int length = Array.getLength(groupedComps);

      int index = 0;
      boolean flag = false;
      for (int type = 0; type < componentsCount; type++) {
        for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
            numbComp++) {
          if (allComps[type][numbComp].getActiveState()) {

            if (allComps[type][numbComp].isGrouped()) {
              index = allComps[type][numbComp].getGroupNumb();
              allComps[type][numbComp].unGroup();
              flag = true;
            }

          }
        }
      }
      if (flag) {
        groupedComps = (Components[][][]) shrinkArray(groupedComps, index);
      }
      flag = false;

      redrawGridCanvas();
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void regroupComps() {
    ungroupComps();
    groupComps();
  }
```

```
////////////////////////////////////////////////////////////////////////////
  private void deSelectComps() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numbComp = 0; numbComp < Array.getLength(allComps[type]) - 1;
          numbComp++) {
        if (allComps[type][numbComp].getActiveState()) {

          allComps[type][numbComp].setActiveState(false);
          allComps[type][numbComp].setNameState(false);
          allComps[type][numbComp].setValueState(false);

        }
      }
    }
    redrawGridCanvas();
  }

////////////////////////////////////////////////////////////////////////////
  private void copyPage() {
    copiedPage = new WorkPlace(canvasWidth, canvasHeight, canvasBackgroundColor,
                         canvasForegroundColor, "copiedPage", pageTitle);
    copiedPage.saveComponents(allComps, drawComps);

  }

////////////////////////////////////////////////////////////////////////////
  private void pastePage() {

    allComps = copiedPage.getCircuitComponents();
    drawComps = copiedPage.getDrawingComponents();
    updateCanvas(copiedPage.canvasWidth, copiedPage.canvasHeight,
              copiedPage.canvasBackgroundColor,
              copiedPage.canvasForegroundColor);
    pageTitle = copiedPage.getPageTitle();
    redrawGridCanvas();
    copiedPage.resetPage();

  }

////////////////////////////////////////////////////////////////////////////
  private void showDisplayPropertiesFrame() {
    if (ActiveState.active || ActiveState.multiActive) {
      DisplayPropertiesFrame displayProperties = new DisplayPropertiesFrame(this);
      displayProperties.show();
```

```
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void showEditNetlistFrame() {
    NetlistEditorFrame editor = new NetlistEditorFrame(this);
    editor.show();
  }

////////////////////////////////////////////////////////////////////////////
  private void showCreateNetlistFrame() {
    if (generateNetList()) {
      OutputFrame netlistFrame = new OutputFrame("Netlist Display",
                                      netListString);
      netlistFrame.show();
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void simulate() {

    if (generateNetList()) {
      runSimulation();

    }
  }

////////////////////////////////////////////////////////////////////////////
  private void showCIRMLFrame() {
    if (generateNetList()) {

      cirMLString = parseNETLISTToCIRML();

      OutputFrame CIRMLFrame = new OutputFrame("CIRML Display", cirMLString);
      CIRMLFrame.show();
    }
  }

////////////////////////////////////////////////////////////////////////////
  private boolean checkOnlyComponentType(int compType) {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getComponentType() != compType) {
```

```
        return false;
       }
      }

    }
    return true;
  }

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
  private boolean checkSchematicEmpty() {
    int numbOfComps = 0;
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getComponentType() != ActiveState.CONNECTOR &&
            allComps[type][numb].getComponentType() != ActiveState.CONNODE &&
            allComps[type][numb].getComponentType() != ActiveState.NODEL &&
            allComps[type][numb].getComponentType() != ActiveState.NODER &&
            allComps[type][numb].getComponentType() != ActiveState.CURRENTMARK &&
            allComps[type][numb].getComponentType() != ActiveState.VOLTAGEMARK) {

          numbOfComps++;
        }
      }

    }
    if (numbOfComps > 0) {
      return false;
    }
    else {
      return true;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private boolean checkShortCircuit() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        if (allComps[type][numb].getComponentType() != ActiveState.GROUND &&
            allComps[type][numb].getComponentType() != ActiveState.CONNECTOR &&
            allComps[type][numb].getComponentType() != ActiveState.CONNODE &&
            allComps[type][numb].getComponentType() != ActiveState.NODEL &&
            allComps[type][numb].getComponentType() != ActiveState.NODER &&
```

```
          allComps[type][numb].getComponentType() != ActiveState.CURRENTMARK &&
          allComps[type][numb].getComponentType() != ActiveState.VOLTAGEMARK) {

        int[] node1 = allComps[type][numb].getNode(0).getEquivPos();
        int[] node2 = allComps[type][numb].getNode(1).getEquivPos();
        if (node1[0] == node2[0] && node1[1] == node2[1]) {
          return true;

        }
       }

     }

   }
   return false;

  }

////////////////////////////////////////////////////////////////////////////
  private void splitLineByConenctionNode(Components comp) {
   int[][] node = new int[2][2];
   int[] compn1 = comp.getNode(0).getPosition();
   //int[] compn2 = comp.getNode(1).getPosition();
   int[] points = {
      compn1[0], compn1[1]};
   node[0][0] = points[0];
   node[0][1] = points[1];
   //node[1][0] = points[2];
   //node[1][1] = points[3];
   int[] pts = new int[4];
   ActiveState.deletedLines = (Components[]) cutArray(ActiveState.deletedLines);
   ActiveState.splitedLines = (Components[]) cutArray(ActiveState.splitedLines);

   //int maxnumb = Array.getLength(allComps[ActiveState.CONNECTOR]) - 2;
   for (int numb = 0;
       numb < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; numb++) {
    if (!allComps[ActiveState.CONNECTOR][numb].equals(comp)) {
      for (int i = 0; i < 1; i++) {
        int[] point1 = allComps[ActiveState.CONNECTOR][numb].getNode(0).
            getPosition();
        int[] point2 = allComps[ActiveState.CONNECTOR][numb].getNode(1).
            getPosition();
        pts[0] = point1[0];
        pts[1] = point1[1];
        pts[2] = point2[0];
```

```
pts[3] = point2[1];
int a = node[i][0] - pts[0];
int b = node[i][0] - pts[2];

double Y1 = node[i][1] - pts[1];
double X1 = node[i][0] - pts[0];
double Y2 = node[i][1] - pts[3];
double X2 = node[i][0] - pts[2];
double m1 = 0, m2 = 0;
if (X2 != 0) {
  m1 = Y2 / X2;
}
if (X1 != 0) {
  m2 = Y1 / X1;
}
double lineLength = Math.sqrt(Math.pow(pts[1] - pts[3], 2) +
                        Math.pow(pts[0] - pts[2], 2));
double d1 = Math.sqrt(Math.pow(node[i][0] - pts[0], 2) +
                    Math.pow(node[i][1] - pts[1], 2));
double d2 = Math.sqrt(Math.pow(node[i][0] - pts[2], 2) +
                    Math.pow(node[i][1] - pts[3], 2));

if ( (m1 == m2 || (X1 == 0 && X2 == 0)) && (lineLength == d1 + d2)) {
  int len = Array.getLength(ActiveState.splitedLines);
  ActiveState.splitedLines[len -
     1] = ActiveState.createNew(ActiveState.CONNECTOR);
  int[] p1 = {
     pts[0], pts[1], node[i][0], node[i][1]};
  int[] n1 = {
     pts[0], pts[1]};
  int[] n2 = {
     node[i][0], node[i][1]};

  ActiveState.splitedLines[len - 1].getNode(0).setPosition(n1);
  ActiveState.splitedLines[len - 1].getNode(1).setPosition(n2);

  ActiveState.splitedLines[len - 1].setPoints(p1);
  ActiveState.splitedLines[len - 1].setNodes();
  ActiveState.splitedLines[len - 1].setIndex(len - 1);
  ActiveState.splitedLines = (Components[]) growArray(ActiveState.
     splitedLines);
  ActiveState.splitedLines[len] = ActiveState.createNew(ActiveState.
     CONNECTOR);
  int[] p2 = {
     pts[2], pts[3], node[i][0], node[i][1]};
```

```
            int[] l1 = {
               pts[2], pts[3]};
            int[] l2 = {
               node[i][0], node[i][1]};

            ActiveState.splitedLines[len].getNode(0).setPosition(l1);
            ActiveState.splitedLines[len].getNode(1).setPosition(l2);
            ActiveState.splitedLines[len].setPoints(p2);

            ActiveState.splitedLines[len].setNodes();
            ActiveState.splitedLines[len].setIndex(len);
            ActiveState.splitedLines = (Components[]) growArray(ActiveState.
               splitedLines);
            // ActiveState.deletedLines[Array.getLength(ActiveState.deletedLines) -
            //   1] =
            // allComps[ActiveState.CONNECTOR][numb];
            //ActiveState.deletedLines = (Components[]) growArray(ActiveState.
            //  deletedLines);
          }

        }
      }
    }
////////////////////////////////////////////////////////////////////////////

    Vector POINTS = new Vector();
    pts[0] = compn1[0];
    pts[1] = compn1[1];
    //pts[2] = compn2[0];
    //pts[3] = compn2[1];
    POINTS.addElement(compn1);
    for (int numb = 0;
        numb < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; numb++) {
      if (!allComps[ActiveState.CONNECTOR][numb].equals(comp)) {
        int[] n1 = allComps[ActiveState.CONNECTOR][numb].getNode(0).getPosition();
        int[] n2 = allComps[ActiveState.CONNECTOR][numb].getNode(1).getPosition();
        node[0][0] = n1[0];
        node[0][1] = n1[1];
        node[1][0] = n2[0];
        node[1][1] = n2[1];

        for (int i = 0; i < 1; i++) {
          int a = node[i][0] - pts[0];
          int b = node[i][0] - pts[2];
```

```
double Y1 = node[i][1] - pts[1];
double X1 = node[i][0] - pts[0];
double Y2 = node[i][1] - pts[3];
double X2 = node[i][0] - pts[2];
double m1 = 0, m2 = 0;
if (X2 != 0) {
  m1 = Y2 / X2;
}
if (X1 != 0) {
  m2 = Y1 / X1;
}
double lineLength = Math.sqrt(Math.pow(pts[1] - pts[3], 2) +
                        Math.pow(pts[0] - pts[2], 2));
double d1 = Math.sqrt(Math.pow(node[i][0] - pts[0], 2) +
                  Math.pow(node[i][1] - pts[1], 2));
double d2 = Math.sqrt(Math.pow(node[i][0] - pts[2], 2) +
                  Math.pow(node[i][1] - pts[3], 2));

if ( (m1 == m2 || (X1 == 0 && X2 == 0)) && (lineLength == d1 + d2)) {
  int[] p = new int[2];
  p[0] = node[i][0];
  p[1] = node[i][1];
  POINTS.addElement(p);

  /*
      int len=Array.getLength(ActiveState.splitedLines);
      ActiveState.splitedLines[len-1]=ActiveState.createNew(ActiveState.CONNECTOR);
      int[] p1 = {pts[0], pts[1], node[i][0], node[i][1]};
      int[] k1={pts[0],pts[1]};
      int[] k2={node[i][0],node[i][1]};
      ActiveState.splitedLines[len-1].getNode(0).setPosition(k1);
      ActiveState.splitedLines[len-1].getNode(1).setPosition(k2);
      ActiveState.splitedLines[len - 1].setPoints(p1);
      ActiveState.splitedLines[len - 1].setNodes();
      ActiveState.splitedLines[len-1].setIndex(len-1);
      ActiveState.splitedLines = (Components[]) growArray(ActiveState.splitedLines);
      ActiveState.splitedLines[len] = ActiveState.createNew(ActiveState.CONNECTOR);
      int[] p2 = {pts[2], pts[3], node[i][0], node[i][1]};
      int[] l1={pts[2],pts[3]};
      int[] l2={node[i][0],node[i][1]};
      ActiveState.splitedLines[len].getNode(0).setPosition(l1);
      ActiveState.splitedLines[len].getNode(1).setPosition(l2);
      ActiveState.splitedLines[len].setPoints(p2);
    (ActiveState.deletedLines)-1]=comp;
      //allComps[ActiveState.CONNECTOR][comp.getComponentType()];
```

```
            ActiveState.deletedLines=(Components[])  growArray(ActiveState.deletedLines);
          */
        }

      }

    }
  }
  //POINTS.addElement(compn2);
  if (POINTS.size() > 2) {
    for (int i = 0; i < POINTS.size() - 1; i++) {
      int n1[] = new int[2];
      int n2[] = new int[2];

      int[] p = (int[]) POINTS.get(i);

      n1 = (int[]) POINTS.get(i);
      n2 = (int[]) POINTS.get(i + 1);
      int len = Array.getLength(ActiveState.splitedLines);
      ActiveState.splitedLines[len -
         1] = ActiveState.createNew(ActiveState.CONNECTOR);
      ActiveState.splitedLines[len - 1].getNode(0).setPosition(n1);
      ActiveState.splitedLines[len - 1].getNode(1).setPosition(n2);
      int[] p1 = {
         n1[0], n1[1], n2[0], n2[1]};
      ActiveState.splitedLines[len - 1].setPoints(p1);
      ActiveState.splitedLines[len - 1].setNodes();
      ActiveState.splitedLines[len - 1].setIndex(len - 1);
      ActiveState.splitedLines = (Components[]) growArray(ActiveState.
         splitedLines);
    }
    ActiveState.deletedLines[Array.getLength(ActiveState.deletedLines) -
       1] = comp;
    ActiveState.deletedLines = (Components[]) growArray(ActiveState.
       deletedLines);

  }

//.....................................................

  for (int numb = 0; numb < Array.getLength(ActiveState.deletedLines) - 1;
      numb++) {
    for (int n = 0; n < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1;
        n++) {
      if (ActiveState.deletedLines[numb].equals(allComps[ActiveState.
```

```
                                        CONNECTOR][n])) {
          allComps[ActiveState.CONNECTOR] = (Components[]) shrinkArray(
              allComps[ActiveState.CONNECTOR], n);
        }
      }
    }

    for (int numb = 0; numb < Array.getLength(ActiveState.splitedLines) - 1;
        numb++) {
      int len = Array.getLength(allComps[ActiveState.CONNECTOR]);
      allComps[ActiveState.CONNECTOR][len - 1] = ActiveState.splitedLines[numb];
      allComps[ActiveState.CONNECTOR] =
          (Components[]) growArray(allComps[ActiveState.CONNECTOR]);
    }

    for (int i = 0; i < Array.getLength(allComps[ActiveState.CONNECTOR]) - 1; i++) {

      int[] point1 = allComps[ActiveState.CONNECTOR][i].getNode(0).getPosition();
      int[] point2 = allComps[ActiveState.CONNECTOR][i].getNode(1).getPosition();

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void showSimulationSettingsFrame() {
    SimulationSettingsFrame simSettings = new SimulationSettingsFrame(this);
    simSettings.setSize(new Dimension(300, 250));
    simSettings.show();

  }

////////////////////////////////////////////////////////////////////////////
  private void exitApplet() {
    try {

      getAppletContext().showDocument(new URL("Javascript: closePage()"));
    }
    catch (IOException ioe) {

    }
  }

////////////////////////////////////////////////////////////////////////////
  private void rotateText() {}
```

```
////////////////////////////////////////////////////////////////////////////
  private void drawResult() {}

////////////////////////////////////////////////////////////////////////////
  private void showSimulationResultFrame() {
    if (generateNetList()) {
      outputResultFrame = new SimulationResultFrame();
      outputResultFrame.result.setText(outputResult);
      outputResultFrame.show();
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void zoomInCanvas() {}

////////////////////////////////////////////////////////////////////////////
  private void zoomOutCanvas() {}

////////////////////////////////////////////////////////////////////////////
  private void fitZoomCanvas() {}

////////////////////////////////////////////////////////////////////////////
  private void resetActiveComp() {
    ActiveState.CurComp = null;
    redrawGridCanvas();

  }

////////////////////////////////////////////////////////////////////////////
  private void showAllMarkers() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        switch (allComps[type][numb].getComponentType()) {
          case ActiveState.CURRENTCOMP:
          case ActiveState.VOLTAGEMARK:
          case ActiveState.NODEL:
            allComps[type][numb].setVisible(true);
            break;

        }
        ;
      }
      redrawGridCanvas();
```

```
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void hideAllMarkers() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        switch (allComps[type][numb].getComponentType()) {
          case ActiveState.CURRENTCOMP:
          case ActiveState.VOLTAGEMARK:
          case ActiveState.NODEL:
          case ActiveState.NODER:

            if (allComps[type][numb].getControl() == "") {
              allComps[type][numb].setVisible(false);
            }
            break;

        }
        ;
      }
      redrawGridCanvas();
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void resetNodesEquivalentPositions() {
    for (int type = 0; type < componentsCount; type++) {
      for (int numb = 0; numb < Array.getLength(allComps[type]) - 1; numb++) {
        allComps[type][numb].setNodes();
      }
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void resetSimulationParameters() {
    probesString = "";
    probesVector.removeAllElements();
    outputResult = "";
    graphicalResultStr = "";
    cirMLString = "";
    netListString = "";
    netListNodes.removeAllElements();
    ActiveState.IsNetlistNodesShown = false;
```

```
    ActiveState.IsNodesVoltagesShown = false;
    ActiveState.IsNodesCurrentsShown = false;

  }

////////////////////////////////////////////////////////////////////////////
  private void SSSS() {
    for (int i = 0; i < componentsCount; i++) {
      for (int j = 0; j < Array.getLength(allComps[i]) - 1; j++) {

        if (allComps[i][j].getComponentType() == ActiveState.NODEL ||
            allComps[i][j].getComponentType() == ActiveState.NODER ||
            allComps[i][j].getComponentType() == ActiveState.VOLTAGEMARK ||
            allComps[i][j].getComponentType() == ActiveState.CURRENTMARK ||
            allComps[i][j].getComponentType() == ActiveState.GROUND) {
          int[] n = allComps[i][j].getNode(0).getPosition();
          int[] n1 = allComps[i][j].getPosition();
        }
        else if (allComps[i][j].getComponentType() == ActiveState.TRANSFORMERA) {
          int[] n1 = allComps[i][j].getNode(0).getPosition();

          int[] n2 = allComps[i][j].getNode(1).getPosition();

          int[] n3 = allComps[i][j].getNode(2).getPosition();

          int[] n4 = allComps[i][j].getNode(3).getPosition();

          int[] n = allComps[i][j].getPosition();

        }
        else {
          int[] n1 = allComps[i][j].getNode(0).getPosition();
          int[] n2 = allComps[i][j].getNode(1).getPosition();
          int[] n3 = allComps[i][j].getPosition();

        }

      }

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void zoomIn() {
```

```
    Graphics2D zoomIn = (Graphics2D) gridCanvas.gridImage;
    zoomIn.scale(1.5, 1.5);
    redrawGridCanvas();
   }
////////////////////////////////////////////////////////////////////////////
//*************************************************************************/
/
//                        END OF MEMBER FUNCTIONS                        //
//*************************************************************************/
/

}
//*************************************************************************/
/
    /*//////////////////////////////////////////////////////////////////////
**********************************************************
**                  END OF APPLET                       **
**********************************************************
//////////////////////////////////////////////////////////////////////////*/
```

ActiveState.java

```
package ECS;

import java.awt.*;
import java.io.*;

import ECS.CircuitComponents.*;
import ECS.CircuitComponents.Pin1Components.*;
import ECS.CircuitComponents.Pin2Components.*;
import ECS.CircuitComponents.Pin2Components.DependentSources.*;
import ECS.CircuitComponents.Pin2Components.IndependentSources.*;
import ECS.CircuitComponents.Pin2Components.PassiveComponents.*;
import ECS.CircuitComponents.Pin4Components.*;
import ECS.DrawingTools.*;

public class ActiveState {

  public static Components CurComp = null;
```

```
public static boolean active = false;
public static boolean multiActive = false;
public static Components activeComp = null;
public static Components copiedComp = null;
public static Components[][] copiedComps;
public static Components[] splitedLines = null;
public static Components[] deletedLines = null;
public static boolean singleCopied = false;
public static boolean multipleCopied = false;
public static final int UNSELECTED = -1;
public static int number = UNSELECTED;
public static int nType;
public static boolean DELETED = false;
public static boolean Moved = false;
public static boolean Resized = false;
public static double currentXScale = 1.0;
public static double currentYScale = 1.0;
public static boolean undoEnabled = false;
public static Components[][] origComps;
public static boolean nameEditable = false;
public static boolean valueEditable = false;
public static boolean multiMove = false;
public static Components movedComp;
public static boolean grouped = false;
public static boolean disableGroup = false;
public static boolean disableUngroup = false;
public static boolean zoomInCmd = false;
public static boolean zoomOutCmd = false;
public static boolean handCmd = false;
public static boolean simulationCmd = false;
public static boolean defaultCursorCmd = false;
public static int numbOfSelectedItems = 0;
public static boolean pageSaved = false;
public static File pageFile = null;
public static boolean textEntry = false;

public static boolean moving = false;
public static boolean pressing = false;
public static boolean releasing = false;
public static boolean dragging = false;
public static boolean clicking = false;
public static boolean insertTextCmd = false;
public static int activePanel = -1;
public static boolean txtPressed = false;
public static int[] startPoint = new int[2];
```

```
public static int numberOfPages = 1;
public static boolean page1Active = false;
public static boolean page2Active = false;
public static boolean page3Active = false;
public static boolean page4Active = false;
public static boolean page5Active = false;
public static boolean rectangleCmd = false;
public static boolean lineCmd = false;
public static boolean circleCmd = false;
public static boolean roundRectangleCmd = false;
public static boolean textColorCmd = false;
public static boolean arcCmd = false;
public static int activePageNumber = 1;
public static int currentPageNumber = 1;
public static DrawTool drawingCompActive = null;
public static boolean IsNetlistNodesShown = false;
public static boolean IsNodesVoltagesShown = false;
public static boolean IsNodesCurrentsShown = false;
public static boolean selectGroup = false;
public static boolean showGrid = true;
public static boolean showStandardBar = true;
public static boolean showComponentBar = false;
public static boolean showCanvasBar = false;
public static boolean showSimulationBar = false;
public static boolean showPaintingBar = true;
public static boolean showVisitedElementsBar = true;
public static boolean showStatusBar = true;
public static boolean showPageExplorerBar = true;
public static boolean showNodesVisible = true;
public static boolean functionalKey = false;
public static int toolBarLevel = 1;

public static final int RESISTOR = 0;
public static final int INDUCTOR = 1;
public static final int CAPACITOR = 2;
public static final int TRANSFORMER = 3;
public static final int TRANSFORMERA = 4;
public static final int TRANSFORMERB = 5;
public static final int TRANSFORMERC = 6;
public static final int TRANSFORMERD = 7;
public static final int DCCSOURCE = 8;
public static final int DCVSOURCE = 9;
public static final int ACCSOURCE = 10;
public static final int ACVSOURCE = 11;
public static final int DEPCCCS = 12;
```

```
public static final int DEPCCVS = 13;
public static final int DEPVCVS = 14;
public static final int DEPVCCS = 15;
public static final int CURRENTCOMP = 16;
public static final int GROUND = 17;
public static final int CONNODE = 18;
public static final int NODEL = 19;
public static final int NODER = 20;
public static final int CONNECTOR = 21;
public static final int VOLTAGEMARK = 22;
public static final int CURRENTMARK = 23;
public static final int CONNODE1 = 254;
public static final int CONNODE2 = 255;

public static final int PASSIVEELEMENTS = 0;
public static final int INDEPENDENTSOURCES = 1;
public static final int DEPENDENTSOURCES = 2;
public static final int CONNECTIONS = 3;
public static final int OPENSCHEMATIC = 4;
public static final int SAVESCHEMATIC = 5;
public static final int NEWSCHEMATIC = 6;
public static final int CREATENETLIST = 7;
public static final int SIMULATE = 8;
public static final int FILEMANAGER = 9;
public static final int NEW = 10;
public static final int OPEN = 11;
public static final int SAVE = 12;
public static final int CUT = 13;
public static final int COPY = 14;
public static final int PASTE = 15;
public static final int UNDO = 16;
public static final int REMOVE = 17;
public static final int RUN = 18;
public static final int ZOOMIN = 19;
public static final int FITZOOM = 20;
public static final int ZOOMOUT = 21;
public static final int HAND = 22;
public static final int DCURSOR = 23;
public static final int HELP = 24;
public static final int SRESISTOR = 25;
public static final int SINDUCTOR = 26;
public static final int SCAPACITOR = 27;
public static final int STRANSFORMER = 28;
public static final int STRANSFORMERA = 29;
public static final int STRANSFORMERB = 30;
```

```
public static final int STRANSFORMERC = 31;
public static final int STRANFORMERD = 32;
public static final int SDCCSOURCE = 33;
public static final int SDCVSOURCE = 34;
public static final int SACCSOURCE = 35;
public static final int SACVSOURCE = 36;
public static final int SDEPCCCS = 37;
public static final int SDEPCCVS = 38;
public static final int SDEPVCVS = 39;
public static final int SDEPVCCS = 40;
public static final int SCONNECTOR = 41;
public static final int SCURRENTCOMP = 42;
public static final int SGROUND = 43;
public static final int SCONNODE = 44;
public static final int SNODEL = 45;
public static final int NEW1 = 46;
public static final int NEW2 = 47;
public static final int NEW3 = 48;
public static final int NEW4 = 49;
public static final int NEW5 = 50;
public static final int CONNECTFM = 51;
public static final int DISCONNECTFM = 52;
public static final int DOWNLOADFM = 53;
public static final int UPLOADFM = 54;
public static final int MKDIRFM = 55;
public static final int FILERENAMEFM = 56;
public static final int FILEDELETEFM = 57;
public static final int EXITFM = 58;
public static final int TEXT = 59;
public static final int FILLCOLOR = 60;
public static final int LINECOLOR = 61;
public static final int CANVASPROPERTIES = 62;
public static final int NETLIST = 63;
public static final int CIRCLE = 64;
public static final int SQUARE = 65;
public static final int SIMSETTINGS = 66;
public static final int SETTINGS = 67;
public static final int LINE = 68;
public static final int ROUNDRECTANGLE = 69;
public static final int TEXTCOLOR = 70;
public static final int ARC = 71;
public static final int DELETEPAGE = 72;
public static final int SELECTALLCOMPS = 73;
public static final int DELETESCHEMATIC = 74;
public static final int EDITNETLIST = 75;
```

```
public static final int SHOWVOLTAGES = 76;
public static final int SHOWCURRENTS = 77;
public static final int SHOWOUTPUTRESULT = 78;
public static final int SHOWHIDENODES = 79;
public static final int SHOWHIDENODESDISABLE = 80;
public static final int SHOWCIRML = 81;
public static final int VOLTAGEMARKER = 82;
public static final int VOLTAGEDIFFERENTIALMARKER = 83;
public static final int CURRENTMARKER = 84;
public static final int NETLISTEDITING = 85;
public static final int MENUCOPY = 86;
public static final int MENUCUT = 87;
public static final int MENUDELETE = 88;
public static final int MENUEDITPROPERTIES = 89;
public static final int MENUROTATER = 90;
public static final int MENUROTATEL = 91;
public static final int MENUGROUP = 92;
public static final int MENUUNGROUP = 93;
public static final int MENUUNDO = 94;
public static final int MENUPASTE = 95;
public static final int MENUZOOMIN = 96;
public static final int MENUZOOMOUT = 97;
public static final int FORWARD = 98;
public static final int BACKWARD = 99;
public static final int REFRESH = 100;
public static final int DISPLAYPROPERTIES = 101;
public static final int REDO = 102;
public static final int ROTATERIGHT = 103;
public static final int ROTATELEFT = 104;
public static final int FLIPVERTICAL = 105;
public static final int FLIPHORIZANTAL = 106;
public static final int ROTATETEXT = 107;
public static final int GROUP = 108;
public static final int UNGROUP = 109;
public static final int CANVASCOLORS = 110;
public static final int CANVASSIZE = 111;
public static final int SERVERFILEMANAGER = 112;
public static final int ALIGNRIGHT = 113;
public static final int ALIGNLEFT = 114;
public static final int ALIGNUP = 115;
public static final int ALIGNDOWN = 116;
public static final int DRAWRESULT = 117;
public static final int LASTBACKWARD = 118;
public static final int LASTFORWARD = 119;
public static final int GRID = 120;
```

```
public static final int CLEARALLCOMPS = 121;
public static final int CONTACTUS = 122;
public static final int GOTO = 123;
public static final int REGROUP = 124;
public static final int SAVESCHEMATICFM = 125;
public static final int LOADSCHEMATICFM = 126;
public static final int TOOLBAREDITPROPERTIES = 127;
public static final int MENUGOTO = 128;
public static final int MENUNEW = 129;
public static final int MENUOPEN = 130;
public static final int MENUSAVE = 131;
public static final int DELETEPAGEMENU = 132;
public static final int MENUREDO = 133;
public static final int MENUSELECTALL = 134;
public static final int MENUCLEARALL = 135;
public static final int MENUFIND = 136;
public static final int MENUREMOVE = 137;
public static final int MENUTEXT = 138;
public static final int MENULINE = 139;
public static final int MENUCIRCLE = 140;
public static final int MENURECTANGLE = 141;
public static final int MENUROUNDRECTANGLE = 142;
public static final int MENULASTFORWARD = 143;
public static final int MENUFORWARD = 144;
public static final int MENUBACKWARD = 145;
public static final int MENULASTBACKWARD = 146;
public static final int MENUREFRESH = 147;
public static final int MENUFITZOOM = 148;
public static final int MENUROTATERIGHT = 149;
public static final int MENUROTATELEFT = 150;
public static final int MENUFLIPVERTICAL = 151;
public static final int MENUFLIPHORIZONTAL = 152;
public static final int MENUREGROUP = 153;
public static final int MENUALIGNRIGHT = 154;
public static final int MENUALIGNLEFT = 155;
public static final int MENUALIGNTOP = 156;
public static final int MENUALIGNBOTTOM = 157;
public static final int MENUDISPLAYPROPERTIES = 158;
public static final int MENUROTATETEXT = 159;
public static final int MENUCREATENETLIST = 160;
public static final int MENUEDITNETLIST = 161;
public static final int MENUSIMULATIONSETTINGS = 162;
public static final int MENURUN = 163;
public static final int MENUSHOWVOLTAGES = 164;
public static final int MENUSHOWCURENTS = 165;
```

```
public static final int MENUVIEWCIRML = 166;
public static final int MENUVIEWSIMULATIONRESULTS = 167;
public static final int MENUDRAWRESULT = 168;
public static final int MENUVOLTAGEMARKER = 169;
public static final int MENUCURRENTMARKER = 170;
public static final int MENUDIFFVOLTAGEMARKER = 171;
public static final int MENUSHOWNODES = 172;
public static final int MENUCANVASCOLORS = 173;
public static final int MENUCANVASSIZE = 174;
public static final int MENUHELP = 175;
public static final int MENUGRID = 176;
public static final int MENUCHECKED = 177;
public static final int MENUNOTCHECKED = 178;
public static final int MENUFILLCOLOR = 179;
public static final int MENULINECOLOR = 180;
public static final int MENUTEXTCOLOR = 181;
public static final int ARROWUP = 182;
public static final int ARROWDOWN = 183;
public static final int VOLTAGEM = 184;
public static final int CURRENTM = 185;

public static final int ALERT = 0;
public static final int BACKGROUND = 1;
public static final int STATUSBAR = 2;
public static final int QUESTION = 3;
public static final int WELCOMEMESSAGE = 4;
public static final int MSGBACKGROUND = 5;
public static final int QUESTIONBACKGROUND = 6;
public static final int ALERTMESSAGEBACKGROUND = 7;
public static final int ERRORMESSAGEBACKGROUND = 8;
public static final int INFOMESSAGEBACKGROUND = 9;
public static final int PANELBACKGROUND = 10;
public static final int RESULTBACKGROUND = 11;
public static final int SEPARATOR = 12;
public static final int NETLISTPANELBACKGROUND = 13;
public static final int SIMULATIONBACKGROUND = 14;
public static final int ACKNOWLEDGMENT = 15;
public static final int OURCONTACT = 16;

public static final int DEPCOMPS[] = {
   DEPCCCS, DEPCCVS, DEPVCVS, DEPVCCS};

public static final String StrCompGifs[] = {
   "Resistor", "Inductor", "Capacitor", "Transformer", "TransformerA",
   "TransformerB",
```

```
    "TransformerC", "TransformerD", "DCCsource", "DCVsource",
    "ACCsource", "ACVsource", "DepCCCS", "DepCCVS", "DepVCVS", "DepVCCS",
    "CurrentComp",
    "Ground", "Node", "Node+", "Node-", "", "Node+", "Node+",
};

public static final String strBtnsImgs[] = {
    "passiveElements", "independentSources", "dependentSources",
    "connections",
    "openSchematic", "saveSchematic", "newSchematic",
    "createNetlist", "simulate", "fileManager", "new", "open", "save", "cut",
    "copy", "paste"
    , "undo", "remove", "run", "zoomIn", "fitZoom", "zoomOut", "hand",
    "dCursor", "help",
    "SResistor", "SInductor", "SCapacitor",
    "STransformer", "STransformerA", "STransformerB",
    "STransformerC", "STransformerD", "SDCCsource", "SDCVsource",
    "SACCsource", "SACVsource", "SDepCCCS", "SDepCCVS", "SDepVCVS",
    "SDepVCCS", "SConnector"
    , "SCurrentComp", "SGround", "SConNode", "SNodeL", "NEW1", "NEW2", "NEW3",
    "NEW4", "NEW5",
    "connectFM", "disconnectFM", "downloadFM", "uploadFM", "mkDirFM",
    "renameFileFM",
    "deleteFileFM", "exitFM", "text", "fillColor"
    , "lineColor", "canvasProperties", "netList", "circle", "square",
    "simSettings", "settings", "line", "roundRectangle",
    "textColor", "arc", "deletePage", "selectAllComps", "deleteSchematic",
    "editNetlist", "showVoltages", "showCurrents",
    "showOutputResult", "showHideNodes", "showHideNodes_disable", "showCIRML",
    "voltageMarker",
    "voltageDifferentialMarker", "currentMarker", "netlistEditing",
    "menuCopy", "menuCut", "menuDelete", "editPropertiesMenu", "menuRotateR",
    "menuRotateL",
    "groupMenu", "ungroupMenu", "undoMenu", "menuPaste", "ZoomInMenu",
    "zoomOutMenu",
    "forward", "backward", "refresh", "displayProperties", "rendo",
    "rotateRight", "rotateLeft", "flipHorizontal", "flipVertical",
    "rotateText", "group",
    "ungroup", "canvasColors", "canvasSize", "serverFileManager",
    "alignRight", "alignLeft",
    "alignUp", "alignDown", "drawResult", "lastBackward", "lastForward",
    "grid", "clearAllComps",
    "contactus", "goTO", "reGroup", "saveSchematicFM", "loadSchematicFM",
    "menuEnter", "menuGoTo",
    "newMenu", "openMenu", "saveMenu", "deletePageMenu", "redoMenu",
```

```
    "selectAllMenu", "clearAllMenu",
    "findMenu", "removeMenu", "textMenu", "lineMenu", "circleMenu",
    "rectangleMenu", "roundRectangleMenu",
    "lastForwardMenu", "forwardMenu", "backwardMenu", "lastBackwardMenu",
    "refreshMenu", "fitZoomMenu",
    "rotateRightMenu", "rotateLeftMenu", "flipVerticalMenu",
    "flipHorizontalMenu", "regroupMenu",
    "alignRightMenu", "alignLeftMenu", "alignUpMenu", "alignDownMenu",
    "displayPropertiesMenu",
    "rotateTextMenu", "createNetlistMenu", "editNetlistMenu",
    "simulationSettingsMenu", "runMenu",
    "showVoltagesMenu", "showCurrentsMenu", "viewCIRMLMenu",
    "showOutputResultMenu", "drawResultMenu",
    "voltageMarkerMenu", "currentMarkerMenu", "voltageDifferentialMarkerMenu",
    "showNodesMenu", "canvasColorsMenu",
    "canvasSizeMenu", "helpMenu", "gridMenu", "checkedMenu", "notCheckedMenu",
    "fillColorMenu", "lineColorMenu",
    "textColorMenu", "arrowUp", "arrowDown", "voltageM", "currentM"

};

public static final String Images[] = {
    "alert", "background", "statusbar", "question", "welcomeMsg",
    "msgBackground", "questionBackground"
    , "alertMessageBackground", "errorMessageBackground",
    "infoMessageBackground", "panelBackground",
    "resultBackground", "separator", "netlistPanelBackground",
    "simulationBackground", "acknowledgment", "ourContact"
};

public static Image[] ImageNormal, ImageActive;

public static Image[] btnsEntered, btnsExited;
public static Image[] diffImages;

public static Components[] createNewElements(int index) {
  switch (index) {
    case RESISTOR:
      return new Resistor[1];
    case INDUCTOR:
      return new Inductor[1];
    case CAPACITOR:
      return new Capacitor[1];
    case TRANSFORMER:
      return new Transformer[1];
```

```
    case TRANSFORMERA:
      return new PPTransformer[1];
    case TRANSFORMERB:
      return new PNTransformer[1];
    case TRANSFORMERC:
      return new NPTransformer[1];
    case TRANSFORMERD:
      return new NNTransformer[1];
    case DCCSOURCE:
      return new DCCSource[1];
    case DCVSOURCE:
      return new DCVSource[1];
    case ACCSOURCE:
      return new ACCSource[1];
    case ACVSOURCE:
      return new ACVSource[1];
    case DEPCCCS:
      return new DepCCCS[1];
    case DEPCCVS:
      return new DepCCVS[1];
    case DEPVCVS:
      return new DepVCVS[1];
    case DEPVCCS:
      return new DepVCCS[1];
    case CURRENTCOMP:
      return new CurrentComp[1];
    case GROUND:
      return new Ground[1];
    case CONNODE:
      return new ConnectionNode[1];
    case NODEL:
      return new NodeL[1];
    case NODER:
      return new NodeR[1];
    case CONNECTOR:
      return new ConnectionLine[1];
    case VOLTAGEMARK:
      return new VoltageMarker[1];
    case CURRENTMARK:
      return new CurrentMarker[1];
    default:
  }
  return null;
}
```

```
public static Components createNew(int index) {
  switch (index) {
    case RESISTOR:
      return new Resistor();
    case INDUCTOR:
      return new Inductor();
    case CAPACITOR:
      return new Capacitor();
    case TRANSFORMER:
      return new Transformer();
    case TRANSFORMERA:
      return new PPTransformer();
    case TRANSFORMERB:
      return new PNTransformer();
    case TRANSFORMERC:
      return new NPTransformer();
    case TRANSFORMERD:
      return new NNTransformer();
    case DCCSOURCE:
      return new DCCSource();
    case DCVSOURCE:
      return new DCVSource();
    case ACCSOURCE:
      return new ACCSource();
    case ACVSOURCE:
      return new ACVSource();
    case DEPCCCS:
      return new DepCCCS();
    case DEPCCVS:
      return new DepCCVS();
    case DEPVCVS:
      return new DepVCVS();
    case DEPVCCS:
      return new DepVCCS();
    case CURRENTCOMP:
      return new CurrentComp();
    case GROUND:
      return new Ground();
    case CONNODE:
      return new ConnectionNode();
    case NODEL:
      return new NodeL();
    case NODER:
      return new NodeR();
    case CONNECTOR:
```

```
      return new ConnectionLine();
    case VOLTAGEMARK:
      return new VoltageMarker();
    case CURRENTMARK:
      return new CurrentMarker();
    default:
   }
  return null;
 }

 public static void reSet()

 {
  zoomInCmd = false;
  zoomOutCmd = false;
  handCmd = false;
  defaultCursorCmd = false;
  insertTextCmd = false;
  circleCmd = false;
  rectangleCmd = false;
  lineCmd = false;
 }

 public static void resetGridMouseAction() {
  moving = false;
  pressing = false;
  releasing = false;
  dragging = false;
  clicking = false;
 }

 public static void deActivatePages() {
  page1Active = false;
  page2Active = false;
  page3Active = false;
  page4Active = false;
  page5Active = false;
 }

}
```

CompsStack.java

```
package ECS;

import java.lang.reflect.*;
import java.util.*;

import ECS.CircuitComponents.*;

public class CompsStack {

  public int top = 0;
  private Vector stack = new Vector(10);

  public CompsStack() {

  }

  public void push(Components[][] comps) {
    if (top <= 9) {

      stack.add(top, comps);

      for (int i = 0; i < stack.size(); i++) {

        Components[][] comp = (Components[][]) stack.get(i);
        for (int type = 0; type < ActiveState.CONNECTOR + 1; type++) {
          for (int numbComp = 0; numbComp < Array.getLength(comp[type]) - 1;
              numbComp++) {

          }
        }

      }
      for (int i = 0; i < stack.size(); i++) {

        for (int type = 0; type < ActiveState.CONNECTOR + 1; type++) {
          for (int numbComp = 0; numbComp < Array.getLength(comps[type]) - 1;
              numbComp++) {

          }
```

```
        }

      }

      top++;
    }
    else {
      for (int i = 0; i < stack.size() - 1; i++) {
        stack.setElementAt(stack.get(i + 1), i);
      }
      stack.remove(stack.size() - 1);
      stack.setSize(stack.size() - 1);
      top = 0;
      stack.add(top, comps);

    }
  }

  public Components[][] pop() {
    if (top > 0) {
      top--;

    }
    for (int i = 0; i < stack.size(); i++) {

      Components[][] comp = (Components[][]) stack.get(i);
      for (int type = 0; type < ActiveState.CONNECTOR + 1; type++) {
        for (int numbComp = 0; numbComp < Array.getLength(comp[type]) - 1;
            numbComp++) {

        }
      }

    }

    return (Components[][]) stack.get(top);

  }

}
```

NeighbourComp.java

```
package ECS;

public class NeighbourComp {
  public int compType;
  public int compNumb;
  public int nodeNumb1, nodeNumb2;

  public NeighbourComp(int type, int numb, int nNumb1, int nNumb2) {
    compType = type;
    compNumb = numb;
    nodeNumb1 = nNumb1;
    nodeNumb2 = nNumb2;
  }

}
```

NetNode.java

```
 package ECS;

import java.awt.*;

public class NetNode {
  public Point pos;
  public int index;
  public String netName;
  public int hit = 1;
  public NetNode(Point p, String net, int x) {
    pos = p;
    index = x;
    netName = net;
  }

  public void Increment() {
    hit++;
  }
}
```

Probe.java

```
package ECS;

import java.awt.*;

public class Probe {
  public String probeString;
  public Color probeColor;
  public Probe(String str, Color c) {
    probeString = str;
    probeColor = c;
  }

}
```

WorkPlace.java

```
 package ECS;

import java.lang.reflect.*;

import java.io.*;
import java.util.*;

import java.awt.*;
import javax.swing.*;

import java.awt.*;

import ECS.CircuitComponents.*;
import ECS.DrawingTools.*;

public class WorkPlace
    implements Serializable {
```

```
public String name;
public boolean pageSaved = false;
public File pageFile = null;

public String pageTitle = "";

public Components[][] circuitComps;
public DrawTool[] drawComps;
public int canvasWidth = 2500, canvasHeight = 2500;
public Color canvasBackgroundColor = Color.WHITE;
public Color canvasForegroundColor = Color.BLUE;
public WorkPlace(int w, int h, Color backColor, Color foreColor, String nm,
            String pgTitle) {
  name = nm;
  circuitComps = new Components[mainApplet.componentsCount][];
  drawComps = new DrawTool[1];
  for (int i = 0; i < mainApplet.componentsCount; i++) {
    circuitComps[i] = ActiveState.createNewElements(i);
  }
  canvasWidth = w;
  canvasHeight = h;
  canvasBackgroundColor = backColor;
  canvasForegroundColor = foreColor;
  pageTitle = pgTitle;
}

public void saveComponents(Components[][] circuit, DrawTool[] draw) {
  circuitComps = (Components[][]) circuit.clone();
  drawComps = (DrawTool[]) draw.clone();

}

public Components[][] getCircuitComponents() {
  return circuitComps;
}

public DrawTool[] getDrawingComponents() {
  return drawComps;
}

public void deleteComponents() {
  for (int type = 0; type < mainApplet.componentsCount; type++) {
    circuitComps[type] = (Components[]) mainApplet.cutArray(circuitComps[type]);
  }
```

```
  drawComps = (DrawTool[]) mainApplet.cutArray(drawComps);
}

public void saveCanvasWidth(int w) {
  canvasWidth = w;
}

public void saveCanvasHeight(int h) {
  canvasHeight = h;
}

public void saveCanvasBackgroundColor(Color back) {
  canvasBackgroundColor = back;
}

public void saveCanvasForegroundColor(Color fore) {
  canvasForegroundColor = fore;
}

public int getCanvasWidth() {
  return canvasWidth;
}

public int getCanvasHeight() {
  return canvasHeight;
}

public Color getCanvasBackgroundColor() {
  return canvasBackgroundColor;
}

public Color getCanvasForegroundColor() {
  return canvasForegroundColor;
}

public void savePageTitle(String title) {
  pageTitle = title;
}

public String getPageTitle() {
  return pageTitle;
}

public void resetPage() {
  canvasWidth = 2500;
```

```
    canvasHeight = 2500;
    canvasBackgroundColor = Color.WHITE;
    canvasForegroundColor = Color.BLUE;
    pageTitle = "";
    pageSaved = false;
    pageFile = null;
    deleteComponents();
  }

}
```

Components.java

```
package ECS.CircuitComponents;

import java.io.*;
import java.util.*;

import java.awt.*;
import javax.swing.*;

import ECS.GUPanels.*;
import ECS.ActiveState;

public abstract class Components
    implements Serializable {

  private int[] XYpos = new int[2];
  private int[] zoomXYPos = new int[2];
  private int[] namePos = new int[2];
  private int[] valuePos = new int[2];

  private boolean Active = false, nameActive = false, valueActive = false;
  private int Alignment = 0;
  private String ComponentName;
  private int[] points;
```

```
  private double n1 = 1.0, n2 = 1.0;
  private boolean grouped = false;
  private int groupNumb = 0;
  private int displayFormat = 3;
  private boolean nameVisible = true;
  private boolean valueVisible = true;
  private boolean nodeVisible = true;
  private boolean compVisible = true;
  private boolean isCosine = false;

  double Value = 0.0, Frequency, Phase;
  int count = 0, coordinateNumber = -1;

  protected int nIndex = -1;
  protected int order;
  public String Unit, FreqUnit, PhaseUnit, Control = "";
  public JTextField txtValue, txtName;
  public JPanel valueUnitPanel;
  public JComboBox units;
  public Vector NeighbourComps = new Vector();
  public boolean rotated = false;
  public boolean rotatedFinished = false;
  public boolean done = false;
  public boolean deleted = false;

////////////////////////////////////////////////////////////////////////////
  public abstract Node getNode(int nIndex);

  public abstract void setNodes();

  public abstract int getNumNodes();

  public abstract String getGenericName();

  public abstract String getName();

  public abstract int getComponentType();

  public abstract String[] getAllUnits();

  public abstract String[] getAllInputUnits();

  public abstract String getDefaultUnit();

  public abstract String getImageName();
```

```
  public abstract String getNetListString();

  public abstract String getCIRMLString();

  public abstract void draw(GridCanvas gridcanvas, Component comp);

  public abstract boolean occupiedArea(int[] pos);

  public abstract boolean mouseOverArea(int[] pos);

  public abstract boolean occupiedNameArea(int[] pos);

  public abstract boolean occupiedValueArea(int[] pos);

  public abstract boolean mouseOverNameArea(int[] pos);

  public abstract boolean mouseOverValueArea(int[] pos);

////////////////////////////////////////////////////////////////////////////
  public void setCount(int n) {
    count = n;
  }

  public int getCount() {
    return count;
  }

  public void setNameVisible(boolean vis) {
    nameVisible = vis;
  }

  public boolean getNameVisible() {
    return nameVisible;
  }

  public void setValueVisible(boolean vis) {
    valueVisible = vis;
  }

  public boolean getValueVisible() {
    return valueVisible;
  }

  public void setNameANDValueDisplayFormat(int f) {
```

```
  displayFormat = f;
}

public int getNameANDValueDisplayFormat() {
  return displayFormat;
}

public void setNodevisible(boolean vis) {
  nodeVisible = vis;
}

public boolean getNodeVisible() {
  return nodeVisible;
}

public void setCosine(boolean a) {
  isCosine = a;
}

public boolean isCosine() {
  return isCosine;
}

public void setCoordinateNumber(int n) {
  coordinateNumber = n;
}

public int getCoordinateNumber() {
  return coordinateNumber;
}

public void setControl(String cr) {
  Control = cr;
}

public String getControl() {
  return Control;
}

public void setFreqUnit(String unitv) {
  FreqUnit = unitv;
}

public String getFreqUnit() {
  if (FreqUnit != null) {
```

```
    return FreqUnit;
  }
  else {
    return "HZ";
  }
}

public void setValue(double val) {
  Value = val;
}

public double getValue() {
  return Value;
}

public void setFrequency(double freqv) {
  Frequency = freqv;
}

public double getFrequency() {
  return Frequency;
}

public void setPhase(double phasv) {
  Phase = phasv;
}

public double getPhase() {
  return Phase;
}

public void setPhaseUnit(String punit) {
  PhaseUnit = punit;
}

public String getPhaseUnit() {
  if (PhaseUnit != null) {
    return PhaseUnit;
  }
  else {
    return "deg";
  }
}

public void setVisible(boolean vis) {
```

```
  compVisible = vis;
}

public boolean isVisible() {
  return compVisible;
}

public void setUnit(String unitv) {
  Unit = unitv;
}

public String getUnit() {
  if (Unit != null) {
    return Unit;
  }
  else {
    return getDefaultUnit();
  }
}

public void setOrder(int n) {
  order = n;
}

public void setComponentName(String comName) {
  ComponentName = comName;
}

public String getComponentName() {
  return ComponentName;
}

public void setAlignment(int align) {
  Alignment = align;
}

public int getAlignment() {
  return Alignment;
}

public void setPosition(int[] pos) {
  XYpos[0] = pos[0];
  XYpos[1] = pos[1];

}
```

```
public int[] getPosition() {
  return XYpos;
}

public int[] getZoomPosition() {
  return zoomXYPos;
}

public void setZoomPosition(int[] pos) {
  pos[0] = zoomXYPos[0];
  pos[1] = zoomXYPos[1];
}

public void setActiveState(boolean state) {
  Active = state;
}

public boolean getActiveState() {
  return Active;
}

public void setValueState(boolean state) {
  valueActive = state;
}

public boolean getValueState() {
  return valueActive;
}

public void setNameState(boolean state) {
  nameActive = state;
}

public boolean getNameState() {
  return nameActive;
}

public void setIndex(int indexv) {
  nIndex = indexv;
}

public int getIndex() {
  return nIndex;
}
```

```
public void setPoints(int[] p) {
  points = p;
}

public int[] getPoints() {
  return points;
};
public void setN1(double n) {
  n1 = n;
}

public void setN2(double n) {
  n2 = n;
}

public double getN1() {
  return n1;
}

public double getN2() {
  return n2;
}

public void setNamePos(int[] pos) {
  namePos[0] = pos[0];
  namePos[1] = pos[1];
}

public int[] getNamePos() {
  return namePos;
}

public void setValuePos(int[] pos) {
  valuePos[0] = pos[0];
  valuePos[1] = pos[1];
}

public int[] getValuePos() {
  return valuePos;
}

public int getGroupNumb() {
  return groupNumb;
}
```

```
  public void setGroupNumb(int numb) {
    grouped = true;
    groupNumb = numb;
  }

  public boolean isGrouped() {
    return grouped;
  }

  public void unGroup() {
    grouped = false;
  }
}
```

CompPos.java

```
 package ECS.CircuitComponents;

public class CompPos {
  public int xPos;
  public int yPos;
  public int index;

  public CompPos(int xArg, int yArg, int indexArg) {
    xPos = xArg;
    yPos = yArg;
    index = indexArg;
  }

}
```

Current.java

```
 package ECS.CircuitComponents;

import java.awt.*;
import java.io.*;

public class Current implements Serializable{
  private final String[] units = {
      "A", "mA", "uA", "kA"};
  private double current;
  private String direction;

  public String[] getAllUnits() {
    return units;
  }

  public String getDefaultUnit() {
    return units[0];
  }

  public void setCurrent(double curr) {
    current = curr;
  }

  public double getCurrent() {
    return current;
  }

  public void setDirection(String dir) {
    direction = dir;
  }

  public String getDirection() {
    return direction;
  }

}
```

ImgComponents.java

```
 package ECS.CircuitComponents;
```

```
import java.awt.*;
import java.awt.image.*;

abstract public class ImgComponents
    extends Components {

  public static Image[] imageNormal, imageActive;

  public Image getImageNormal() {
    return imageNormal[getComponentType()];
  }

  public Image getImageActive() {
    return imageActive[getComponentType()];
  }

  public Image getImageByState() {
    if (getActiveState()) {
      return getImageActive();
    }
    else {
      return getImageNormal();
    }
  }

  static public Image rotateImage(Image img, int nr, Component form) {
    int width, height;
    Image draw_image = img;

    for (int l = 0; l < nr; l++) {
      PixelGrabber grabber = new PixelGrabber(draw_image, 0, 0, -1, -1, true);

      try {
        if (grabber.grabPixels()) {
          width = grabber.getWidth();
          height = grabber.getHeight();
          int[] pixels = (int[]) grabber.getPixels();
          int[] transposedPixels = new int[height * width];
          for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
              {
                transposedPixels[ (height - 1 - y) +
                    x * height] = pixels[y * width + x];
              }
```

```
          }
         }
         draw_image = form.createImage(new MemoryImageSource(width, height,
            transposedPixels, 0, height));
       }
      }
      catch (InterruptedException ie) {}
    }
    return draw_image;
  }
}
```

Node.java

```
 package ECS.CircuitComponents;

import java.awt.*;

import ECS.*;
import ECS.GUPanels.*;

public class Node
    extends ImgComponents {
  private static final String[] units = {
      "V"};
  protected int[] EquivPos = new int[2];
  public String strName;

  public Node() {

  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return units;
  }

  public String getDefaultUnit() {
    return units[0];
```

```
}

public String getName() {
  return "Node";
}

public String getImageName() {
  return "";
}

public String getGenericName() {
  return "";
}

public int getComponentType() {
  return ActiveState.CONNODE;
}

public Node getNode(int index) {
  if (index == 0) {
    return this;
  }
  else {
    return null;
  }
}

public int getNumNodes() {
  return 1;
}

public void setNodes() {
  ;
}

public String getNetListString() {
  return "";
}

public String getCIRMLString() {
  return "";
}

public void draw(GridCanvas gridcanvas, Component comp) {
  ;
```

```
}

public boolean occupiedArea(int[] pos) {
  int[] xypos = getPosition();
  if ( (pos[0] > xypos[0] - 10) & (pos[0] < xypos[0] + 10)
      & (pos[1] > xypos[1] - 10) & (pos[1] < xypos[1] + 10)) {
    setActiveState(true);
    return true;
  }
  else {
    setActiveState(false);
    return false;
  }
}

public boolean mouseOverArea(int[] pos) {
  int[] xypos = getPosition();
  if ( (pos[0] > xypos[0] - 10) & (pos[0] < xypos[0] + 10)
      & (pos[1] > xypos[1] - 10) & (pos[1] < xypos[1] + 10)) {
    setActiveState(true);
    return true;
  }
  else {
    setActiveState(false);
    return false;
  }
}

public void setEquivPos(int[] argPos) {
  EquivPos[0] = argPos[0];
  EquivPos[1] = argPos[1];
}

public int[] getEquivPos() {
  return EquivPos;
}

public boolean occupiedValueArea(int[] pos) {

  return false;
}

public boolean occupiedNameArea(int[] pos) {

  return false;
```

```
  }

  public boolean mouseOverNameArea(int[] pos) {
    return false;
  }

  public boolean mouseOverValueArea(int[] pos) {
    return false;
  }
}
```

Pin1Comp.java

```
 package ECS.CircuitComponents;

import java.awt.*;

import ECS.GUPanels.*;

abstract public class Pin1Comp
    extends ImgComponents {

  Node[] nodes;

  public Pin1Comp() {
    nodes = new Node[1];
    nodes[0] = new Node();
  }

  public final Node getNode(int index) {
    if (index < 1) {
      return nodes[index];
    }
    else {
      return null;
    }
```

```
}

public int getNumNodes() {
  return 1;
}

/*   public final void setNodes()
   {
   int[] xy=getPosition();
   xy[0]-=25;
   getNode(0).setPosition(xy);
   getNode(0).setEquivPos(xy);
   int[] k=getNode(0).getEquivPos();
   System.out.println("HAYYYYY   "+k[0]+","+k[1]);
   xy[0]+=25;
   }*/

public void draw(GridCanvas gridCanvas, Component comp) {
  int[] coordinates = getPosition();
  gridCanvas.gridImage.drawImage(rotateImage(getImageByState(), getAlignment(),
                                      comp)
                              , coordinates[0], coordinates[1], comp);
  gridCanvas.gridImage.drawString(getName(), coordinates[0],
                               coordinates[1] - 5);
}

public boolean occupiedArea(int[] pos) {
  int[] xypos = getPosition();

  if ( (pos[0] > xypos[0]) & (pos[0] < xypos[0] + 25) &
      (pos[1] > xypos[1] - 25) & (pos[1] < xypos[1])) {
    setActiveState(true);
    return true;
  }
  else {
    setActiveState(false);
  }
  return false;
}

public boolean mouseOverArea(int[] pos) {
  int[] xypos = getPosition();

  if ( (pos[0] > xypos[0]) & (pos[0] < xypos[0] + 25) &
      (pos[1] > xypos[1] - 25) & (pos[1] < xypos[1])) {
```

```
      return true;
    }
    else {

      return false;
    }
  }

  public boolean mouseOverNameArea(int[] pos) {
    return false;
  }

  public boolean mouseOverValueArea(int[] pos) {
    return false;
  }

}
```

Pin2Comp.java

```
 package ECS.CircuitComponents;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

import ECS.*;
import ECS.GUPanels.*;

abstract public class Pin2Comp
    extends ImgComponents {

  Node[] nodes;
  //GridCanvas grid;

  public Pin2Comp() {
    nodes = new Node[2];
```

```
nodes[0] = new Node();
nodes[1] = new Node();
txtValue = new JTextField("");
txtValue.setBorder(null);
txtName = new JTextField("");
txtName.setBorder(null);
txtName.setEditable(true);
txtValue.setEditable(true);

txtValue.setBorder(null);
txtValue.setBackground(Color.WHITE);
txtValue.setForeground(Color.BLUE);
txtName.setBorder(null);
txtName.setBackground(Color.WHITE);
txtName.setForeground(Color.BLUE);
txtName.setBounds(50, 50, 25, 15);
txtValue.setBounds(0, 0, 45, 15);
txtName.addMouseListener(new MouseAdapter() {
  public void mouseClicked(MouseEvent me) {
    txtName_mouse_clicked(me);
  }

});
txtName.addFocusListener(new FocusAdapter() {
  public void focusLost(FocusEvent fe) {
    txtName_focus_lost(fe);
  }

});
txtName.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    txtName_action_performed(ae);
  }
}
);
txtName.getDocument().addDocumentListener(new DocumentListener() {
  public void insertUpdate(DocumentEvent e) {

    setComponentName(txtName.getText());

  }

  public void removeUpdate(DocumentEvent e) {

    setComponentName(txtName.getText());
```

```
      }

      public void changedUpdate(DocumentEvent e) {
        ;
      }

    }
    );

    txtValue.addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent me) {
        //txtValue_mouse_clicked(me);
      }

    });
    txtValue.addFocusListener(new FocusAdapter() {
      public void focusLost(FocusEvent fe) {
        txtValue_focus_lost(fe);
      }

    });
    txtValue.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        txtValue_action_performed(ae);
      }
    }
    );

    txtValue.getDocument().addDocumentListener(new DocumentListener() {
      public void insertUpdate(DocumentEvent e) {

        String strValue = txtValue.getText();
        int index = strValue.indexOf(" ");

//setValue(" ");

        Double val = new Double(strValue.substring(0, strValue.indexOf(" ")));
        setValue(val.doubleValue());

      }

      public void removeUpdate(DocumentEvent e) {
//System.out.println("REMOVE UPDATE");
        //      String strValue=txtValue.getText();
        //    int index=strValue.indexOf(" ");
```

```
//setValue(" ");
//System.out.println();
       //  Double val=new Double(strValue.substring(0,strValue.indexOf(" ")));
       //setValue(val.doubleValue());

     }

     public void changedUpdate(DocumentEvent e) {
       ;
     }
   }
   );

 }

 public final Node getNode(int index) {
   if (index < 2) {
     return nodes[index];
   }
   else {
     return null;
   }
 }

 public int getNumNodes() {
   return 2;
 }

 public final void setNodes() {
   int[] valuePos = new int[2];
   int[] namePos = new int[2];

   if (getAlignment() == 1 || getAlignment() == 3) {
     int[] xyPos = getPosition();
     int x = xyPos[0];
     int y = xyPos[1];
     int[] node1 = {
         x, y - 25};
     getNode(0).setPosition(node1);
     getNode(0).setEquivPos(node1);
     int[] node2 = {
         x, y + 25};
     getNode(1).setPosition(node2);
     getNode(1).setEquivPos(node2);
```

```
      namePos[0] = x + 15;
      namePos[1] = y - 5;
      valuePos[0] = x + 15;
      valuePos[1] = y + 20;
      setNamePos(namePos);
      setValuePos(valuePos);
    }
    else {
      int[] xyPos = getPosition();
      int x = xyPos[0];
      int y = xyPos[1];
      int[] node1 = {
         x - 25, y};
      getNode(0).setPosition(node1);
      getNode(0).setEquivPos(node1);
      int[] node2 = {
         x + 25, y};
      getNode(1).setPosition(node2);
      getNode(1).setEquivPos(node2);

      namePos[0] = x - 25;
      namePos[1] = y - 14;
      valuePos[0] = x - 10;
      valuePos[1] = y + 22;
      setNamePos(namePos);
      setValuePos(valuePos);

      int[] a = getNode(0).getPosition();
      int[] b = getNode(1).getPosition();

    }

    int[] coordinates = getPosition();
    int[] node1 = getNode(0).getPosition();
    int[] node2 = getNode(1).getPosition();

  }

////////////////////////////////////////////////////////////////////////////
  public void draw(GridCanvas gridCanvas, Component comp) {

    // grid = gridCanvas;
    int[] coordinates = getPosition();
    int[] node1 = getNode(0).getPosition();
```

```
int[] node2 = getNode(1).getPosition();

int[] node1Pos, node2Pos;
int[] valuePos = getValuePos();
int[] namePos = getNamePos();

node1Pos = getNode(0).getPosition();
node2Pos = getNode(1).getPosition();
int count = getIndex();
double value = getValue();

int xPos = coordinates[0];
int yPos = coordinates[1];
int node1XPos = node1Pos[0];
int node1YPos = node1Pos[1];
int node2XPos = node2Pos[0];
int node2YPos = node2Pos[1];
int nameXPos = namePos[0];
int nameYPos = namePos[1];
int valueXPos = valuePos[0];
int valueYPos = valuePos[1];
node1XPos -= 25;
node1YPos -= 25;
node2XPos -= 25;
node2YPos -= 25;

//     xPos -= 25;
//   yPos -= 25;

//}

String unit = getUnit();
if (unit.equals(null) || unit.equals("Null")) {
  unit = "";

}

if (getNodeVisible()) {
  if (getActiveState()) {

    gridCanvas.gridImage.setColor(new Color(53, 165, 33));
    //gridCanvas.gridImage.drawRect(coordinates[0]-25,coordinates[1]-25,50,50);
    if (getAlignment() == 0 || getAlignment() == 2) {
      gridCanvas.gridImage.drawRect(node1XPos + 23, node1YPos + 23, 5, 5);
      gridCanvas.gridImage.drawRect(node2XPos + 23, node2YPos + 23, 5, 5);
```

```
      }
      else {
        gridCanvas.gridImage.drawRect(xPos + 23, yPos - 3, 5, 5);
        gridCanvas.gridImage.drawRect(xPos + 23, yPos + 50, 5, 5);
      }

      setNameState(true);
      setValueState(true);
    }

    if (getNode(0).getActiveState()) {
      gridCanvas.gridImage.drawRect(node1XPos + 23, node1YPos + 23, 5, 5);
    }
    if (getNode(1).getActiveState()) {
      gridCanvas.gridImage.drawRect(node2XPos + 23, node2YPos + 23, 5, 5);
    }
  }
  gridCanvas.gridImage.drawImage(rotateImage(getImageByState(), getAlignment(),
                                 comp), xPos - 25, yPos - 25,
                         comp);

  //gridCanvas.gridImage.drawImage(getImageByState(),coordinates[0]-25,coordinates[1]-
25,comp);
  if (getNodeVisible()) {
    gridCanvas.gridImage.drawImage(getNode(0).getImageByState(),
                         node1XPos + 25,
                         node1YPos + 25, comp);
    gridCanvas.gridImage.drawImage(getNode(1).getImageByState(),
                         node2XPos + 25,
                         node2YPos + 25, comp);
  }
  //valueUnitPanel.setLocation(valuePos[0],valuePos[1]);
  // txtName.setLocation(namePos[0],namePos[1]);
  // txtValue.setLocation(valuePos[0],valuePos[1]);
  // txtValue.setText(value+" "+unit);
  // txtName.setText(getComponentName());

  //if(ActiveState.nameEditable)
  //gridCanvas.add(txtName);
  //else
  ///{
  //gridCanvas.remove(txtName);
  if (getNameVisible()) {
    if (getNameState()) {
      if (getAlignment() == 1 || getAlignment() == 3) {
```

```
      gridCanvas.gridImage.setColor(Color.PINK);
      gridCanvas.gridImage.drawRect(nameXPos - 5, nameYPos - 10, 40, 10);
      gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
      gridCanvas.gridImage.drawString(getComponentName(), nameXPos,
                                      nameYPos);
    }
    else {
      gridCanvas.gridImage.setColor(Color.PINK);
      gridCanvas.gridImage.drawRect(nameXPos - 5, nameYPos - 10, 40, 10);
      gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
      gridCanvas.gridImage.drawString(getComponentName(), nameXPos,
                                      nameYPos);

    }
  }
  else {
    if (getAlignment() == 1 || getAlignment() == 3) {
      gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
      gridCanvas.gridImage.drawString(getComponentName(), nameXPos,
                                      nameYPos);
    }
    else {
      gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
      gridCanvas.gridImage.drawString(getComponentName(), nameXPos,
                                      nameYPos);
    }

  }
}
if (getComponentType() != ActiveState.CURRENTCOMP) {
  if (getValueVisible()) {
    if (getValueState()) {
      if (getAlignment() == 1 || getAlignment() == 3) {
        gridCanvas.gridImage.setColor(Color.PINK);
        gridCanvas.gridImage.drawRect(valueXPos - 5, valueYPos - 10, 40, 10);
        gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
        gridCanvas.gridImage.drawString(value + unit, valueXPos, valueYPos);

      }
      else {
        gridCanvas.gridImage.setColor(Color.PINK);
        gridCanvas.gridImage.drawRect(valueXPos - 5, valueYPos - 10, 40, 10);
        gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
        gridCanvas.gridImage.drawString(value + unit, valueXPos, valueYPos);
```

```
        }

       }
       else {
        if (getAlignment() == 1 || getAlignment() == 3) {
          gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
          gridCanvas.gridImage.drawString(value + unit, valueXPos, valueYPos);
        }
        else {
          gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
          gridCanvas.gridImage.drawString(value + unit, valueXPos, valueYPos);
        }
       }
      }
    }

    int[] coordinates1 = getPosition();
    int[] node11 = getNode(0).getPosition();
    int[] node12 = getNode(1).getPosition();

  }

////////////////////////////////////////////////////////////////////////////
   public boolean occupiedArea(int[] pos) {
    int[] node1, node2;
    node1 = getNode(0).getPosition();
    node2 = getNode(1).getPosition();
    /*if(ActiveState.currentXScale==1.1)
      {
      node1[0]-=25;node1[1]-=25;
      node2[0]-=25;node2[1]-=25;
      }*/

    if ( ( ( (pos[0] >= node1[0]) & (pos[0] <= node2[0])) &
          ( (pos[1] > node1[1] - 10) & (pos[1] < node1[1] + 10)))
        || ( ( (pos[1] >= node1[1]) & (pos[1] <= node2[1])) &
            ( (pos[0] > node1[0] - 10) & (pos[0] < node1[0] + 10))))

    {
      //node1[0]+=25;node1[1]+=25;
      //node2[0]+=25;node2[1]+=25;
      setActiveState(true);
      return true;
    }
    else {
```

```
    //node1[0]+=25;node1[1]+=25;
    //node2[0]+=25;node2[1]+=25;
    setActiveState(false);
    return false;
   }

  }

////////////////////////////////////////////////////////////////////////////
  public boolean mouseOverArea(int[] pos) {
   int[] node1, node2;
   node1 = getNode(0).getPosition();
   node2 = getNode(1).getPosition();
   /*if(ActiveState.currentXScale==1.1)
    {
    node1[0]-=25;node1[1]-=25;
    node2[0]-=25;node2[1]-=25;
    }*/

   if ( ( ( (pos[0] >= node1[0] - 5) & (pos[0] <= node2[0] + 5)) &
        ( (pos[1] > node1[1] - 10) & (pos[1] < node1[1] + 10)))
      || ( ( (pos[1] >= node1[1] - 5) & (pos[1] <= node2[1] + 5)) &
         ( (pos[0] > node1[0] - 10) & (pos[0] < node1[0] + 10))))

   {

    //node1[0]+=25;node1[1]+=25;
    //node2[0]+=25;node2[1]+=25;
    return true;

   }
   else {
    //node1[0]+=25;node1[1]+=25;
    //node2[0]+=25;node2[1]+=25;
    return false;
   }

  }

////////////////////////////////////////////////////////////////////////////

  public boolean occupiedValueArea(int[] pos) {

   int[] valuePos = getValuePos();
```

```
    if ( ( ( (pos[0] >= valuePos[0] - 10) & (pos[0] <= valuePos[0] + 40)) &
         ( (pos[1] >= valuePos[1] - 10) & (pos[1] <= valuePos[1])))) {
      setValueState(true);

      return true;
    }
    else {
      setValueState(false);
      return false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  public boolean mouseOverValueArea(int[] pos) {

    int[] valuePos = getValuePos();

    if ( ( ( (pos[0] >= valuePos[0] - 10) & (pos[0] <= valuePos[0] + 40)) &
         ( (pos[1] >= valuePos[1] - 10) & (pos[1] <= valuePos[1])))) {
      return true;
    }
    else {
      return false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  public boolean occupiedNameArea(int[] pos) {

    int[] namePos = getNamePos();

    if ( ( ( (pos[0] >= namePos[0] - 10) & (pos[0] <= namePos[0] + 40)) &
         ( (pos[1] >= namePos[1] - 10) & (pos[1] <= namePos[1] - 5)))) {
      setNameState(true);
      return true;
    }
    else {
      setNameState(false);
      return false;
    }

  }
```

```
////////////////////////////////////////////////////////////////////////////
  public boolean mouseOverNameArea(int[] pos) {

    int[] namePos = getNamePos();

    if ( ( ( (pos[0] >= namePos[0] - 10) & (pos[0] <= namePos[0] + 40)) &
         ( (pos[1] >= namePos[1] - 10) & (pos[1] <= namePos[1] - 5)))) {
      return true;
    }
    else {
      return false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void txtName_mouse_clicked(MouseEvent me) {

    int[] namePos = getNamePos();

//grid.gridImage.setColor(new Color(235,240,248));
//grid.gridImage.clearRect(0, 0, 1200, 1200);
//grid.gridImage.drawRect(namePos[0]-1,namePos[1]-1,26,16);
    txtName.setBorder(BorderFactory.createLineBorder(new Color(235, 250, 255)));
//grid.gridImage.setColor(new Color(235,240,248));
//grid.gridImage.fillRect(namePos[0]-10,namePos[1],10,15);
    // grid.gridImage.setColor(Color.BLUE);
//grid.redraw();

    //grid.drawGridLayout(ActiveState.startPoint);
//drawAllComps();
    //grid.redraw();

//txtName.setEditable(true);
    txtName.setCursor(new Cursor(Cursor.TEXT_CURSOR));

  }

  private void txtName_focus_lost(FocusEvent me) {

    txtName.setBorder(null);
    txtName.setEditable(false);
  }

  private void txtName_action_performed(ActionEvent me) {
```

```
    String name = txtName.getText();
    setComponentName(name);

  }

  /*  private void txtValue_mouse_clicked(MouseEvent me) {
      int[] valuePos = getValuePos();
      //txtValue.setBorder(BorderFactory.createLineBorder(Color.BLUE));
       txtValue.setBorder(BorderFactory.createLineBorder(new Color(235, 250, 255)));
      grid.gridImage.setColor(new Color(235, 240, 248));
      grid.gridImage.fillRect(valuePos[0] - 10, valuePos[1], 10, 15);
      grid.gridImage.setColor(Color.BLUE);
      grid.redraw();
      if (me.getClickCount() >= 2) {
        txtValue.setEditable(true);
        txtValue.setCursor(new Cursor(Cursor.TEXT_CURSOR));
      }
    }*/

  private void txtValue_focus_lost(FocusEvent me) {

    txtValue.setBorder(null);
    txtValue.setEditable(false);

  }

  private void txtValue_action_performed(ActionEvent me) {

    String name = txtValue.getText();

    setValue(new Double(name).doubleValue());

  }

}
```

Pin4Comp.java

```
 package ECS.CircuitComponents;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

import ECS.*;
import ECS.GUPanels.*;

abstract public class Pin4Comp
    extends ImgComponents {

  Node[] nodes;
  //GridCanvas grid;

  public Pin4Comp() {
    nodes = new Node[4];
    nodes[0] = new Node();
    nodes[1] = new Node();
    nodes[2] = new Node();
    nodes[3] = new Node();
    txtValue = new JTextField("");
    txtValue.setBorder(null);
    txtName = new JTextField("");
    txtName.setBorder(null);
    txtName.setEditable(true);
    txtValue.setEditable(true);

    txtValue.setBorder(null);
    txtValue.setBackground(Color.WHITE);
    txtValue.setForeground(Color.BLUE);
    txtName.setBorder(null);
    txtName.setBackground(Color.WHITE);
    txtName.setForeground(Color.BLUE);
    txtName.setBounds(50, 50, 25, 15);
    txtValue.setBounds(0, 0, 45, 15);
    txtName.addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent me) {
        txtName_mouse_clicked(me);
      }
```

```
});
txtName.addFocusListener(new FocusAdapter() {
  public void focusLost(FocusEvent fe) {
    txtName_focus_lost(fe);
  }

});
txtName.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    txtName_action_performed(ae);
  }
}
);
txtName.getDocument().addDocumentListener(new DocumentListener() {
  public void insertUpdate(DocumentEvent e) {
    ;
    setComponentName(txtName.getText());

  }

  public void removeUpdate(DocumentEvent e) {

    setComponentName(txtName.getText());
  }

  public void changedUpdate(DocumentEvent e) {
    ;
  }

}
);

txtValue.addMouseListener(new MouseAdapter() {
  public void mouseClicked(MouseEvent me) {
    txtValue_mouse_clicked(me);
  }

});
txtValue.addFocusListener(new FocusAdapter() {
  public void focusLost(FocusEvent fe) {
    txtValue_focus_lost(fe);
  }

});
```

```
    txtValue.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        txtValue_action_performed(ae);
      }
    }
    );

    txtValue.getDocument().addDocumentListener(new DocumentListener() {
      public void insertUpdate(DocumentEvent e) {

        String strValue = txtValue.getText();
        int index = strValue.indexOf(" ");

//setValue(" ");
//System.out.println();
        Double val = new Double(strValue.substring(0, strValue.indexOf(" ")));
        setValue(val.doubleValue());
//System.out.println(txtName.getText());
      }

      public void removeUpdate(DocumentEvent e) {
//System.out.println("REMOVE UPDATE");
        //      String strValue=txtValue.getText();
        //    int index=strValue.indexOf(" ");

//setValue(" ");
//System.out.println();
        //  Double val=new Double(strValue.substring(0,strValue.indexOf(" ")));
        //setValue(val.doubleValue());

      }

      public void changedUpdate(DocumentEvent e) {
        ;
      }
    }
    );

  }

  public final Node getNode(int index) {
    if (index < 4) {
      return nodes[index];
    }
    else {
```

```
    return null;
  }
}

public int getNumNodes() {
  return 4;
}

public final void setNodes() {
  int[] valuePos = new int[2];
  int[] namePos = new int[2];

  int[] xyPos = getPosition();
  xyPos[0] -= 25;
  xyPos[1] -= 25;
  getNode(0).setPosition(xyPos);
  getNode(0).setEquivPos(xyPos);
  xyPos[0] += 50;
  getNode(1).setPosition(xyPos);
  getNode(1).setEquivPos(xyPos);
  xyPos[1] += 50;
  getNode(2).setPosition(xyPos);
  getNode(2).setEquivPos(xyPos);
  xyPos[0] -= 50;
  getNode(3).setPosition(xyPos);
  getNode(3).setEquivPos(xyPos);
  xyPos[0] += 25;
  xyPos[1] -= 25;

  namePos[0] = xyPos[0];
  namePos[1] = xyPos[1] - 30;
  valuePos[0] = xyPos[0] - 40;
  valuePos[1] = xyPos[1] + 40;
  setNamePos(namePos);
  setValuePos(valuePos);

}

public void draw(GridCanvas gridCanvas, Component comp) {
  //grid = gridCanvas;
  int[] coordinates = getPosition();
  int[] node1Pos, node2Pos, node3Pos, node4Pos;
  node1Pos = getNode(0).getPosition();
  node2Pos = getNode(1).getPosition();
  node3Pos = getNode(2).getPosition();
```

```
node4Pos = getNode(3).getPosition();
int[] valuePos = getValuePos();
int[] namePos = getNamePos();
String unit = getUnit();
int count = getIndex();
double value = getValue();

if (getNodeVisible()) {
  if (getActiveState()) {
    gridCanvas.gridImage.setColor(new Color(53, 165, 33));
    gridCanvas.gridImage.drawRect(coordinates[0] - 27, coordinates[1] - 28,
                                  5,
                                  5);
    gridCanvas.gridImage.drawRect(coordinates[0] + 23, coordinates[1] - 28,
                                  5,
                                  5);
    gridCanvas.gridImage.drawRect(coordinates[0] - 27, coordinates[1] + 22,
                                  5,
                                  5);
    gridCanvas.gridImage.drawRect(coordinates[0] + 23, coordinates[1] + 22,
                                  5,
                                  5);

    setNameState(true);
    setValueState(true);

  }
  if (getNode(0).getActiveState()) {
    gridCanvas.gridImage.drawRect(coordinates[0] - 27, coordinates[1] - 28,
                                  5,
                                  5);
  }
  if (getNode(1).getActiveState()) {
    gridCanvas.gridImage.drawRect(coordinates[0] + 23, coordinates[1] - 28,
                                  5,
                                  5);
  }
  if (getNode(2).getActiveState()) {
    gridCanvas.gridImage.drawRect(coordinates[0] + 23, coordinates[1] + 22,
                                  5,
                                  5);
  }
  if (getNode(3).getActiveState()) {
    gridCanvas.gridImage.drawRect(coordinates[0] + 23, coordinates[1] + 22,
                                  5,
```

```
                        5);
    }
   }
   gridCanvas.gridImage.drawImage(rotateImage(getImageByState(), getAlignment(),
                                comp), coordinates[0] - 25,
                        coordinates[1] - 25, comp);

   //gridCanvas.gridImage.drawImage(getImageByState(),coordinates[0]-25,coordinates[1]-
25,comp);
   if (getNodeVisible()) {
    gridCanvas.gridImage.drawImage(getNode(0).getImageByState(), node1Pos[0],
                        node1Pos[1], comp);
    gridCanvas.gridImage.drawImage(getNode(1).getImageByState(), node2Pos[0],
                        node2Pos[1], comp);
    gridCanvas.gridImage.drawImage(getNode(2).getImageByState(), node3Pos[0],
                        node3Pos[1], comp);
    gridCanvas.gridImage.drawImage(getNode(3).getImageByState(), node4Pos[0],
                        node4Pos[1], comp);
   }
   if (getNameState()) {

    gridCanvas.gridImage.setColor(Color.PINK);
    gridCanvas.gridImage.drawRect(namePos[0] - 15, namePos[1] - 10, 40, 10);
    gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
    gridCanvas.gridImage.drawString(getComponentName(), namePos[0] - 10,
                        namePos[1]);

   }
   else {

    gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
    gridCanvas.gridImage.drawString(getComponentName(), namePos[0] - 10,
                        namePos[1]);

   }

   if (getValueState()) {
    gridCanvas.gridImage.setColor(Color.PINK);
    gridCanvas.gridImage.drawRect(valuePos[0] + 20, valuePos[1] - 10, 50, 10);
    gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
    gridCanvas.gridImage.drawString(getN1() + " : " + getN2(),
                        valuePos[0] + 25, valuePos[1]);
   }
   else {
    gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
```

```
      gridCanvas.gridImage.drawString(getN1() + " : " + getN2(),
                                valuePos[0] + 25, valuePos[1]);
    }

  }

  public boolean occupiedArea(int[] pos) {
    int[] node1Pos, node2Pos, node3Pos, node4Pos;
    node1Pos = getNode(0).getPosition();
    node2Pos = getNode(1).getPosition();
    node3Pos = getNode(2).getPosition();
    node4Pos = getNode(3).getPosition();
    if ( (pos[0] > node1Pos[0]) & (pos[0] < node2Pos[0]) & (pos[1] > node1Pos[1]) &
        (pos[1] < node3Pos[1])) {
      setActiveState(true);
      return true;
    }
    else {
      setActiveState(false);
      return false;
    }
  }

  public boolean mouseOverArea(int[] pos) {
    int[] node1Pos, node2Pos, node3Pos, node4Pos;
    node1Pos = getNode(0).getPosition();
    node2Pos = getNode(1).getPosition();
    node3Pos = getNode(2).getPosition();
    node4Pos = getNode(3).getPosition();
    if ( (pos[0] > node1Pos[0] - 5) & (pos[0] < node2Pos[0] + 5) &
        (pos[1] > node1Pos[1] - 5) & (pos[1] < node3Pos[1] + 5)) {

      return true;
    }
    else {

      return false;
    }
  }

////////////////////////////////////////////////////////////////////////////

  public boolean occupiedValueArea(int[] pos) {

    int[] valuePos = getValuePos();
```

```
    if ( ( ( (pos[0] >= valuePos[0] - 20) & (pos[0] <= valuePos[0] + 70)) &
         ( (pos[1] >= valuePos[1] - 10) & (pos[1] <= valuePos[1] + 20)))) {
      setValueState(true);

      return true;
    }
    else {
      setValueState(false);
      return false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  public boolean mouseOverValueArea(int[] pos) {

    int[] valuePos = getValuePos();

    if ( ( ( (pos[0] >= valuePos[0] + 20) & (pos[0] <= valuePos[0] + 70)) &
         ( (pos[1] >= valuePos[1] - 10) & (pos[1] <= valuePos[1] + 20)))) {
      return true;
    }
    else {
      return false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  public boolean occupiedNameArea(int[] pos) {

    int[] namePos = getNamePos();

    if ( ( ( (pos[0] >= namePos[0] - 10) & (pos[0] <= namePos[0] + 30)) &
         ( (pos[1] >= namePos[1] - 10) & (pos[1] <= namePos[1] - 5)))) {
      setNameState(true);
      return true;
    }
    else {
      setNameState(false);
      return false;
    }

  }
```

```
////////////////////////////////////////////////////////////////////////////
  public boolean mouseOverNameArea(int[] pos) {

    int[] namePos = getNamePos();

    if ( ( ( (pos[0] >= namePos[0] - 10) & (pos[0] <= namePos[0] + 30)) &
         ( (pos[1] >= namePos[1] - 10) & (pos[1] <= namePos[1] - 5)))) {
      return true;
    }
    else {
      return false;
    }

  }

////////////////////////////////////////////////////////////////////////////
  private void txtName_mouse_clicked(MouseEvent me) {

    int[] namePos = getNamePos();

//grid.gridImage.setColor(new Color(235,240,248));
//grid.gridImage.clearRect(0, 0, 1200, 1200);
//grid.gridImage.drawRect(namePos[0]-1,namePos[1]-1,26,16);
    txtName.setBorder(BorderFactory.createLineBorder(new Color(235, 250, 255)));
//grid.gridImage.setColor(new Color(235,240,248));
//grid.gridImage.fillRect(namePos[0]-10,namePos[1],10,15);
    //  grid.gridImage.setColor(Color.BLUE);
//grid.redraw();

    //grid.drawGridLayout(ActiveState.startPoint);
//drawAllComps();
    //grid.redraw();

//txtName.setEditable(true);
    txtName.setCursor(new Cursor(Cursor.TEXT_CURSOR));

  }

  private void txtName_focus_lost(FocusEvent me) {

    txtName.setBorder(null);
    txtName.setEditable(false);
  }
```

```
  private void txtName_action_performed(ActionEvent me) {

    String name = txtName.getText();
    setComponentName(name);

  }

  private void txtValue_mouse_clicked(MouseEvent me) {
    /*
        int[] valuePos = getValuePos();
        //txtValue.setBorder(BorderFactory.createLineBorder(Color.BLUE));
         txtValue.setBorder(BorderFactory.createLineBorder(new Color(235, 250, 255)));
        grid.gridImage.setColor(new Color(235, 240, 248));
        grid.gridImage.fillRect(valuePos[0] - 10, valuePos[1], 10, 15);
        grid.gridImage.setColor(Color.BLUE);
        grid.redraw();
        if (me.getClickCount() >= 2) {
          txtValue.setEditable(true);
          txtValue.setCursor(new Cursor(Cursor.TEXT_CURSOR));
        }*/
  }

  private void txtValue_focus_lost(FocusEvent me) {

    txtValue.setBorder(null);
    txtValue.setEditable(false);

  }

  private void txtValue_action_performed(ActionEvent me) {

    String name = txtValue.getText();

    setValue(new Double(name).doubleValue());

  }

}
```

ConnectionNode.java

```
 package ECS.CircuitComponents.Pin1Components;

import java.awt.*;

import ECS.*;
import ECS.CircuitComponents.*;
import ECS.GUPanels.*;

public class ConnectionNode
    extends ImgComponents {

  private static final String[] units = {
      ""};
  Node[] nodes;

  public ConnectionNode() {
    nodes = new Node[1];
    nodes[0] = new Node();
  }

  public String getGenericName() {
    return "Connection Point";
  }

  public String getName() {
    return "";
  }

  public int getComponentType() {
    return ActiveState.CONNODE;
  }

  public String getImageName() {
    return "SConNode.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
```

```
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public boolean occupiedArea(int[] pos) {
  int[] xypos = getNode(0).getPosition();
  if ( (pos[0] > xypos[0] - 5) & (pos[0] < xypos[0] + 5)
      & (pos[1] > xypos[1] - 5) & (pos[1] < xypos[1] + 5)) {
    setActiveState(true);
    return true;
  }
  else {
    setActiveState(false);
    return false;
  }
}

public boolean mouseOverArea(int[] pos) {
  int[] xypos = getNode(0).getPosition();
  if ( (pos[0] > xypos[0] - 5) & (pos[0] < xypos[0] + 5)
      & (pos[1] > xypos[1] - 5) & (pos[1] < xypos[1] + 5)) {
    return true;
  }
  else {

    return false;
  }
}

public void draw(GridCanvas gridCanvas, Component com) {
  int[] pos = getPosition();
  if (getNodeVisible()) {
    gridCanvas.gridImage.drawImage(getImageByState(), pos[0], pos[1], com);
  }
}

public String getNetListString() {
  return "";
}

public String getCIRMLString() {
  return "";
```

```
  }

  public int getNumNodes() {
    return 1;
  }

  public void setNodes() {
    int[] xy = getPosition();
    getNode(0).setPosition(xy);

  }

  public Node getNode(int index) {
    if (index < 1) {
      return nodes[0];
    }
    else {
      return null;
    }
  }

  public boolean occupiedValueArea(int[] pos) {

    return false;
  }

  public boolean occupiedNameArea(int[] pos) {

    return false;
  }

  public boolean mouseOverNameArea(int[] pos) {

    return false;
  }

  public boolean mouseOverValueArea(int[] pos) {

    return false;
  }

}
```

CurrentMarker.java

```
package ECS.CircuitComponents.Pin1Components;

import java.awt.*;

import ECS.*;
import ECS.CircuitComponents.*;
import ECS.GUPanels.*;

public class CurrentMarker
   extends Pin1Comp {
  private static final String[] units = {
     ""};
  private boolean controller = false;
  public CurrentMarker() {
   setComponentName(getName());
  }

  public String getGenericName() {
   return "Current Marker";
  }

  public String getName() {
   return "I";
  }

  public int getComponentType() {
   return ActiveState.CURRENTMARK;
  }

  public String getImageName() {
   return "currentM.jpg";
  }

  public String[] getAllUnits() {
   return units;
  }

  public String[] getAllInputUnits() {
   return units;
  }

  public String getDefaultUnit() {
```

```
  return units[0];
}

public void setNodes() {
  int[] xy = getPosition();
  //xy[0]-=25;
  getNode(0).setPosition(xy);
  getNode(0).setEquivPos(xy);
}

public void draw(GridCanvas gridCanvas, Component comp) {
  if (isVisible()) {
    int[] coordinates = getPosition();
    gridCanvas.gridImage.drawImage(rotateImage(getImageByState(),
                                   getAlignment(),
                                   comp)
                             , coordinates[0], coordinates[1] - 25,
                             comp);
    if (getActiveState()) {
      gridCanvas.gridImage.setColor(Color.red);
      gridCanvas.gridImage.drawString(getComponentName(),
                                 coordinates[0] + 15,
                                 coordinates[1] - 15);
    }
    else {
      gridCanvas.gridImage.setColor(Color.BLUE);
      gridCanvas.gridImage.drawString(getComponentName(),
                                 coordinates[0] + 15,
                                 coordinates[1] - 15);
    }
  }
}

public String getNetListString() {

  String strNetlist = "";
  strNetlist += getName() + "_" + getComponentName();
  strNetlist += "  " + getNode(0).strName;

  return strNetlist;
  //return "";

}

public String getCIRMLString() {
```

```
    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName;

    return strNetlist;
    //return "";

  }

  public boolean occupiedValueArea(int[] pos) {

    return false;
  }

  public boolean occupiedNameArea(int[] pos) {

    return false;
  }

  public void setAsController() {
    controller = true;
  }

  public boolean isController() {
    return controller;
  }

}
```

Ground.java

```
 package ECS.CircuitComponents.Pin1Components;

import java.awt.*;

import ECS.*;
import ECS.CircuitComponents.*;
import ECS.GUPanels.*;

public class Ground
    extends ImgComponents {
```

```
private static final String[] units = {
    ""};
Node[] nodes;
int alignment = 0;

public Ground() {
  nodes = new Node[1];
  nodes[0] = new Node();
}

public String getGenericName() {
  return "Ground";
}

public String getName() {
  return "Ground";
}

public int getComponentType() {
  return ActiveState.GROUND;
}

public String getImageName() {
  return "SGround.gif";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public Node getNode(int index) {
  switch (index) {
    case 0:
      return nodes[0];
    default:
      return null;
```

```
  }
}

public int getNumNodes() {
  return 1;
}

public void setNodes() {
  int[] xy = new int[2];
  xy = getPosition();
  getNode(0).setPosition(xy);
  getNode(0).setEquivPos(xy);

}

public void draw(GridCanvas gridCanvas, Component comp) {
  int[] coordinates = getPosition();
  gridCanvas.gridImage.drawImage(getImageByState(), coordinates[0] - 25,
                                 coordinates[1] - 25, comp);
  //coordinates[0],coordinates[1]-25,comp);

  int[] rxy1 = nodes[0].getPosition();
  if (getNodeVisible()) {
    gridCanvas.gridImage.drawImage(getNode(0).getImageByState(), rxy1[0] - 1,
                                   rxy1[1], comp);
  }
}

public boolean occupiedArea(int[] pos) {
  int[] xypos = getNode(0).getPosition();
  if ( (pos[0] > xypos[0] - 5) & (pos[0] < xypos[0] + 5)
      & (pos[1] > xypos[1] - 5) & (pos[1] < xypos[1] + 25)) {
    setActiveState(true);
    return true;
  }
  else {
    setActiveState(false);
    return false;
  }

}

public boolean mouseOverArea(int[] pos) {
  int[] xypos = getNode(0).getPosition();
  if ( (pos[0] > xypos[0] - 5) & (pos[0] < xypos[0] + 5)
```

```
      & (pos[1] > xypos[1] - 5) & (pos[1] < xypos[1] + 25)) {
    return true;
   }
   else {
    return false;
   }

  }

  public String getNetListString() {
   return "";
   // String netList="";
   //netList+=getName()+"_"+getComponentName();
   //netList+="  "+getNode(0).strName;
   //return netList;
  }

  public String getCIRMLString() {
   return "";
   // String netList="";
   //netList+=getName()+"_"+getComponentName();
   //netList+="  "+getNode(0).strName;
   //return netList;
  }

  public boolean occupiedValueArea(int[] pos) {

   return false;
  }

  public boolean occupiedNameArea(int[] pos) {

   return false;
  }

  public boolean mouseOverNameArea(int[] pos) {
   return false;
  }

  public boolean mouseOverValueArea(int[] pos) {
   return false;
  }

}
```

NodeL.java

```
 package ECS.CircuitComponents.Pin1Components;

import java.awt.*;

import ECS.*;
import ECS.CircuitComponents.*;
import ECS.GUPanels.*;

public class NodeL
    extends Pin1Comp {
  private static final String[] units = {
      "V"};
  private boolean controller = false;
  public NodeL() {
    setComponentName(getName());
  }

  public String getGenericName() {
    return "Voltage Differential Marker";
  }

  public String getName() {
    return "V";
  }

  public int getComponentType() {
    return ActiveState.NODEL;
  }

  public String getImageName() {
    return "SNodeL.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return units;
  }
```

```
public String getDefaultUnit() {
  return units[0];
}

public void setNodes() {
  int[] xy = getPosition();
  //xy[0]-=25;
  getNode(0).setPosition(xy);
  getNode(0).setEquivPos(xy);
}

public void draw(GridCanvas gridCanvas, Component comp) {
  if (isVisible()) {
    int[] coordinates = getPosition();
    gridCanvas.gridImage.drawImage(rotateImage(getImageByState(),
                                   getAlignment(),
                                   comp)
                             , coordinates[0], coordinates[1] - 25,
                             comp);
    if (getActiveState()) {
      gridCanvas.gridImage.setColor(Color.red);
      gridCanvas.gridImage.drawString(getComponentName() + "+",
                                 coordinates[0] + 15,
                                 coordinates[1] - 15);
    }
    else {
      gridCanvas.gridImage.setColor(Color.BLUE);
      gridCanvas.gridImage.drawString(getComponentName() + "+",
                                 coordinates[0] + 15,
                                 coordinates[1] - 15);
    }
  }
}

public String getNetListString() {

  String strNetlist = "";
  strNetlist += getName() + "_" + getComponentName();
  strNetlist += "  " + getNode(0).strName;

  return strNetlist;

}
```

```
  public String getCIRMLString() {

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName;

    return strNetlist;

  }

  public boolean occupiedValueArea(int[] pos) {

    return false;
  }

  public boolean occupiedNameArea(int[] pos) {

    return false;
  }

  public void setAsController() {
    controller = true;
  }

  public boolean isController() {
    return controller;
  }

}
```

NodeR.java

```
 package ECS.CircuitComponents.Pin1Components;

import java.awt.*;

import ECS.*;
import ECS.CircuitComponents.*;
import ECS.GUPanels.*;

public class NodeR
```

```
  extends Pin1Comp {
private static final String[] units = {
    "V"};

public NodeR() {
  setComponentName(getName());
}

public String getGenericName() {
  return "NodeR";
}

public String getName() {
  return "V";
}

public int getComponentType() {
  return ActiveState.NODER;
}

public String getImageName() {
  return "SNode+.gif";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public void draw(GridCanvas gridCanvas, Component comp) {
  if (isVisible()) {
    int[] coordinates = getPosition();
    gridCanvas.gridImage.drawImage(rotateImage(getImageByState(),
                                   getAlignment(),
                                   comp)
                             , coordinates[0], coordinates[1] - 25,
                             comp);
    if (getActiveState()) {
```

```
        gridCanvas.gridImage.setColor(Color.red);
        gridCanvas.gridImage.drawString(getComponentName() + "-",
                                        coordinates[0] + 15,
                                        coordinates[1] - 15);
      }
      else {
        gridCanvas.gridImage.setColor(Color.BLUE);
        gridCanvas.gridImage.drawString(getComponentName() + "-",
                                        coordinates[0] + 15,
                                        coordinates[1] - 15);

      }
    }
  }

  public String getNetListString() {

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName;

    return strNetlist;

  }

  public String getCIRMLString() {

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName;

    return strNetlist;

  }

  public void setNodes() {
    int[] xy = getPosition();

//xy[0]-=25;
    getNode(0).setPosition(xy);
    getNode(0).setEquivPos(xy);
  }

  public boolean occupiedValueArea(int[] pos) {
```

```
    return false;
  }

  public boolean occupiedNameArea(int[] pos) {

    return false;
  }

}
```

VoltageMarker.java

```
package ECS.CircuitComponents.Pin1Components;

import java.awt.*;

import ECS.*;
import ECS.CircuitComponents.*;
import ECS.GUPanels.*;

public class VoltageMarker
    extends Pin1Comp {
  private static final String[] units = {};
  private boolean controller = false;
  public VoltageMarker() {
    setComponentName(getName());
  }

  public String getGenericName() {
    return "Voltage Marker";
  }

  public String getName() {
    return "V";
  }

  public int getComponentType() {
    return ActiveState.VOLTAGEMARK;
  }

  public String getImageName() {
    return "voltageM.jpg";
```

```
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public void setNodes() {
  int[] xy = getPosition();
  //xy[0]-=25;
  getNode(0).setPosition(xy);
  getNode(0).setEquivPos(xy);
}

public void draw(GridCanvas gridCanvas, Component comp) {
  if (isVisible()) {
    int[] coordinates = getPosition();
    gridCanvas.gridImage.drawImage(rotateImage(getImageByState(),
                                   getAlignment(),
                                   comp)
                             , coordinates[0], coordinates[1] - 25,
                             comp);
    if (getActiveState()) {
      gridCanvas.gridImage.setColor(Color.red);
      gridCanvas.gridImage.drawString(getComponentName(),
                                 coordinates[0] + 15,
                                 coordinates[1] - 15);
    }
    else {
      gridCanvas.gridImage.setColor(Color.BLUE);
      gridCanvas.gridImage.drawString(getComponentName(),
                                 coordinates[0] + 15,
                                 coordinates[1] - 15);
    }
  }
}

public String getNetListString() {
```

```
    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName;

    return strNetlist;

  }

  public String getCIRMLString() {

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName;

    return strNetlist;

  }

  public boolean occupiedValueArea(int[] pos) {

    return false;
  }

  public boolean occupiedNameArea(int[] pos) {

    return false;
  }

  public void setAsController() {
    controller = true;
  }

  public boolean isController() {
    return controller;
  }

}
```

ConnectionLine.java

```
package ECS.CircuitComponents.Pin2Components;
```

```
import java.awt.*;

import ECS.*;
import ECS.CircuitComponents.*;
import ECS.GUPanels.*;

public class ConnectionLine
    extends Components {
  private static final String[] units = {
      ""};

  public Node[] nodes;

  public ConnectionLine() {
    nodes = new Node[2];
    nodes[0] = new Node();
    nodes[1] = new Node();
  }

////////////////////////////////////////////////////////////////////////////
  public String getGenericName() {
    return "Connector";
  }

  public String getName() {
    return null;
  }

  public int getComponentType() {
    return ActiveState.CONNECTOR;
  }

  public String getImageName() {
    return "SConnector.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return null;
  }
```

```
public String getDefaultUnit() {
  return units[0];
}

public Node getNode(int index) {
  if (index < 2) {
    return nodes[index];
  }
  else {
    return null;
  }
}

public int getNumNodes() {
  return 2;
}

public void setNodes() {
  int[] node1 = getNode(0).getPosition();
  int[] node2 = getNode(1).getPosition();
  int[] p1 = {
      node1[0], node1[1]};
  int[] p2 = {
      node2[0], node2[1]};
  nodes[0].setPosition(p1);
  nodes[0].setEquivPos(p1);
  nodes[1].setPosition(p2);
  nodes[1].setEquivPos(p2);
}

public boolean occupiedArea(int[] pos) {
  int a, b, c;

  // int[] points=getPoints();
  int[] p1 = nodes[0].getPosition();
  int[] p2 = nodes[1].getPosition();
  int[] xy = pos;
  a = - (p2[1] - p1[1]);
  b = (p2[0] - p1[0]);
  c = (p2[1] - p1[1]) * p1[0] - (p2[0] - p1[0]) * p1[1];
  double distance = Math.abs(a * pos[0] + b * pos[1] + c) /
      Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

  if (p1[0] == p2[0]) {
    if (distance < 3 &&
```

```
      ( ( (pos[1] >= p1[1] && pos[1] <= p2[1]) ||
        (pos[1] >= p2[1] && pos[1] <= p1[1])))) {

    setActiveState(true);
    return true;
   }
   else {
    setActiveState(false);
    return false;
   }

  }
  else {
   if (distance < 3 &&
      ( ( (pos[0] >= p1[0] && pos[0] <= p2[0]) ||
        (pos[0] >= p2[0] && pos[0] <= p1[0])))) {
    setActiveState(true);
    return true;
   }
   else {
    setActiveState(false);
    return false;
   }
  }

}

public boolean mouseOverArea(int[] pos) {
  int a, b, c;

  // int[] points=getPoints();
  int[] p1 = nodes[0].getPosition();
  int[] p2 = nodes[1].getPosition();
  int[] xy = pos;
  a = - (p2[1] - p1[1]);
  b = (p2[0] - p1[0]);
  c = (p2[1] - p1[1]) * p1[0] - (p2[0] - p1[0]) * p1[1];
  double distance = Math.abs(a * pos[0] + b * pos[1] + c) /
     Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));
  if (p1[0] == p2[0]) {
   if (distance < 3 &&
      ( ( (pos[1] >= p1[1] && pos[1] <= p2[1]) ||
        (pos[1] >= p2[1] && pos[1] <= p1[1])))) {

    return true;
```

```
    }
    else {

      return false;
    }

  }
  else {
    if (distance < 3 &&
        ( ( (pos[0] >= p1[0] && pos[0] <= p2[0]) ||
          (pos[0] >= p2[0] && pos[0] <= p1[0])))) {

      return true;
    }
    else {

      return false;
    }
  }
}

////////////////////////////////////////////////////////////////////////////
  public boolean isOnLine(int[] pos) {

    int a, b, c;

    // int[] points=getPoints();
    int[] p1 = nodes[0].getPosition();
    int[] p2 = nodes[1].getPosition();
    int[] xy = pos;
    a = - (p2[1] - p1[1]);
    b = (p2[0] - p1[0]);
    c = (p2[1] - p1[1]) * p1[0] - (p2[0] - p1[0]) * p1[1];
    double distance = Math.abs(a * pos[0] + b * pos[1] + c) /
        Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));
    if (p1[0] == p2[0]) {
      if (distance < 1 &&
          ( ( (pos[1] >= p1[1] && pos[1] <= p2[1]) ||
            (pos[1] >= p2[1] && pos[1] <= p1[1])))) {

        return true;
      }
      else {

        return false;
```

```
    }

   }
   else {
    if (distance < 1 &&
        ( ( (pos[0] >= p1[0] && pos[0] <= p2[0]) ||
           (pos[0] >= p2[0] && pos[0] <= p1[0])))) {

     return true;
    }
    else {

     return false;
    }
   }

  }

////////////////////////////////////////////////////////////////////////////
  public void draw(GridCanvas gridCanvas, Component comp) {
   int[] p1 = nodes[0].getPosition();
   int[] p2 = nodes[1].getPosition();

   if (getActiveState()) {
    gridCanvas.gridImage.setColor(Color.RED);
    gridCanvas.gridImage.drawLine(p1[0], p1[1], p2[0], p2[1]);
    if (getNodeVisible()) {
     gridCanvas.gridImage.setColor(new Color(24, 130, 17));
     gridCanvas.gridImage.drawRect(p1[0] - 2, p1[1] - 2, 5, 5);
     gridCanvas.gridImage.drawRect(p2[0] - 2, p2[1] - 2, 5, 5);
    }
   }
   else {
    gridCanvas.gridImage.setColor(new Color(24, 130, 17)); //new Color(106,39,19));
    gridCanvas.gridImage.drawLine(p1[0], p1[1], p2[0], p2[1]);

   }
   if (getNodeVisible()) {
    gridCanvas.gridImage.setColor(Color.BLUE);
    gridCanvas.gridImage.drawImage(nodes[0].getImageByState(), p1[0], p1[1],
                                   comp);
    gridCanvas.gridImage.drawImage(nodes[1].getImageByState(), p2[0], p2[1],
                                   comp);
    gridCanvas.gridImage.setColor(Color.BLUE);
   }
```

```
  }

  public String getNetListString() {
    return "";
  }

  public String getCIRMLString() {
    return "";
  }

////////////////////////////////////////////////////////////////////////////
  public boolean occupiedValueArea(int[] pos) {

    return false;
  }

  public boolean occupiedNameArea(int[] pos) {

    return false;
  }

  public boolean mouseOverNameArea(int[] pos) {

    return false;
  }

  public boolean mouseOverValueArea(int[] pos) {

    return false;
  }

}
```

CurrentComp.java

```
 package ECS.CircuitComponents.Pin2Components;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
import javax.swing.event.*;

import ECS.GUPanels.GridCanvas;

import ECS.*;
import ECS.CircuitComponents.*;

public class CurrentComp
    extends Pin2Comp {
  GridCanvas grid;
  private static final String[] units = {
      ""};
  private boolean controller = false;
  Current current;

  public CurrentComp() {
    current = new Current();
  }

  public int getComponentType() {
    return ActiveState.CURRENTCOMP;
  }

  public String getGenericName() {
    return "CurrentComp";
  }

  public String getName() {
    return "CC";
  }

  public String getImageName() {
    return "SCurrentComp.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return null;
  }

  public String getDefaultUnit() {
    return units[0];
```

```
}

public String getProbString() {
  return "";
}

public void setAsController() {
  controller = true;
}

public boolean isController() {
  return controller;
}

public void draw(GridCanvas gridCanvas, Component comp) {

  if (isVisible()) {
    grid = gridCanvas;
    int[] coordinates = getPosition();
    int[] node1Pos, node2Pos;
    int[] valuePos = getValuePos();
    int[] namePos = getNamePos();

    node1Pos = getNode(0).getPosition();
    node2Pos = getNode(1).getPosition();
    int count = getIndex();
    double value = getValue();

    int xPos = coordinates[0];
    int yPos = coordinates[1];
    int node1XPos = node1Pos[0];
    int node1YPos = node1Pos[1];
    int node2XPos = node2Pos[0];
    int node2YPos = node2Pos[1];
    int nameXPos = namePos[0];
    int nameYPos = namePos[1];
    int valueXPos = valuePos[0];
    int valueYPos = valuePos[1];
    node1XPos -= 25;
    node1YPos -= 25;
    node2XPos -= 25;
    node2YPos -= 25;
    if (ActiveState.currentXScale == 1.1) {
      xPos -= 50;
      yPos -= 50;
```

```
      node1XPos -= 25;
      node1YPos -= 25;
      node2XPos -= 25;
      node2YPos -= 25;
      nameXPos -= 25;
      nameYPos -= 25;
      valueXPos -= 25;
      valueYPos -= 25;

    }
    else if (ActiveState.currentXScale == 1.2) {
      xPos -= 1000;
      yPos -= 1000;
      node1XPos -= 25;
      node1YPos -= 25;
      node2XPos -= 25;
      node2YPos -= 25;
      nameXPos -= 25;
      nameYPos -= 25;
      valueXPos -= 25;
      valueYPos -= 25;

    }
    else if (ActiveState.currentXScale == 0.9) {
//  xPos+=50;yPos+=50;
      node1XPos += 25;
      node1YPos += 25;
      node2XPos += 25;
      node2YPos += 25;
      nameXPos += 25;
      nameYPos += 25;
      valueXPos += 25;
      valueYPos += 25;

    }
    else if (ActiveState.currentXScale == 0.8) {
      xPos += 100;
      yPos += 100;
    }
    else {
      xPos -= 25;
      yPos -= 25;

    }
```

```
    String unit = getUnit();
    if (unit.equals(null) || unit.equals("Null")) {
      unit = "";

    }

    if (getNodeVisible()) {
      if (getActiveState()) {

        gridCanvas.gridImage.setColor(new Color(53, 165, 33));
        //gridCanvas.gridImage.drawRect(coordinates[0]-25,coordinates[1]-25,50,50);
        if (getAlignment() == 0 || getAlignment() == 2) {
          gridCanvas.gridImage.drawRect(node1XPos + 23, node1YPos + 23, 5, 5);
          gridCanvas.gridImage.drawRect(node2XPos + 23, node2YPos + 23, 5, 5);
        }
        else {
          gridCanvas.gridImage.drawRect(xPos + 23, yPos - 3, 5, 5);
          gridCanvas.gridImage.drawRect(xPos + 23, yPos + 50, 5, 5);
        }

        setNameState(true);
        setValueState(true);
      }

      if (getNode(0).getActiveState()) {
        gridCanvas.gridImage.drawRect(node1XPos + 23, node1YPos + 23, 5, 5);
      }
      if (getNode(1).getActiveState()) {
        gridCanvas.gridImage.drawRect(node2XPos + 23, node2YPos + 23, 5, 5);
      }
    }
    gridCanvas.gridImage.drawImage(rotateImage(getImageByState(),
                                   getAlignment(),
                                   comp), xPos, yPos, comp);

    //gridCanvas.gridImage.drawImage(getImageByState(),coordinates[0]-25,coordinates[1]-
25,comp);
    if (getNodeVisible()) {
      gridCanvas.gridImage.drawImage(getNode(0).getImageByState(),
                             node1XPos + 25,
                             node1YPos + 25, comp);
      gridCanvas.gridImage.drawImage(getNode(1).getImageByState(),
                             node2XPos + 25,
                             node2YPos + 25, comp);
    }
```

```
//valueUnitPanel.setLocation(valuePos[0],valuePos[1]);
// txtName.setLocation(namePos[0],namePos[1]);
// txtValue.setLocation(valuePos[0],valuePos[1]);
// txtValue.setText(value+" "+unit);
// txtName.setText(getComponentName());

//if(ActiveState.nameEditable)
//gridCanvas.add(txtName);
//else
///{
//gridCanvas.remove(txtName);
if (getNameVisible()) {
  if (getNameState()) {
    if (getAlignment() == 1 || getAlignment() == 3) {
      gridCanvas.gridImage.setColor(Color.PINK);
      gridCanvas.gridImage.drawRect(nameXPos - 5, nameYPos - 10, 40, 10);
      gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
      gridCanvas.gridImage.drawString(getComponentName(), nameXPos,
                                      nameYPos);
    }
    else {
      gridCanvas.gridImage.setColor(Color.PINK);
      gridCanvas.gridImage.drawRect(nameXPos - 5, nameYPos - 10, 40, 10);
      gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
      gridCanvas.gridImage.drawString(getComponentName(), nameXPos,
                                      nameYPos);

    }
  }
  else {
    if (getAlignment() == 1 || getAlignment() == 3) {
      gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
      gridCanvas.gridImage.drawString(getComponentName(), nameXPos,
                                      nameYPos);
    }
    else {
      gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
      gridCanvas.gridImage.drawString(getComponentName(), nameXPos,
                                      nameYPos);
    }

  }
}
if (getComponentType() != ActiveState.CURRENTCOMP) {
  if (getValueVisible()) {
```

```
      if (getValueState()) {
        if (getAlignment() == 1 || getAlignment() == 3) {
          gridCanvas.gridImage.setColor(Color.PINK);
          gridCanvas.gridImage.drawRect(valueXPos - 5, valueYPos - 10, 40,
                                        10);
          gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
          gridCanvas.gridImage.drawString(value + unit, valueXPos,
                                          valueYPos);

        }
        else {
          gridCanvas.gridImage.setColor(Color.PINK);
          gridCanvas.gridImage.drawRect(valueXPos - 5, valueYPos - 10, 40,
                                        10);
          gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
          gridCanvas.gridImage.drawString(value + unit, valueXPos,
                                          valueYPos);

        }

      }
      else {
        if (getAlignment() == 1 || getAlignment() == 3) {
          gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
          gridCanvas.gridImage.drawString(value + unit, valueXPos,
                                          valueYPos);
        }
        else {
          gridCanvas.gridImage.setColor(mainApplet.canvasForegroundColor);
          gridCanvas.gridImage.drawString(value + unit, valueXPos,
                                          valueYPos);
        }
      }
    }
  }
}

public String getNetListString() {

  String strNetlist = "";
  strNetlist += getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
  return strNetlist;
```

```
  }

  public String getCIRMLString() {

    String strNetlist = "";
    strNetlist += getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    return strNetlist;

  }

}
```

DepComp.java

```
 package ECS.CircuitComponents.Pin2Components;

import ECS.CircuitComponents.*;

abstract public class DepComp
    extends Pin2Comp {
  public Components[] controllers;

  public void init(int numbControl) {
    controllers = new Components[numbControl];
  }
}
```

Resistor.java

```
package ECS.CircuitComponents.Pin2Components.PassiveComponents;

import java.lang.reflect.*;

import ECS.*;
import ECS.CircuitComponents.*;
```

```
public class Resistor
    extends Pin2Comp {

  private static final String[] units = {
      "\u00B5\u2126",
      "m\u2126", "\u2126", "K\u2126", "M\u2126"};
  private static final String[] inputUnits = {
      "\u00B5Ohm  (\u00B5\u2126)",
      "mOhm (m\u2126)", "Ohm (\u2126)", "KOhm (K\u2126)", "MOhm (M\u2126)"};
  private static final String[] netListUnits = {
      "UOhm",
      "MOhm", "Ohm", "KOhm", "MEGOhm"};
  public int getComponentType() {
    return ActiveState.RESISTOR;
  }

  public String getGenericName() {
    return "Resistor";
  }

  public String getName() {
    return "R";
  }

  public String getImageName() {
    return "SResistor.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return inputUnits;
  }

  public String getDefaultUnit() {
    return units[0];
  }

  public String getNetListString() {
    String unit = getUnit();
    for (int i = 0; i < Array.getLength(units); i++) {
      if (units[i].equals(unit)) {
```

```
      unit = netListUnits[i];
     }
    }

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += "  " + getValue() + unit;

    return strNetlist;

   }

   public String getCIRMLString() {
    String unit = getUnit();
    for (int i = 0; i < Array.getLength(units); i++) {
     if (units[i].equals(unit)) {
      unit = netListUnits[i];
     }
    }

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += "  " + getValue() + unit;

    return strNetlist;

   }

//public String getProbString();
}
```

Inductor.java

```
 package ECS.CircuitComponents.Pin2Components.PassiveComponents;

import java.lang.reflect.*;

import ECS.*;
import ECS.CircuitComponents.*;
```

```
public class Inductor
    extends Pin2Comp {

  private static final String[] units = {
      "\u00B5H",
      "mH", "H"};
  private static final String[] inputUnits = {
      "\u00B5H  (\u00B5H)",
      "mH (mH)", "H (H)"};
  private static final String[] netListUnits = {
      "UH",
      "MH", "H"};

  public int getComponentType() {
    return ActiveState.INDUCTOR;
  }

  public String getGenericName() {
    return "Inductor";
  }

  public String getName() {
    return "L";
  }

  public String getImageName() {
    return "SInductor.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return inputUnits;
  }

  public String getDefaultUnit() {
    return units[0];
  }

  public String getNetListString() {
    String unit = getUnit();
    for (int i = 0; i < Array.getLength(units); i++) {
```

```
      if (units[i].equals(unit)) {
        unit = netListUnits[i];
      }
    }

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += "  " + getValue() + unit;
    return strNetlist;

  }

  public String getCIRMLString() {
    String unit = getUnit();
    for (int i = 0; i < Array.getLength(units); i++) {
      if (units[i].equals(unit)) {
        unit = netListUnits[i];
      }
    }

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += "  " + getValue() + unit;
    return strNetlist;

  }

  public String getProbString() {
    return "";
  }
}
```

Capacitor.java

```
 package ECS.CircuitComponents.Pin2Components.PassiveComponents;

import java.lang.reflect.*;

import ECS.*;
```

```
import ECS.CircuitComponents.*;

public class Capacitor
    extends Pin2Comp {
  private static final String[] units = {
      "\u03C1F", "\u00B5F",
      "mF", "F"};
  private static final String[] inputUnits = {
      "\u03C1F (\u03C1F)", "\u00B5F (\u00B5F)",
      "mF (mF)", "F (F)"};
  private static final String[] netListUnits = {
      "PF", "UF", "MF", "F"};

  public int getComponentType() {
    return ActiveState.CAPACITOR;
  }

  public String getGenericName() {
    return "Capacitor";
  }

  public String getName() {
    return "C";
  }

  public String getImageName() {
    return "SCapacitor.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return inputUnits;
  }

  public String getDefaultUnit() {
    return units[0];
  }

  public String getNetListString() {

    String unit = getUnit();
    for (int i = 0; i < Array.getLength(units); i++) {
```

```
    if (units[i].equals(unit)) {
      unit = netListUnits[i];
    }
   }

   String strNetlist = "";
   strNetlist += getName() + "_" + getComponentName();
   strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
   strNetlist += "  " + getValue() + unit;
   return strNetlist;

  }

  public String getCIRMLString() {

   String unit = getUnit();
   for (int i = 0; i < Array.getLength(units); i++) {
    if (units[i].equals(unit)) {
      unit = netListUnits[i];
    }
   }

   String strNetlist = "";
   strNetlist += getName() + "_" + getComponentName();
   strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
   strNetlist += "  " + getValue() + unit;
   return strNetlist;

  }

}
```

DCVSource.java

```
package ECS.CircuitComponents.Pin2Components.IndependentSources;

import java.lang.reflect.*;

import ECS.*;
import ECS.CircuitComponents.*;
```

```
public class DCVSource
    extends Pin2Comp {

  private static final String[] units = {
      "\u00B5V", "mV", "V", "kV"};
  private static final String[] netlistUnits = {
      "UV",
      "MV", "V", "KV"};

  public int getComponentType() {
    return ActiveState.DCVSOURCE;
  }

  public String getGenericName() {
    return "DC Voltage Source";
  }

  public String getName() {
    return "Vdc";
  }

  public String getImageName() {
    return "SDCVsource.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return units;
  }

  public String getDefaultUnit() {
    return units[0];
  }

  public String getNetListString() {

    String unit = getUnit();
    String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
    for (int i = 0; i < Array.getLength(units); i++) {
      if (units[i].equals(unit)) {
        unit = netlistUnits[i];
      }
```

```
    }

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += "  " + negative + getValue() + unit;
    return strNetlist;

  }

  public String getCIRMLString() {
    String unit = getUnit();
    String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
    for (int i = 0; i < Array.getLength(units); i++) {
      if (units[i].equals(unit)) {
        unit = netlistUnits[i];
      }
    }

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += "  " + negative + getValue() + unit;
    return strNetlist;

  }

}
```

DCCSource.java

```
 package ECS.CircuitComponents.Pin2Components.IndependentSources;

import java.lang.reflect.*;

import ECS.*;
import ECS.CircuitComponents.*;

public class DCCSource
    extends Pin2Comp {

  private static final String[] units = {
```

```
    "\u00B5A", "mA", "A", "kA"};
private static final String[] netlistUnits = {
    "UA",
    "MA", "A", "KA"};

public int getComponentType() {
  return ActiveState.DCCSOURCE;
}

public String getGenericName() {
  return "DC Current Source";
}

public String getName() {
  return "Idc";
}

public String getImageName() {
  return "SDCCsource.jpg";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {
  String unit = getUnit();
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  for (int i = 0; i < Array.getLength(units); i++) {
    if (units[i].equals(unit)) {
      unit = netlistUnits[i];
    }
  }

  String strNetlist = "";
  strNetlist += getName() + "_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
```

```
    strNetlist += "  " + negative + getValue() + unit;
    return strNetlist;

  }

  public String getCIRMLString() {
    String unit = getUnit();
    String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
    for (int i = 0; i < Array.getLength(units); i++) {
      if (units[i].equals(unit)) {
        unit = netlistUnits[i];
      }
    }

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += "  " + negative + getValue() + unit;
    return strNetlist;

  }

}
```

ACVSource.java

```
 package ECS.CircuitComponents.Pin2Components.IndependentSources;

import java.lang.reflect.*;

import ECS.*;
import ECS.CircuitComponents.*;

public class ACVSource
    extends Pin2Comp {
  private static final String[] units = {
      "\u00B5V", "mV", "V", "kV"};
  private static final String[] netlistUnits = {
      "UV",
      "MV", "V", "KV"};
```

```
public ACVSource() {
  setValue(0.0);
  setFrequency(0);
  setPhase(0);
}

public int getComponentType() {
  return ActiveState.ACVSOURCE;
}

public String getGenericName() {
  return "AC Voltage Source";
}

public String getName() {
  return "Vac";
}

public String getImageName() {
  return "SACVsource.jpg";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {
  String negative=(getAlignment()==0||getAlignment()==1)?"-":"";
  String unit = getUnit();
  double phase=0;
  if(isCosine())
  {
  phase=getPhase()+45;
  }
  else
  {
  phase=getPhase();
```

```
    }
    for (int i = 0; i < Array.getLength(units); i++) {
      if (units[i].equals(unit)) {
        unit = netlistUnits[i];
      }
    }

    String strNetlist = "";
    strNetlist += getName() + "_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += "  " +getValue() + unit + "  " + getFrequency() +
        getFreqUnit()
        + "  " + negative+phase + getPhaseUnit();
    return strNetlist;

  }

  public String getCIRMLString() {
    String negative=(getAlignment()==0||getAlignment()==1)?"-":"+";
     String unit = getUnit();
     double phase=0;
if(isCosine())
{
phase=getPhase()+45;
}
else
{
phase=getPhase();
}


     for (int i = 0; i < Array.getLength(units); i++) {
       if (units[i].equals(unit)) {
         unit = netlistUnits[i];
       }
     }

     String strNetlist = "";
     strNetlist += getName() + "_" + getComponentName();
     strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
     strNetlist += " \n" +"+SIN  0  "+getValue() + unit + "  " + getFrequency() +
         getFreqUnit()
         + "  0  0  " + negative+phase + getPhaseUnit();
     return strNetlist;
```

```
  }

  // public String getACNetListString();
  //public string getProbString();
  // public void draw (GridCanvas gr,Component comp);
}
```

ACCSource.java

```
 package ECS.CircuitComponents.Pin2Components.IndependentSources;

import java.lang.reflect.*;

import ECS.*;
import ECS.CircuitComponents.*;

public class ACCSource
    extends Pin2Comp {

  private static final String[] units = {
      "\u00B5A", "mA", "A", "kA"};
  private static final String[] netlistUnits = {
      "UA",
      "MA", "A", "KA"};
  public ACCSource() {

  }

  public int getComponentType() {
    return ActiveState.ACCSOURCE;
  }

  public String getGenericName() {
    return "AC Current Source";
  }

  public String getName() {
    return "Iac";
  }

  public String getImageName() {
```

```
  return "SACCsource.jpg";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {
  String unit = getUnit();
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  for (int i = 0; i < Array.getLength(units); i++) {
    if (units[i].equals(unit)) {
      unit = netlistUnits[i];
    }
  }

  String strNetlist = "";
  strNetlist += getName() + "_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
  strNetlist += "  " + getValue() + unit + "  " + getFrequency() +
     getFreqUnit()
     + "  " + negative + getPhase() + getPhaseUnit();
  return strNetlist;

}

public String getCIRMLString() {
  String unit = getUnit();
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  for (int i = 0; i < Array.getLength(units); i++) {
    if (units[i].equals(unit)) {
      unit = netlistUnits[i];
    }
  }

  String strNetlist = "";
  strNetlist += getName() + "_" + getComponentName();
```

```
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName;
    strNetlist += " \n" + "+SIN  0  " + getValue() + unit + "  " + getFrequency() +
        getFreqUnit()
        + "  0  0  " + negative + getPhase() + getPhaseUnit();

    return strNetlist;

  }

  public String getACNetListString() {
    return "";
  }
  //public void getProbString()

}
```

DepVCVS.java

```
 package ECS.CircuitComponents.Pin2Components.DependentSources;

import ECS.*;
import ECS.CircuitComponents.Pin2Components.*;

public class DepVCVS
    extends DepComp {
  private static final String[] units = {
      "Null"};
  private static final String[] inputUnits = {
      ""};
  private final int numbControls = 2;
  public DepVCVS() {
    init(numbControls);
  }

  public int getComponentType() {
    return ActiveState.DEPVCVS;
  }

  public String getGenericName() {
    return "Voltage Controlled Voltage Source";
  }
```

```
public String getName() {
  return "E";
}

public String getImageName() {
  return "SVCVS.jpg";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return inputUnits[0];
}

public String getNetListString() {

  String strNetlist = "";
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  strNetlist += "E_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +
     "  " + controllers[0].getNode(0).strName + "  " +
     controllers[1].getNode(0).strName;

  strNetlist += "  " + negative + getValue();
  return strNetlist;

}

public String getCIRMLString() {

  String strNetlist = "";
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  strNetlist += "E_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +
     "  " + controllers[0].getNode(0).strName + "  " +
     controllers[1].getNode(0).strName;

  strNetlist += "  " + negative + getValue();
```

```
    return strNetlist;

  }

}
```

DepVCCS.java

```
 package ECS.CircuitComponents.Pin2Components.DependentSources;

import java.lang.reflect.*;

import ECS.*;
import ECS.CircuitComponents.Pin2Components.*;

public class DepVCCS
    extends DepComp {
  private static final String[] units = {
      "\u00B5S", "mS", "S", "kS", "MS"};
  private static final String[] netlistUnits = {
      "U",
      "M", "", "K", "MEG"};
  private final int numbControls = 2;
  public DepVCCS() {
    init(numbControls);
  }

  public int getComponentType() {
    return ActiveState.DEPVCCS;
  }

  public String getGenericName() {
    return "Voltage Controlled Current Source";
  }

  public String getName() {
    return "G";
  }

  public String getImageName() {
    return "SVCCS.jpg";
  }
```

```
public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return null;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {

  String unit = getUnit();
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  for (int i = 0; i < Array.getLength(units); i++) {
    if (units[i].equals(unit)) {
      unit = netlistUnits[i];
    }
  }

  String strNetlist = "";
  strNetlist += "G_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +
      "  " + controllers[0].getNode(0).strName + "  " +
      controllers[1].getNode(0).strName;

  strNetlist += "  " + negative + getValue() + getUnit();
  return strNetlist;

}

public String getCIRMLString() {

  String unit = getUnit();
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  for (int i = 0; i < Array.getLength(units); i++) {
    if (units[i].equals(unit)) {
      unit = netlistUnits[i];
    }
  }

  String strNetlist = "";
```

```
    strNetlist += "G_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +
        "  " + controllers[0].getNode(0).strName + "  " +
        controllers[1].getNode(0).strName;

    strNetlist += "  " + negative + getValue() + getUnit();
    return strNetlist;

  }

}
```

DepCCVS.java

```
 package ECS.CircuitComponents.Pin2Components.DependentSources;

import java.lang.reflect.*;

import ECS.*;
import ECS.CircuitComponents.Pin2Components.*;

public class DepCCVS
    extends DepComp {
  private static final String[] units = {
      "\u00B5Ohm", "mOhm", "Ohm", "kOhm", "MOhm"};
  private static final String[] netlistUnits = {
      "UOhm",
      "MOhm", "Ohm", "KOhm", "MEGOhm"};
  private final int numbControls = 1;
  public DepCCVS() {
    init(numbControls);
  }

  public int getComponentType() {
    return ActiveState.DEPCCVS;
  }

  public String getGenericName() {
    return "Current Controlled Voltage Source";
  }
```

```
public String getName() {
  return "H";
}

public String getImageName() {
  return "SCCVS.jpg";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {
  String strNetlist = "";
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  int direction = (controllers[0].getAlignment() == 0 ||
                controllers[0].getAlignment() == 1) ? 1 : 0;
  if (direction == 0) {
    strNetlist += "V" + controllers[0].getComponentName() + "  " +
       controllers[0].getNode(0).strName + "  " +
       controllers[0].getNode(1).strName + " 0V";
    strNetlist += "\nH_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +
       "  " +
       "V" + controllers[0].getComponentName();
    strNetlist += "  " + negative + getValue();

  }
  else {
    strNetlist += "V" + controllers[0].getComponentName() + "  " +
       controllers[0].getNode(1).strName + "  " +
       controllers[0].getNode(0).strName + " 0V";
    strNetlist += "\nH_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +
       "  " +
       "V" + controllers[0].getComponentName();
    strNetlist += "  " + negative + getValue();
```

```
    }

    return strNetlist;

  }

  public String getCIRMLString() {
    String strNetlist = "";
    String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
    int direction = (controllers[0].getAlignment() == 0 ||
                     controllers[0].getAlignment() == 1) ? 1 : 0;
    if (direction == 0) {
      strNetlist += "V" + controllers[0].getComponentName() + "  " +
          controllers[0].getNode(0).strName + "  " +
          controllers[0].getNode(1).strName + " 0V";
      strNetlist += "\nH_" + getComponentName();
      strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +
          "  " +
          "V" + controllers[0].getComponentName();
      strNetlist += "  " + negative + getValue();

    }
    else {
      strNetlist += "V" + controllers[0].getComponentName() + "  " +
          controllers[0].getNode(1).strName + "  " +
          controllers[0].getNode(0).strName + " 0V";
      strNetlist += "\nH_" + getComponentName();
      strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +
          "  " +
          "V" + controllers[0].getComponentName();
      strNetlist += "  " + negative + getValue();

    }

    return strNetlist;

  }

}
```

DepCCCS.java

```
 package ECS.CircuitComponents.Pin2Components.DependentSources;

import ECS.*;
import ECS.CircuitComponents.Pin2Components.*;

public class DepCCCS
    extends DepComp {
  private static final String[] units = {
      "Null"};
  private static final String[] inputUnits = {
      ""};
  private final int numbControls = 1;
  public DepCCCS() {
    init(numbControls);
  }

  public int getComponentType() {
    return ActiveState.DEPCCCS;
  }

  public String getGenericName() {
    return "Current Controlled Current Source";
  }

  public String getName() {
    return "F";
  }

  public String getImageName() {
    return "SCCCS.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return inputUnits;
  }

  public String getDefaultUnit() {
```

```
  return inputUnits[0];
}

public String getNetListString() {
  String strNetlist = "";
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  int direction = (controllers[0].getAlignment() == 0 ||
                controllers[0].getAlignment() == 1) ? 1 : 0;
  if (direction == 0) {
    strNetlist += "V" + controllers[0].getComponentName() +"  "+
       controllers[0].getNode(0).strName + "  " +
       controllers[0].getNode(1).strName + " 0V";
    strNetlist += "\nF_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +"  "+
       "V" + controllers[0].getComponentName();
    strNetlist += "  " + negative + getValue();

  }
  else {
    strNetlist += "V" + controllers[0].getComponentName() +"  "+
       controllers[0].getNode(1).strName + "  " +
       controllers[0].getNode(0).strName + " 0V";
    strNetlist += "\nF_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +"  "+
       "V" + controllers[0].getComponentName();
    strNetlist += "  " + negative + getValue();

  }

  return strNetlist;


}

public String getCIRMLString() {
  String strNetlist = "";
  String negative = (getAlignment() == 0 || getAlignment() == 1) ? "-" : "";
  int direction = (controllers[0].getAlignment() == 0 ||
                controllers[0].getAlignment() == 1) ? 1 : 0;
  if (direction == 0) {
    strNetlist += "V" + controllers[0].getComponentName() +"  "+
       controllers[0].getNode(0).strName + "  " +
       controllers[0].getNode(1).strName + " 0V";
    strNetlist += "\nF_" + getComponentName();
    strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +"  "+
```

```
        "V" + controllers[0].getComponentName();
      strNetlist += "  " + negative + getValue();

    }
    else {
      strNetlist += "V" + controllers[0].getComponentName() +"  "+
          controllers[0].getNode(1).strName + "  " +
          controllers[0].getNode(0).strName + " 0V";
      strNetlist += "\nF_" + getComponentName();
      strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName +"  "+
          "V" + controllers[0].getComponentName();
      strNetlist += "  " + negative + getValue();

    }

    return strNetlist;

  }

}
```

Transformer.java

```
package ECS.CircuitComponents.Pin4Components;

import ECS.*;
import ECS.CircuitComponents.*;

public class Transformer
    extends Pin4Comp {

  private static final String[] units = {
      ""};

  public int getComponentType() {
    return ActiveState.TRANSFORMER;
  }

  public String getGenericName() {
    return "Transformer";
  }

  public String getName() {
    return "X";
  }
```

```
  public String getImageName() {
    return "STransformer.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }

  public String[] getAllInputUnits() {
    return units;
  }

  public String getDefaultUnit() {
    return units[0];
  }

  public String getNetListString() {
    return "";
  }

  public String getCIRMLString() {
    return "";
  }

}
```

PPTransformer.java

```
 package ECS.CircuitComponents.Pin4Components;

import ECS.*;
import ECS.CircuitComponents.*;

public class PPTransformer
    extends Pin4Comp {
  final String[] units = {
      ""};
  private int n1, n2;
  public int getRation() {
    return n1;
```

```
}

public int getComponentType() {
  return ActiveState.TRANSFORMERA;
}

public String getGenericName() {
  return "PPTransformer";
}

public String getName() {
  return "T";
}

public String getImageName() {
  return "STransformerA.jpg";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {
  String strNetlist = "";
  strNetlist += "T_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName + "  " +
     getNode(2).strName + "  " + getNode(3).strName;
  strNetlist += "  " + getN1() + " " + getN2();
  return strNetlist;

}

public String getCIRMLString() {
  String strNetlist = "";
  double n1 = getN1();
  double n2 = getN2();
  if (n1 > n2) {
```

```
      double L1 = 20 * Math.pow(n1 / n2, 2);
      strNetlist += "L1  " + getNode(0).strName + "  " + getNode(2).strName +
          "  " + L1 + "H\n";
      strNetlist += "L2  " + getNode(1).strName + "  " + getNode(3).strName +
          "  20.0H\n";
      strNetlist += "K_" + getComponentName() + "  L1  L2  0.99999";

    }
    else {
      double L2 = 20 / Math.pow(n1 / n2, 2);
      strNetlist += "L1  " + getNode(0).strName + "  " + getNode(2).strName +
          "  20.0H\n";
      strNetlist += "L2  " + getNode(1).strName + "  " + getNode(3).strName +
          "  " + L2 + "H\n";
      strNetlist += "K_" + getComponentName() + "  L1  L2  0.99999";

    }
    return strNetlist;

  }

}
```

PNTransformer.java

```
 package ECS.CircuitComponents.Pin4Components;

import ECS.*;
import ECS.CircuitComponents.*;

public class PNTransformer
    extends Pin4Comp {

  private static final String[] units = {
      ""};

  public int getComponentType() {
    return ActiveState.TRANSFORMERB;
  }

  public String getGenericName() {
```

```
  return "PNTransformer";
}

public String getName() {
  return "T";
}

public String getImageName() {
  return "STransformerB.jpg";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {
  String strNetlist = "";
  strNetlist += "T_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName + "  " +
     getNode(2).strName + "  " + getNode(3).strName;
  strNetlist += "  " + getN1() + "  " + getN2();
  return strNetlist;

}

public String getCIRMLString() {
  String strNetlist = "";
  double n1 = getN1();
  double n2 = getN2();
  if (n1 > n2) {
    double L1 = 20 * Math.pow(n1 / n2, 2);
    strNetlist += "L1  " + getNode(0).strName + "  " + getNode(2).strName +
       "  " + L1 + "H\n";
    strNetlist += "L2  " + getNode(1).strName + "  " + getNode(3).strName +
       "  20.0H\n";
    strNetlist += "K_" + getComponentName() + "  L1  L2  0.99999";
```

```
    }
    else {
      double L2 = 20 / Math.pow(n1 / n2, 2);
      strNetlist += "L1  " + getNode(0).strName + "  " + getNode(2).strName +
          "  20.0H\n";
      strNetlist += "L2  " + getNode(1).strName + "  " + getNode(3).strName +
          "  " + L2 + "H\n";
      strNetlist += "K_" + getComponentName() + "  L1  L2  0.99999";

    }
    return strNetlist;

  }

}
```

NPTransformer.java

```
 package ECS.CircuitComponents.Pin4Components;

import ECS.*;
import ECS.CircuitComponents.*;

public class NPTransformer
    extends Pin4Comp {

  private static final String[] units = {
      ""};

  public int getComponentType() {
    return ActiveState.TRANSFORMERC;
  }

  public String getGenericName() {
    return "NPTransformer";
  }

  public String getName() {
    return "T";
  }
```

```
public String getImageName() {
  return "STransformerC.jpg";
}

public String[] getAllUnits() {
  return units;
}

public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {
  String strNetlist = "";
  strNetlist += "T_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName + "  " +
     getNode(2).strName + "  " + getNode(3).strName;
  strNetlist += "  " + getN1() + "  " + getN2();
  return strNetlist;

}

public String getCIRMLString() {
  String strNetlist = "";
  double n1 = getN1();
  double n2 = getN2();
  if (n1 > n2) {
    double L1 = 20 * Math.pow(n1 / n2, 2);
    strNetlist += "L1  " + getNode(0).strName + "  " + getNode(2).strName +
       "  " + L1 + "H\n";
    strNetlist += "L2  " + getNode(1).strName + "  " + getNode(3).strName +
       "  20.0H\n";
    strNetlist += "K_" + getComponentName() + "  L1  L2  0.99999";

  }
  else {
    double L2 = 20 / Math.pow(n1 / n2, 2);
    strNetlist += "L1  " + getNode(0).strName + "  " + getNode(2).strName +
       "  20.0H\n";
    strNetlist += "L2  " + getNode(1).strName + "  " + getNode(3).strName +
       "  " + L2 + "H\n";
```

```
      strNetlist += "K_" + getComponentName() + "  L1  L2  0.99999";

    }
    return strNetlist;

  }

}
```

NNTransformer.java

```
 package ECS.CircuitComponents.Pin4Components;

import ECS.*;
import ECS.CircuitComponents.*;

public class NNTransformer
    extends Pin4Comp {

  private static final String[] units = {
      ""};

  public int getComponentType() {
    return ActiveState.TRANSFORMERD;
  }

  public String getGenericName() {
    return "NNTransformer";
  }

  public String getName() {
    return "T";
  }

  public String getImageName() {
    return "STransformerD.jpg";
  }

  public String[] getAllUnits() {
    return units;
  }
```

```
public String[] getAllInputUnits() {
  return units;
}

public String getDefaultUnit() {
  return units[0];
}

public String getNetListString() {
  String strNetlist = "";
  strNetlist += "T_" + getComponentName();
  strNetlist += "  " + getNode(0).strName + "  " + getNode(1).strName + "  " +
      getNode(2).strName + "  " + getNode(3).strName;
  strNetlist += "  " + getN1() + "  " + getN2();
  return strNetlist;

}

public String getCIRMLString() {
  String strNetlist = "";
  double n1 = getN1();
  double n2 = getN2();
  if (n1 > n2) {
    double L1 = 20 * Math.pow(n1 / n2, 2);
    strNetlist += "L1  " + getNode(0).strName + "  " + getNode(2).strName +
        "  " + L1 + "H\n";
    strNetlist += "L2  " + getNode(1).strName + "  " + getNode(3).strName +
        "  20.0H\n";
    strNetlist += "K_" + getComponentName() + "  L1  L2  0.99999";

  }
  else {
    double L2 = 20 / Math.pow(n1 / n2, 2);
    strNetlist += "L1  " + getNode(0).strName + "  " + getNode(2).strName +
        "  20.0H\n";
    strNetlist += "L2  " + getNode(1).strName + "  " + getNode(3).strName +
        "  " + L2 + "H\n";
    strNetlist += "K_" + getComponentName() + "  L1  L2  0.99999";

  }
  return strNetlist;

}
```

```
}
```

Circle.java

```
 package ECS.DrawingTools;

import java.awt.*;

import ECS.GUPanels.*;

public class Circle
   extends DrawTool {

  public Circle(int x, int y, int w, int h) {

    int[] pos = {
       x, y};
    int[] size = {
       w, h};
    setPosition(pos);
    setSize(size);
    setName("Circle");

  }

  public boolean occupiedArea(int[] pos) {

    int[] size = getSize();
    int[] xyPos = getPosition();
    if ( (pos[0] >= xyPos[0] && pos[0] <= xyPos[0] + size[0]) &&
       (pos[1] >= xyPos[1] && pos[1] <= xyPos[1] + size[1])) {
      setActiveState(true);
      return true;
    }
    else {
      setActiveState(false);
      return false;
    }

  }
```

```
  public boolean mouseOverArea(int[] pos) {
    int[] size = getSize();
    int[] xyPos = getPosition();
    if ( (pos[0] >= xyPos[0] && pos[0] <= xyPos[0] + size[0]) &&
        (pos[1] >= xyPos[1] && pos[1] <= xyPos[1] + size[1])) {

      return true;
    }
    else {

      return false;
    }

  }

  public void drawComponent(GridCanvas grid) {

    int[] xyPos = getPosition();
    int[] size = getSize();

    Color fill = getFillColor();
    Color line = getLineColor();

    grid.gridImage.setColor(fill);
    grid.gridImage.fillOval(xyPos[0], xyPos[1], size[0], size[1]);
    grid.gridImage.setColor(line);
    grid.gridImage.drawOval(xyPos[0], xyPos[1], size[0], size[1]);

    if (getActiveState()) {
      grid.gridImage.setColor(Color.RED);
      grid.gridImage.drawRect(xyPos[0] - 2, xyPos[1] - 2, 5, 5);
      grid.gridImage.drawRect(xyPos[0] - 2, xyPos[1] - 2 + size[1], 5, 5);
      grid.gridImage.drawRect(xyPos[0] - 2 + size[0], xyPos[1] - 2, 5, 5);
      grid.gridImage.drawRect(xyPos[0] - 2 + size[0], xyPos[1] - 2 + size[1], 5,
                              5);

    }

  }
}
```

Line.java

```
 package ECS.DrawingTools;

import java.awt.*;

import ECS.GUPanels.*;

public class Line
    extends DrawTool {

  public Line(int[] p1, int[] p2) {
    setStartPoint(p1);
    setEndPoint(p2);
    setName("Line");

  }

  public boolean occupiedArea(int[] pos) {
    int a, b, c;

    // int[] points=getPoints();
    int[] p1 = getStartPoint();
    int[] p2 = getEndPoint();
    int[] xy = pos;
    a = - (p2[1] - p1[1]);
    b = (p2[0] - p1[0]);
    c = (p2[1] - p1[1]) * p1[0] - (p2[0] - p1[0]) * p1[1];
    double distance = Math.abs(a * pos[0] + b * pos[1] + c) /
        Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

    if (p1[0] == p2[0]) {
      if (distance < 3 &&
          ( ( (pos[1] >= p1[1] && pos[1] <= p2[1]) ||
             (pos[1] >= p2[1] && pos[1] <= p1[1])))) {

        setActiveState(true);
        return true;
      }
      else {
        setActiveState(false);
        return false;
      }
```

```
  }
  else {
    if (distance < 3 &&
        ( ( (pos[0] >= p1[0] && pos[0] <= p2[0]) ||
           (pos[0] >= p2[0] && pos[0] <= p1[0])))) {
      setActiveState(true);
      return true;
    }
    else {
      setActiveState(false);
      return false;
    }
  }

}

public boolean mouseOverArea(int[] pos) {
  int a, b, c;

  // int[] points=getPoints();
  int[] p1 = getStartPoint();
  int[] p2 = getEndPoint();
  int[] xy = pos;
  a = - (p2[1] - p1[1]);
  b = (p2[0] - p1[0]);
  c = (p2[1] - p1[1]) * p1[0] - (p2[0] - p1[0]) * p1[1];
  double distance = Math.abs(a * pos[0] + b * pos[1] + c) /
      Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

  if (p1[0] == p2[0]) {
    if (distance < 3 &&
        ( ( (pos[1] >= p1[1] && pos[1] <= p2[1]) ||
           (pos[1] >= p2[1] && pos[1] <= p1[1])))) {

      return true;
    }
    else {
      return false;
    }

  }
  else {
    if (distance < 3 &&
        ( ( (pos[0] >= p1[0] && pos[0] <= p2[0]) ||
```

```
            (pos[0] >= p2[0] && pos[0] <= p1[0]))) {
        return true;
      }
      else {
        return false;
      }
    }

  }

  public void drawComponent(GridCanvas grid) {

    Color line = getLineColor();
    int[] startPoint = getStartPoint();
    int[] endPoint = getEndPoint();

    if (getActiveState()) {
      grid.gridImage.setColor(Color.RED);
    }
    else {
      grid.gridImage.setColor(line);
    }
    grid.gridImage.drawLine(startPoint[0], startPoint[1], endPoint[0],
                            endPoint[1]);
    if (getActiveState()) {

      grid.gridImage.setColor(Color.RED);
      grid.gridImage.drawRect(startPoint[0] - 2, startPoint[1] - 2, 5, 5);
      grid.gridImage.drawRect(endPoint[0] - 2, endPoint[1] - 2, 5, 5);

    }

  }

}
```

Rectangle.java

```
 package ECS.DrawingTools;

import java.awt.*;
```

```
import ECS.GUPanels.*;

public class Rectangle
   extends DrawTool {

  public Rectangle(int x, int y, int w, int h) {
    int[] xyPos = {
       x, y};
    int[] size = {
       w, h};
    setPosition(xyPos);
    setSize(size);
    setName("Rectangle");
  }

  public boolean mouseOverArea(int[] pos) {
    int[] xyPos = getPosition();
    int[] size = getSize();

    if ( (pos[0] >= xyPos[0] && pos[0] <= xyPos[0] + size[0]) &&
       (pos[1] >= xyPos[1] && pos[1] <= xyPos[1] + size[1])) {

      return true;
    }
    else {

      return false;
    }
  }

  public boolean occupiedArea(int[] pos) {
    int[] xyPos = getPosition();
    int[] size = getSize();
    if ( (pos[0] >= xyPos[0] && pos[0] <= xyPos[0] + size[0]) &&
       (pos[1] >= xyPos[1] && pos[1] <= xyPos[1] + size[1])) {
      setActiveState(true);
      return true;
    }
    else {
      setActiveState(false);
      return false;
    }

  }
```

```
  public void drawComponent(GridCanvas grid) {
    boolean round = getRoundType();
    Color fill = getFillColor();
    Color line = getLineColor();
    int[] xyPos = getPosition();
    int[] size = getSize();

    if (!round) {
      grid.gridImage.setColor(fill);
      grid.gridImage.fillRect(xyPos[0], xyPos[1], size[0], size[1]);
      grid.gridImage.setColor(line);
      grid.gridImage.drawRect(xyPos[0], xyPos[1], size[0], size[1]);
    }
    else {
      grid.gridImage.setColor(fill);
      grid.gridImage.fillRoundRect(xyPos[0], xyPos[1], size[0], size[1], 20, 20);
      grid.gridImage.setColor(line);
      grid.gridImage.drawRoundRect(xyPos[0], xyPos[1], size[0], size[1], 20, 20);
    }
    if (getActiveState()) {

      grid.gridImage.setColor(Color.RED);
      grid.gridImage.drawRect(xyPos[0] - 2, xyPos[1] - 2, 5, 5);
      grid.gridImage.drawRect(xyPos[0] - 2, xyPos[1] - 2 + size[1], 5, 5);
      grid.gridImage.drawRect(xyPos[0] - 2 + size[0], xyPos[1] - 2, 5, 5);
      grid.gridImage.drawRect(xyPos[0] - 2 + size[0], xyPos[1] - 2 + size[1], 5,
                              5);

    }

  }
}
```

TextBox.java

```
 package ECS.DrawingTools;

import java.util.*;

import java.awt.*;
```

```
import ECS.*;
import ECS.GUPanels.*;

public class TextBox
   extends DrawTool {
  public Vector lines;
  private int vSpace = 0;
  private int maxWidth = 0;
  private int maxHeight = 0;
  public String text = "";

  public TextBox(Vector txtVector, int[] pos) {
    lines = txtVector;
    setPosition(pos);

    for (int i = 0; i < lines.size(); i++) {
      text += (String) lines.get(i);
      String a = (String) lines.get(i);
      if (a.length() > maxWidth) {

        maxWidth = a.length();

      }
    }
    maxHeight = 15 * lines.size();
    maxWidth = 12 * maxWidth;

    setName("Text");

  }

  public boolean occupiedArea(int[] pos) {
    int[] xyPos = getPosition();
    if ( (pos[0] > xyPos[0] && pos[0] < xyPos[0] + maxWidth) &&
        (pos[1] > xyPos[1] - 10 && pos[1] < xyPos[1] + maxHeight + 5)) {
      setActiveState(true);
      return true;
    }
    else {
      setActiveState(false);
      return false;
    }

  }
```

```
public boolean mouseOverArea(int[] pos) {
  int[] xyPos = getPosition();
  if ( (pos[0] > xyPos[0] && pos[0] < xyPos[0] + maxWidth) &&
      (pos[1] > xyPos[1] - 10 && pos[1] < xyPos[1] + maxHeight + 5)) {

    return true;
  }
  else {

    return false;
  }

}

public void calculateDimension() {
  for (int i = 0; i < lines.size(); i++) {
    text += (String) lines.get(i);
    String a = (String) lines.get(i);
    if (a.length() > maxWidth) {

      maxWidth = a.length();

    }
  }
  maxHeight = 15 * lines.size();
  maxWidth = 6 * maxWidth;

}

public void drawComponent(GridCanvas grid) {

  Color textColor = getTextColor();
  int[] xyPos = getPosition();
  mainApplet.gridCanvas.gridImage.setColor(textColor);
  vSpace = 0;
  if (!getActiveState()) {
    for (int i = 0; i < lines.size(); i++) {
      String str = (String) lines.get(i);
      mainApplet.gridCanvas.gridImage.drawString(str, xyPos[0],
          xyPos[1] + vSpace);
      vSpace += 15;
    }
  }
  else {
```

```
      mainApplet.gridCanvas.gridImage.setColor(Color.RED);
      for (int i = 0; i < lines.size(); i++) {
        String str = (String) lines.get(i);

        mainApplet.gridCanvas.gridImage.drawString(str, xyPos[0],
           xyPos[1] + vSpace);
        vSpace += 15;
      }
      //mainApplet.gridCanvas.gridImage.drawRect(xyPos[0] - 5, xyPos[1] - 15,
      // maxWidth, maxHeight + 10);
    }
  }

}
```

DrawTool.java

```
 package ECS.DrawingTools;

import java.io.*;

import java.awt.*;

import ECS.GUPanels.*;

public abstract class DrawTool
    implements Serializable {

  private boolean roundShape = false;
  private Color fillColor = Color.WHITE;
  private Color lineColor = Color.BLUE;
  private Color textColor = Color.BLUE;
  private boolean activeState = false;
  private int width, height;
  private int[] xyPos = new int[2];
  private int[] startPoint = new int[2];
  private int[] endPoint = new int[2];
  private String text = "";
  private String name;

  public DrawTool() {
```

```
}

abstract public boolean occupiedArea(int[] pos);

abstract public boolean mouseOverArea(int[] pos);

abstract public void drawComponent(GridCanvas grid);

public void setRoundType(boolean round) {
  roundShape = round;
}

public boolean getRoundType() {
  return roundShape;
}

public void setFillColor(Color fill) {
  fillColor = fill;
}

public Color getFillColor() {
  return fillColor;
}

public void setLineColor(Color line) {
  lineColor = line;
}

public Color getLineColor() {
  return lineColor;
}

public void setTextColor(Color text) {
  textColor = text;
}

public Color getTextColor() {
  return textColor;
}

public void setText(String txt) {
  text = txt;
}
```

```
public String getText() {
  return text;
}

public void setActiveState(boolean state) {
  activeState = state;
}

public boolean getActiveState() {
  return activeState;
}

public void setPosition(int[] pos) {
  xyPos[0] = pos[0];
  xyPos[1] = pos[1];
}

public int[] getPosition() {
  return xyPos;
}

public void setSize(int[] size) {
  width = size[0];
  height = size[1];
}

public int[] getSize() {
  int[] size = {
      width, height};
  return size;
}

public void setStartPoint(int[] p) {

  startPoint[0] = p[0];
  startPoint[1] = p[1];
}

public void setEndPoint(int[] p) {

  endPoint[0] = p[0];
  endPoint[1] = p[1];
}

public int[] getStartPoint() {
```

```
    return startPoint;
  }

  public int[] getEndPoint() {

    return endPoint;
  }

  public void setName(String nm) {
    name = nm;
  }

  public String getName() {
    return name;
  }
}
```

CanvasPropertiesFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;
import ECS.GUIMessages.*;
import ECS.GUPanels.*;

public class CanvasPropertiesFrame
    extends CloseableFrame {

  JButton canvasFillColorBtn, canvasLineColorBtn, applyBtn, cancelBtn;
  JLabel canvasColorLbl, canvasSizeLbl, backColorLbl, ForColorLbl, widthLbl,
      heightLbl,
      widthPixelsLbl, heightPixelsLbl, back1Lbl, back2Lbl;
  JTextField widthTxt, heightTxt;
  JPanel ColorPanel, sizePanel;
  public static JTextField canvasBackColorTxt, canvasForColorTxt;
  JTabbedPane tPanel;
```

```
JColorChooser cs;
mainApplet myApplet;

public CanvasPropertiesFrame(mainApplet applet, int tap) {

  this.setSize(310, 240);
  myApplet = applet;

  this.setDefaultLookAndFeelDecorated(true);
  ColorPanel = new JPanel();
  ColorPanel.setLayout(null);
  ColorPanel.setBackground(Color.WHITE);

  sizePanel = new JPanel();
  sizePanel.setLayout(null);
  sizePanel.setBackground(Color.WHITE);

  Image img = ActiveState.diffImages[ActiveState.MSGBACKGROUND];
  back1Lbl = new JLabel(new ImageIcon(img));
  back2Lbl = new JLabel(new ImageIcon(img));

  tPanel = new JTabbedPane();
  tPanel.setBounds(10, 10, 280, 130);
  tPanel.setBackground(Color.WHITE);
  tPanel.setForeground(Color.BLUE);

  this.setTitle("Canvas Properties");
  this.getContentPane().setLayout(null);
  this.getContentPane().setBackground(new Color(91, 141, 255));
  this.getContentPane().add(tPanel);
  this.setResizable(false);

  canvasColorLbl = new JLabel("[Canvas Colors]");
  canvasColorLbl.setForeground(Color.BLUE);

  canvasSizeLbl = new JLabel("[Canvas Size]");
  canvasSizeLbl.setForeground(Color.BLUE);
  backColorLbl = new JLabel("Background Color");
  backColorLbl.setForeground(Color.BLUE);
  ForColorLbl = new JLabel("Forground Color");
  ForColorLbl.setForeground(Color.BLUE);
  widthLbl = new JLabel("Width");
  widthLbl.setForeground(Color.BLUE);
  heightLbl = new JLabel("Height");
  heightLbl.setForeground(Color.BLUE);
```

```
widthTxt = new JTextField();

Integer initWidth = new Integer(GridCanvas.getGridWidth());

widthTxt.setText(initWidth.toString());
heightTxt = new JTextField();
Integer initHeight = new Integer(GridCanvas.getGridHeight());
heightTxt.setText(initHeight.toString());
widthPixelsLbl = new JLabel("Pixels");
widthPixelsLbl.setForeground(Color.BLUE);
heightPixelsLbl = new JLabel("Pixels");
heightPixelsLbl.setForeground(Color.BLUE);

img = ActiveState.btnsExited[ActiveState.FILLCOLOR];
canvasFillColorBtn = new JButton(new ImageIcon(img));
canvasFillColorBtn.setBorder(BorderFactory.createRaisedBevelBorder());
canvasFillColorBtn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    canvasFillColorBtn_actionPerformed(ae);
  }
}
);

img = ActiveState.btnsExited[ActiveState.LINECOLOR];
canvasLineColorBtn = new JButton(new ImageIcon(img));
canvasLineColorBtn.setBorder(BorderFactory.createRaisedBevelBorder());
canvasLineColorBtn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    canvasLineColorBtn_actionPerformed(ae);
  }
}
);

applyBtn = new JButton("Apply");
applyBtn.setBackground(Color.WHITE);
applyBtn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    applyBtn_actionPerformed(ae);
  }
}
);

cancelBtn = new JButton("Cancel");
cancelBtn.setBackground(Color.WHITE);
cancelBtn.addActionListener(new ActionListener() {
```

```
  public void actionPerformed(ActionEvent ae) {
    cancelBtn_actionPerformed(ae);
  }
 }
 );

 canvasBackColorTxt = new JTextField();
 canvasBackColorTxt.setEditable(false);
 canvasBackColorTxt.setBorder(BorderFactory.createLoweredBevelBorder());
 canvasBackColorTxt.setBackground(mainApplet.canvasBackgroundColor);

 canvasForColorTxt = new JTextField();
 canvasForColorTxt.setEditable(false);
 canvasForColorTxt.setBorder(BorderFactory.createLoweredBevelBorder());
 canvasForColorTxt.setBackground(mainApplet.canvasForegroundColor);

 if (tap == 0) {
  tPanel.add("Canvas Colors", ColorPanel);
  tPanel.add("Canvas Size", sizePanel);
 }
 else {
  tPanel.add("Canvas Size", sizePanel);
  tPanel.add("Canvas Colors", ColorPanel);
 }
 addToPanel(ColorPanel, canvasColorLbl, 20, 5, 250, 20);
 addToPanel(ColorPanel, backColorLbl, 60, 30, 70, 20);
 addToPanel(ColorPanel, canvasFillColorBtn, 120, 30, 20, 20);
 addToPanel(ColorPanel, canvasBackColorTxt, 145, 30, 40, 20);
 addToPanel(ColorPanel, ForColorLbl, 60, 60, 70, 20);
 addToPanel(ColorPanel, canvasLineColorBtn, 120, 60, 20, 20);
 addToPanel(ColorPanel, canvasForColorTxt, 145, 60, 40, 20);
 addToPanel(ColorPanel, back1Lbl, 0, 0, 280, 130);
 addToPanel(sizePanel, canvasSizeLbl, 20, 5, 250, 20);
 addToPanel(sizePanel, widthLbl, 60, 30, 70, 20);
 addToPanel(sizePanel, widthTxt, 125, 30, 35, 20);
 addToPanel(sizePanel, widthPixelsLbl, 170, 30, 35, 20);
 addToPanel(sizePanel, heightLbl, 60, 60, 70, 20);
 addToPanel(sizePanel, heightTxt, 125, 60, 35, 20);
 addToPanel(sizePanel, heightPixelsLbl, 170, 60, 35, 20);
 addToPanel(sizePanel, back2Lbl, 0, 0, 280, 130);
 add(cancelBtn, 150, 150, 80, 20);
 add(applyBtn, 70, 150, 80, 20);
 add(this.backgroundLbl, -100, -250, 500, 650);

}
```

```
  private void canvasFillColorBtn_actionPerformed(ActionEvent ae) {
    ColorPanel background = new ColorPanel("canvasBackgroundColor", myApplet);
    background.show();
  }

  private void canvasLineColorBtn_actionPerformed(ActionEvent ae) {
    ColorPanel Foreground = new ColorPanel("canvasForgroundColor", myApplet);
    Foreground.show();
  }

  private void applyBtn_actionPerformed(ActionEvent ae) {
    Integer width = new Integer(widthTxt.getText());
    Integer height = new Integer(heightTxt.getText());

    if ( (width.intValue() >= 650 && width.intValue() <= 2500) &&
        (height.intValue() >= 500 && height.intValue() <= 2500)) {
      myApplet.updateCanvas(width.intValue(), height.intValue(),
                            canvasBackColorTxt.getBackground()
                            , canvasForColorTxt.getBackground());

      this.dispose();
    }
    else {
      AlertMessage errMsg = new AlertMessage(
          "Width and Height must be between 650-2500,500-2500 respectively");
      errMsg.show();
    }

  }

  private void cancelBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

CloseableFrame.java

```
package ECS.GUIFrameWindows;
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;

//*************************************************************************************/
/
//                        ClosableFrame Class                          //
//*************************************************************************************/
/
public class CloseableFrame
   extends JFrame {
////////////////////////////////////////////////////////////////////////////
  public JLabel backgroundLbl;

  public CloseableFrame() {
    this.getContentPane().setLayout(null);
    this.getContentPane().setBackground(Color.WHITE);
    this.setResizable(true);
    this.setLocation(200, 200);
    this.setResizable(false);

    this.setDefaultLookAndFeelDecorated(true);

    this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        setVisible(false);
        dispose();
      }
    });
    Image img = ActiveState.diffImages[ActiveState.MSGBACKGROUND];
    backgroundLbl = new JLabel(new ImageIcon(img));

  }

////////////////////////////////////////////////////////////////////////////
  public void add(Component comp, int x, int y, int width, int height) {
    comp.setBounds(x, y, width, height);
    this.getContentPane().add(comp);

  }

////////////////////////////////////////////////////////////////////////////
  public void addToPanel(JPanel pnl, Component comp, int x, int y, int width,
```

```
                    int height) {
    comp.setBounds(new Rectangle(x, y, width, height));
    pnl.add(comp);
   }
////////////////////////////////////////////////////////////////////////////
}
//***************************************************************************/
/
//                          END                              //
//***************************************************************************/
/
```

CurrentInputFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

import ECS.*;
import ECS.CircuitComponents.*;

public class CurrentInputFrame
    extends CloseableFrame {

  JLabel lblLabel;
  JTextField txtLabel;
  JButton oky;
  Components curComp;
  mainApplet myApplet;
  public CurrentInputFrame() {
    Border border = BorderFactory.createEtchedBorder(Color.white, Color.blue);

    this.setSize(300, 210);
    lblLabel = new JLabel("Label:");
    lblLabel.setBackground(SystemColor.control);
    lblLabel.setForeground(Color.blue);

    txtLabel = new JTextField();
```

```
  oky = new JButton("OK");
  oky.setBackground(Color.white);
  oky.setForeground(Color.blue);
  oky.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent me) {
      oky_mouseClicked(me);
    }
  }
  );
  oky.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });
  txtLabel.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });

  add(lblLabel, 10, 20, 34, 13);

  add(txtLabel, 60, 20, 86, 21);

  add(oky, 100, 100, 73, 25);
  add(this.backgroundLbl, -100, -250, 500, 650);
}

public void setComponent(Components comp, mainApplet applet) {
  curComp = comp;
  myApplet = applet;
  this.setTitle(comp.getGenericName());
  txtLabel.setText(comp.getComponentName());

}

private void oky_mouseClicked(MouseEvent me) {
  this.setVisible(false);
  String txtlbl = txtLabel.getText();
  curComp.setComponentName(txtlbl);
  myApplet.gridCanvas.gridImage.clearRect(0, 0, 650, 550);
  myApplet.gridCanvas.drawGridLayout(ActiveState.startPoint);
  myApplet.drawAllComps();
  myApplet.gridCanvas.redraw();
}
```

```
  private void oky_keyPressed(KeyEvent ke) {
    if (ke.getKeyCode() == 10) {
      this.setVisible(false);
      String txtlbl = txtLabel.getText();
      curComp.setComponentName(txtlbl);
      myApplet.redrawGridCanvas();
    }
  }

}
```

DisplayPropertiesFrame

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.ActiveState;
import ECS.mainApplet;

public class DisplayPropertiesFrame
    extends CloseableFrame {

  ButtonGroup displayGrp, rotationGrp;
  JRadioButton doNotDisplay, valueOnly, nameANDValue, nameOnly, deg0, deg90,
      deg180, deg270,
      deg_90, deg_180, deg_270;
  JLabel displayLbl, rotationLbl, bgColorLbl, forColorLbl, BGColor, FORColor;
  JButton applyBtn, cancelBtn;
  JTextField bgColor, forColor;
  mainApplet myApplet;

  public DisplayPropertiesFrame(mainApplet applet) {

    this.setTitle("Display Properties");
    this.setSize(300, 300);
    this.getContentPane().setBackground(new Color(139, 166, 255));

    myApplet = applet;
```

```
applyBtn = new JButton("Apply");
applyBtn.setBounds(60, 180, 90, 20);
applyBtn.setBackground(Color.WHITE);
applyBtn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    applyBtn_actionPerformed(ae);
  }
});

cancelBtn = new JButton("Cancel");
cancelBtn.setBounds(150, 180, 90, 20);
cancelBtn.setBackground(Color.WHITE);
cancelBtn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    cancelBtn_actionPerformed(ae);
  }
});

displayGrp = new ButtonGroup();
rotationGrp = new ButtonGroup();

doNotDisplay = new JRadioButton("Do Not Display");
valueOnly = new JRadioButton("Value Only");
nameOnly = new JRadioButton("Name Only");
nameANDValue = new JRadioButton("Name and Value");
int displayFormat = myApplet.getNameANDValueDisplayFormat();
switch (displayFormat) {
  case 0:
    doNotDisplay.setSelected(true);
    break;
  case 1:
    valueOnly.setSelected(true);
    break;
  case 2:
    nameOnly.setSelected(true);
    break;
  case 3:
    nameANDValue.setSelected(true);
    break;
}
;

deg0 = new JRadioButton("0\'");
deg0.setSelected(true);
```

```
deg90 = new JRadioButton("90\'");
deg180 = new JRadioButton("180\'");
deg270 = new JRadioButton("270\'");
deg_90 = new JRadioButton("-90\'");
deg_180 = new JRadioButton("-180\'");
deg_270 = new JRadioButton("-270\'");
deg0.setSelected(true);

displayGrp.add(doNotDisplay);
displayGrp.add(nameOnly);
displayGrp.add(valueOnly);
displayGrp.add(nameANDValue);

rotationGrp.add(deg0);
rotationGrp.add(deg90);
rotationGrp.add(deg180);
rotationGrp.add(deg270);
rotationGrp.add(deg_90);
rotationGrp.add(deg_180);
rotationGrp.add(deg_270);

displayLbl = new JLabel("[Display Format]...");
rotationLbl = new JLabel("[Rotation]...");

//Image img=ActiveState.btnsExited[ActiveState.FILLCOLOR];
//bgColorBtn=new JButton(new ImageIcon(img));
//img=ActiveState.btnsExited[ActiveState.LINECOLOR];
//forColorBtn=new JButton(new ImageIcon(img));

bgColor = new JTextField();
forColor = new JTextField();

displayLbl.setBounds(10, 10, 120, 20);
doNotDisplay.setBounds(10, 40, 120, 20);
valueOnly.setBounds(10, 60, 120, 20);
nameOnly.setBounds(10, 80, 120, 20);
nameANDValue.setBounds(10, 100, 120, 20);

rotationLbl.setBounds(170, 10, 60, 20);
deg_90.setBounds(170, 80, 70, 20);
deg_180.setBounds(170, 60, 70, 20);
deg_270.setBounds(170, 40, 70, 20);
deg0.setBounds(170, 100, 70, 20);
deg90.setBounds(240, 40, 70, 20);
deg180.setBounds(240, 60, 70, 20);
```

```
    deg270.setBounds(240, 80, 70, 20);

    //bgColorLbl.setBounds(20,150,90,20);
    //bgColorBtn.setBounds(110,150,25,20);
    //bgColor.setBounds(135,150,25,20);
    //forColorLbl.setBounds(20,180,90,20);
    //forColorBtn.setBounds(110,180,25,20);
    //forColor.setBounds(135,180,25,20);

    displayLbl.setBackground(new Color(139, 166, 255));
    doNotDisplay.setBackground(new Color(139, 166, 255));
    valueOnly.setBackground(new Color(139, 166, 255)); ;
    nameOnly.setBackground(new Color(139, 166, 255));
    nameANDValue.setBackground(new Color(139, 166, 255));
    deg0.setBackground(new Color(139, 166, 255));
    deg90.setBackground(new Color(139, 166, 255));
    deg180.setBackground(new Color(139, 166, 255));
    deg270.setBackground(new Color(139, 166, 255));
    deg_90.setBackground(new Color(139, 166, 255));
    deg_180.setBackground(new Color(139, 166, 255));
    deg_270.setBackground(new Color(139, 166, 255));

    this.getContentPane().add(displayLbl);
    this.getContentPane().add(doNotDisplay);
    this.getContentPane().add(valueOnly);
    this.getContentPane().add(nameOnly);
    this.getContentPane().add(nameANDValue);
    this.getContentPane().add(deg0);
    this.getContentPane().add(deg90);
    this.getContentPane().add(deg180);
    this.getContentPane().add(deg270);
    this.getContentPane().add(rotationLbl);
// this.getContentPane().add(bgColorLbl);
    //this.getContentPane().add(bgColorBtn);
    //this.getContentPane().add(forColorLbl);
    //this.getContentPane().add(forColorBtn);
    //this.getContentPane().add(bgColor);
    //this.getContentPane().add(forColor);
    this.getContentPane().add(applyBtn);
    this.getContentPane().add(cancelBtn);
    this.getContentPane().add(deg_90);
    this.getContentPane().add(deg_180);
    this.getContentPane().add(deg_270);

    add(this.backgroundLbl, 0, 0, 300, 600);
```

```
  }

  private void applyBtn_actionPerformed(ActionEvent ae) {
    if (doNotDisplay.isSelected()) {
      myApplet.setNameANDValueVisible(false, false);
    }
    else if (valueOnly.isSelected()) {
      myApplet.setNameANDValueVisible(false, true);
    }
    else if (nameOnly.isSelected()) {
      myApplet.setNameANDValueVisible(true, false);
    }
    else if (nameANDValue.isSelected()) {
      myApplet.setNameANDValueVisible(true, true);
    }

    if (deg90.isSelected()) {
      myApplet.rotateComp(90);
    }
    else if (deg180.isSelected()) {
      myApplet.rotateComp(180);
    }
    else if (deg270.isSelected()) {
      myApplet.rotateComp(270);
    }
    else if (deg_90.isSelected()) {
      myApplet.rotateComp( -90);
    }
    else if (deg_180.isSelected()) {
      myApplet.rotateComp( -180);
    }
    else if (deg_270.isSelected()) {
      myApplet.rotateComp( -270);

    }
    myApplet.redrawGridCanvas();
    this.dispose();
  }

  private void cancelBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

DepSourcesInputFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

import ECS.*;
import ECS.CircuitComponents.*;

public class DepSourcesInputFrame
    extends CloseableFrame {

  JLabel lblLabel, lblGain, lblControl, lblGainUnits;
  JTextField txtLabel, txtGain, txtControl;
  JComboBox choiceGainUnits;
  JButton oky;

  Components curComp;
  mainApplet myApplet;
  public DepSourcesInputFrame() {
    Border border = BorderFactory.createEtchedBorder(Color.white, Color.blue);

    this.setSize(400, 270);
    lblLabel = new JLabel("Label:");
    lblLabel.setForeground(Color.blue);
    txtLabel = new JTextField(15);
    lblControl = new JLabel("Control Variable:");
    lblControl.setForeground(Color.BLUE);
    txtControl = new JTextField(15);
    lblGain = new JLabel("Gain(r):");
    lblGain.setForeground(Color.BLUE);
    txtGain = new JTextField(15);
    //lblUnits=new JLabel("Units:");
    //lblUnits.setForeground(Color.BLUE);
    //choiceUnits=new JComboBox();
    lblGainUnits = new JLabel("Gain Units:");
```

```
lblGainUnits.setForeground(Color.BLUE);
choiceGainUnits = new JComboBox();
choiceGainUnits.setBackground(Color.WHITE);
oky = new JButton("OK");
oky.setBackground(Color.WHITE);
oky.setBackground(Color.white);
oky.addMouseListener(new MouseAdapter() {
  public void mouseClicked(MouseEvent me) {
    oky_mouseClicked(me);
  }
}
);

oky.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
txtLabel.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
txtControl.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
txtGain.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
choiceGainUnits.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});

add(lblLabel, 10, 10, 90, 20);
add(txtLabel, 115, 10, 90, 20);
add(lblControl, 10, 50, 110, 20);
add(txtControl, 115, 50, 90, 20);
add(lblGain, 10, 100, 90, 20);
add(txtGain, 115, 100, 90, 20);
```

```
  //addToPanel(panel,lblUnits,190,50,90,20);
  //addToPanel(panel,choiceUnits,230,50,90,20);
  add(lblGainUnits, 210, 100, 90, 20);
  add(choiceGainUnits, 275, 100, 90, 20);
  add(oky, 150, 155, 75, 20);
  add(this.backgroundLbl, -100, -250, 500, 750);

}

public void setComponent(Components comp, mainApplet applet) {
  curComp = comp;

  myApplet = applet;
  this.setTitle(comp.getGenericName());
  txtLabel.setText(comp.getComponentName());
  Double gain = new Double(comp.getValue());
  txtGain.setText(gain.toString());
  String[] units = comp.getAllUnits();
  for (int i = 0; i < units.length; i++) {
    choiceGainUnits.addItem(units[i]);
  }
  txtControl.setText(comp.getControl());
  choiceGainUnits.setSelectedItem(comp.getUnit());
}

private void oky_mouseClicked(MouseEvent me) {
  this.setVisible(false);
  curComp.setComponentName(txtLabel.getText());
  Double gain = new Double(txtGain.getText());
  curComp.setValue(gain.doubleValue());
  String gainunit = new String( (String) choiceGainUnits.getSelectedItem());
  curComp.setUnit(gainunit);
  curComp.setControl(txtControl.getText());

  myApplet.redrawGridCanvas();
  myApplet.finalizeDepInput(curComp);

}

private void oky_keyPressed(KeyEvent ke) {
  if (ke.getKeyCode() == 10) {
    this.setVisible(false);
    curComp.setComponentName(txtLabel.getText());
    Double gain = new Double(txtGain.getText());
    curComp.setValue(gain.doubleValue());
```

```
      String gainunit = new String( (String) choiceGainUnits.getSelectedItem());
      curComp.setUnit(gainunit);
      curComp.setControl(txtControl.getText());

      myApplet.gridCanvas.gridImage.clearRect(0, 0, 650, 550);
      myApplet.gridCanvas.drawGridLayout(ActiveState.startPoint);
      myApplet.drawAllComps();
      myApplet.gridCanvas.redraw();
      myApplet.finalizeDepInput(curComp);

    }
  }

}
```

EditPageInfo.java

```
package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.mainApplet;

public class EditPageInfo
    extends CloseableFrame {

  JLabel titleLbl;
  JTextField titleTxt;
  JButton okBtn, cancelBtn;
  mainApplet myApplet;

  public EditPageInfo(mainApplet applet) {
    myApplet = applet;

    this.setSize(300, 150);
    this.setTitle("Page Title");
    titleLbl = new JLabel("Page Title");
    titleTxt = new JTextField();

    okBtn = new JButton("OK");
```

```
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });

    cancelBtn = new JButton("Cancel");
    cancelBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        cancelBtn_actionPerformed(ae);
      }
    });

    titleLbl.setBounds(30, 30, 70, 20);
    titleTxt.setBounds(100, 30, 150, 20);
    okBtn.setBounds(70, 70, 80, 20);
    okBtn.setBackground(Color.white);
    cancelBtn.setBounds(150, 70, 80, 20);
    cancelBtn.setBackground(Color.WHITE);

    titleTxt.setText(myApplet.pageTitle);
    titleTxt.setMaximumSize(new Dimension(100, 30));

    this.getContentPane().add(titleLbl);
    this.getContentPane().add(titleTxt);
    this.getContentPane().add(okBtn);
    this.getContentPane().add(cancelBtn);
    add(this.backgroundLbl, 0, 0, 300, 150);

  }

  private void okBtn_actionPerformed(ActionEvent ae) {

    myApplet.pageTitle = titleTxt.getText();
    myApplet.redrawGridCanvas();
    this.dispose();
  }

  private void cancelBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }
}
```

GoToPosition.java

```
package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.mainApplet;

public class GoToPosition
   extends CloseableFrame {

  JLabel titleLbl, xLbl, yLbl;
  JTextField xTxt, yTxt;
  JButton okBtn, cancelBtn;
  mainApplet myApplet;

  public GoToPosition(mainApplet applet) {
    myApplet = applet;

    this.setSize(300, 150);
    this.setTitle("Go To");
    titleLbl = new JLabel("Go To Position:");
    xLbl = new JLabel("X=");
    yLbl = new JLabel("Y=");
    xTxt = new JTextField();
    int x = myApplet.startPoint[0];
    xTxt.setText(new Integer(x).toString());
    yTxt = new JTextField();
    int y = myApplet.startPoint[1];
    yTxt.setText(new Integer(y).toString());

    okBtn = new JButton("OK");
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });

    cancelBtn = new JButton("Cancel");
    cancelBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        cancelBtn_actionPerformed(ae);
```

```
    }
   });

   titleLbl.setBounds(5, 30, 90, 20);

   xLbl.setBounds(100, 30, 20, 20);
   xTxt.setBounds(120, 30, 30, 20);
   yLbl.setBounds(160, 30, 20, 20);
   yTxt.setBounds(180, 30, 30, 20);
   okBtn.setBounds(70, 70, 80, 20);
   okBtn.setBackground(Color.white);
   cancelBtn.setBounds(150, 70, 80, 20);
   cancelBtn.setBackground(Color.WHITE);

   //tTxt.setText(myApplet.pageTitle);

   this.getContentPane().add(titleLbl);
   this.getContentPane().add(xTxt);
   this.getContentPane().add(yTxt);
   this.getContentPane().add(xLbl);
   this.getContentPane().add(yLbl);
   this.getContentPane().add(okBtn);
   this.getContentPane().add(cancelBtn);
   add(this.backgroundLbl, 0, 0, 300, 150);

  }

  private void okBtn_actionPerformed(ActionEvent ae) {
   Integer x = new Integer(xTxt.getText());
   Integer y = new Integer(yTxt.getText());

   myApplet.goToPosition(x.intValue(), y.intValue());
   this.dispose();
  }

  private void cancelBtn_actionPerformed(ActionEvent ae) {
   this.dispose();
  }
}
```

IndepACSourcesInputFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

import ECS.*;
import ECS.CircuitComponents.*;

public class IndepACSourcesInputFrame
    extends CloseableFrame {

  private JLabel lblLabel, lblPeak, lblUnits, lblFreqUnits, lblPhaseUnits,
      lblFrequency, lblPhase;
  private JTextField txtLabel, txtPeak, txtFrequency, txtPhase;
  private JComboBox choiceUnits, choiceFreqUnits, choicePhaseUnits;
  private ButtonGroup btnGroup;
  private JRadioButton radioSin, radioCos;
  private JButton oky;
  Components curComp;
  mainApplet myApplet;

  public IndepACSourcesInputFrame() {
    this.setSize(360, 270);
    Border border = BorderFactory.createEtchedBorder(Color.white, Color.blue);
    lblLabel = new JLabel("Label:");
    lblLabel.setForeground(Color.BLUE);
    txtLabel = new JTextField();
    lblPeak = new JLabel("Peak:");
    lblPeak.setForeground(Color.BLUE);
    txtPeak = new JTextField();
    lblUnits = new JLabel("Units:");
    lblUnits.setForeground(Color.BLUE);
    choiceUnits = new JComboBox();
    choiceUnits.setBackground(Color.WHITE);
    lblFrequency = new JLabel("Frequency");
```

```
lblFrequency.setForeground(Color.BLUE);
txtFrequency = new JTextField();
choiceFreqUnits = new JComboBox();
choiceFreqUnits.setBackground(Color.WHITE);
choiceFreqUnits.addItem("HZ");
choiceFreqUnits.addItem("KHZ");
choiceFreqUnits.addItem("MHZ");
choiceFreqUnits.addItem("rad/sec");
choiceFreqUnits.setSelectedItem("HZ");

lblPhase = new JLabel("Phase(O)");
lblPhase.setForeground(Color.BLUE);
txtPhase = new JTextField();

choicePhaseUnits = new JComboBox();
choicePhaseUnits.setBackground(Color.WHITE);
choicePhaseUnits.addItem("deg");
choicePhaseUnits.addItem("rad");
choicePhaseUnits.setSelectedItem("deg");
btnGroup = new ButtonGroup();
radioSin = new JRadioButton("sinwt");
radioSin.setBackground(new Color(169, 188, 254));
radioSin.setForeground(Color.BLUE);
radioCos = new JRadioButton("coswt");
radioCos.setBackground(new Color(169, 188, 254));
radioCos.setForeground(Color.BLUE);
btnGroup.add(radioSin);
btnGroup.add(radioCos);
radioSin.setSelected(true);

lblFreqUnits = new JLabel("Units");
lblFreqUnits.setForeground(Color.BLUE);
lblPhaseUnits = new JLabel("Units");
lblPhaseUnits.setForeground(Color.BLUE);
oky = new JButton("OK");
oky.setBackground(Color.WHITE);
oky.addMouseListener(new MouseAdapter() {
  public void mouseClicked(MouseEvent me) {
    oky_mouseClicked(me);
  }
}
);
oky.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
```

```
  }
});

txtLabel.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
txtPeak.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});

choiceUnits.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});

txtFrequency.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
txtPhase.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
choicePhaseUnits.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
choiceFreqUnits.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});

radioSin.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
```

```
  });

  radioCos.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });

  add(lblLabel, 10, 10, 70, 20);
  add(txtLabel, 80, 10, 70, 20);
  add(lblPeak, 10, 40, 70, 30);
  add(txtPeak, 80, 40, 70, 20);
  add(lblUnits, 180, 40, 70, 20);
  add(lblFreqUnits, 180, 70, 70, 20);
  add(lblPhaseUnits, 180, 100, 70, 20);
  add(choiceUnits, 260, 40, 70, 20);
  add(lblFrequency, 10, 70, 70, 20);
  add(txtFrequency, 80, 70, 70, 20);
  add(choiceFreqUnits, 260, 70, 70, 20);
  add(lblPhase, 10, 100, 70, 20);
  add(txtPhase, 80, 100, 70, 20);
  add(choicePhaseUnits, 260, 100, 70, 20);
  add(radioSin, 180, 10, 70, 20);
  add(radioCos, 260, 10, 70, 20);
  add(oky, 140, 160, 80, 20);
  add(this.backgroundLbl, -100, -250, 500, 750);

}

public void setComponent(Components comp, mainApplet applet) {
  curComp = comp;
  myApplet = applet;
  this.setTitle(comp.getGenericName());
  txtLabel.setText(comp.getName() + comp.getIndex());
  Double val = new Double(comp.getValue());
  txtPeak.setText(val.toString());
  Double freq = new Double(comp.getFrequency());
  txtFrequency.setText(freq.toString());
  Double phase = new Double(comp.getPhase());
  txtPhase.setText(phase.toString());
  String[] unit = comp.getAllUnits();
  if (comp.isCosine()) {
    radioCos.setSelected(true);
  }
  else {
```

```
    radioSin.setSelected(true);

   }
   for (int i = 0; i < unit.length; i++) {
    choiceUnits.addItem(unit[i]);

   }
   choiceUnits.setSelectedItem(comp.getUnit());
   //choiceFreqUnits.setSelectedItem(comp.getFreqUnit());
   //choicePhaseUnits.setSelectedItem(comp.getPhaseUnit());
  }

  private void oky_mouseClicked(MouseEvent me) {
   this.setVisible(false);
   String txtlbl = txtLabel.getText();
   Double txtval = new Double(txtPeak.getText());
   Double txtfreq = new Double(txtFrequency.getText());
   Double txtph = new Double(txtPhase.getText());
   String unit = new String( (String) choiceUnits.getSelectedItem());
   String phaseunit = new String( (String) choicePhaseUnits.getSelectedItem());
   String frequnit = new String( (String) choiceFreqUnits.getSelectedItem());

   curComp.setComponentName(txtlbl);
   curComp.setValue(txtval.doubleValue());
   curComp.setFrequency(txtfreq.doubleValue());
   if (radioCos.isSelected()) {
    curComp.setCosine(true);
   }
   else {
    curComp.setCosine(false);

   }
   curComp.setPhase(txtph.doubleValue());

   curComp.setUnit(unit);
   curComp.setFreqUnit(frequnit);
   curComp.setPhaseUnit(phaseunit);

   myApplet.redrawGridCanvas();
  }

  private void oky_keyPressed(KeyEvent ke) {
   if (ke.getKeyCode() == 10) {
    this.setVisible(false);
    String txtlbl = txtLabel.getText();
```

```
    Double txtval = new Double(txtPeak.getText());
    Double txtfreq = new Double(txtFrequency.getText());
    Double txtph = new Double(txtPhase.getText());
    String unit = new String( (String) choiceUnits.getSelectedItem());
    String phaseunit = new String( (String) choicePhaseUnits.getSelectedItem());
    String frequnit = new String( (String) choiceFreqUnits.getSelectedItem());

    curComp.setComponentName(txtlbl);
    curComp.setValue(txtval.doubleValue());
    curComp.setFrequency(txtfreq.doubleValue());
    if (radioCos.isSelected()) {
      curComp.setPhase(txtph.doubleValue() + 45);
    }
    else {
      curComp.setPhase(txtph.doubleValue());
    }
    curComp.setUnit(unit);
    curComp.setFreqUnit(frequnit);
    curComp.setPhaseUnit(phaseunit);

    myApplet.gridCanvas.gridImage.clearRect(0, 0, 650, 550);
    myApplet.gridCanvas.drawGridLayout(ActiveState.startPoint);
    myApplet.drawAllComps();
    myApplet.gridCanvas.redraw();

   }
  }

}
```

IndepDCSourcesInputFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

import ECS.*;
import ECS.CircuitComponents.*;
```

```
public class IndepDCSourcesInputFrame
    extends CloseableFrame {

  JLabel lblLabel, lblValue, lblUnits;
  JTextField txtLabel, txtValue;
  JComboBox choiceUnits;
  JButton oky;
  Components curComp;
  mainApplet myApplet;

  public IndepDCSourcesInputFrame() {
    Border border = BorderFactory.createEtchedBorder(Color.white, Color.blue);

    this.setSize(320, 210);
    lblLabel = new JLabel("Label:");
    lblLabel.setBackground(SystemColor.control);
    lblLabel.setForeground(Color.blue);

    lblValue = new JLabel("Value");
    lblValue.setBackground(SystemColor.control);
    lblValue.setForeground(Color.blue);

    lblUnits = new JLabel("Units:");
    lblUnits.setBackground(SystemColor.control);
    lblUnits.setForeground(Color.blue);
    lblUnits.setPreferredSize(new Dimension(18, 15));

    txtLabel = new JTextField();

    txtValue = new JTextField();

    choiceUnits = new JComboBox();
    choiceUnits.setBackground(Color.WHITE);
    oky = new JButton("OK");
    oky.setBackground(Color.white);
    oky.setForeground(Color.blue);
    oky.addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent me) {
        oky_mouseClicked(me);
      }
    }
    );
    txtLabel.addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent ke) {
```

```
      oky_keyPressed(ke);
    }
  });
  txtValue.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });
  choiceUnits.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });

  add(lblLabel, 10, 20, 34, 13);
  add(lblValue, 10, 60, 34, 15);
  add(lblUnits, 150, 60, 34, 14);
  add(txtLabel, 60, 20, 86, 21);
  add(txtValue, 60, 60, 86, 21);
  add(choiceUnits, 190, 60, 82, 21);
  add(oky, 115, 110, 73, 25);
  add(this.backgroundLbl, -100, -250, 500, 650);
}

public void setComponent(Components comp, mainApplet applet) {
  curComp = comp;
  myApplet = applet;
  this.setTitle(comp.getGenericName());
  txtLabel.setText(comp.getComponentName());
  Double value = new Double(comp.getValue());
  txtValue.setText(value.toString());
  String[] units = comp.getAllInputUnits();
  for (int i = 0; i < units.length; i++) {
    choiceUnits.addItem(units[i]);
  }
  choiceUnits.setSelectedItem(comp.getUnit());
}

private void oky_mouseClicked(MouseEvent me) {
  this.setVisible(false);
  String txtlbl = txtLabel.getText();
  Double txtval = new Double(txtValue.getText());
  String unit = new String( (String) choiceUnits.getSelectedItem());
  curComp.setComponentName(txtlbl);
  curComp.setValue(txtval.doubleValue());
```

```
    curComp.setUnit(unit);
    myApplet.redrawGridCanvas();
  }

  private void oky_keyPressed(KeyEvent ke) {
    if (ke.getKeyCode() == 10) {
      this.setVisible(false);
      String txtlbl = txtLabel.getText();
      Double txtval = new Double(txtValue.getText());
      String unit = new String( (String) choiceUnits.getSelectedItem());
      curComp.setComponentName(txtlbl);
      curComp.setValue(txtval.doubleValue());
      curComp.setUnit(unit);
      myApplet.gridCanvas.gridImage.clearRect(0, 0, 650, 550);
      myApplet.gridCanvas.drawGridLayout(ActiveState.startPoint);
      myApplet.drawAllComps();
      myApplet.gridCanvas.redraw();
    }
  }

}
```

InsertTextFrame.java

```
 package ECS.GUIFrameWindows;

import java.lang.reflect.*;
import java.util.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;
import ECS.DrawingTools.*;
import ECS.GUPanels.*;

public class InsertTextFrame
    extends CloseableFrame {

  JScrollPane txtScroll;
```

```
public JTextArea txtArea;
JButton okBtn, cancelBtn;
GridCanvas gridCanvas;
mainApplet myApplet;
Vector lines = new Vector();
String line = "";
String text;
int numb;

public InsertTextFrame(mainApplet applet, int m) {
  this.getContentPane().setBackground(new Color(91, 141, 255));
  this.setTitle("Insert text");
  this.setSize(310, 200);
  myApplet = applet;
  numb = m;

  txtScroll = new JScrollPane();
  txtScroll.setBounds(10, 10, 200, 100);

  txtArea = new JTextArea();
  txtArea.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      textArea_keyPressed_action(ke);
    }
  }
  );
  okBtn = new JButton("ok");
  okBtn.setBounds(215, 20, 85, 25);
  okBtn.setBackground(Color.WHITE);
  okBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      ok_actionPerformed(ae);
    }
  }
  );

  cancelBtn = new JButton("Cancel");
  cancelBtn.setBackground(Color.WHITE);
  cancelBtn.setBounds(215, 60, 85, 25);
  cancelBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      cancel_actionPerformed(ae);
    }
  }
  );
```

```
  txtScroll.getViewport().add(txtArea);
  this.getContentPane().add(txtScroll);
  this.getContentPane().add(okBtn);
  this.getContentPane().add(cancelBtn);
  this.backgroundLbl.setBounds(0, 0, 310, 200);
  this.getContentPane().add(this.backgroundLbl);
}

public void setText(String str) {
  txtArea.setText(str);
}

public String getText() {
  return txtArea.getText();
}

private void ok_actionPerformed(ActionEvent ae) {
  text = txtArea.getText();

  for (int i = 0; i < text.length(); i++) {
    Character a = new Character(text.charAt(i));

    String cr = "\n";
    String sp = " ";

    if (a.compareTo(new Character(cr.charAt(0))) != 0) {
      line += text.charAt(i);

      if (i == text.length() - 1) {
        lines.add(line);
        line = "";
      }
    }
    else {
      lines.add(line);
      line = "";
    }
  }

  if (numb != -1) {
    ( (TextBox) mainApplet.drawComps[numb]).lines = lines;
    mainApplet.drawComps[numb].setText(text);

  }
```

```
    else {
      int size = Array.getLength(mainApplet.drawComps);
      mainApplet.drawComps[size - 1] = new TextBox(lines, mainApplet.clickedPos);
      mainApplet.drawComps[size - 1].setText(txtArea.getText());
      mainApplet.drawComps[size - 1].setTextColor(mainApplet.drawingTextColor);
      mainApplet.drawComps[size - 1].drawComponent(gridCanvas);
      mainApplet.drawComps = (DrawTool[]) mainApplet.growArray(mainApplet.
          drawComps);
    }
    myApplet.redrawGridCanvas();
    this.dispose();

  }

  private void cancel_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

  private void textArea_keyPressed_action(KeyEvent ke) {

  }

}
```

MakeDirectoryFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MakeDirectoryFrame
    extends JDialog {

  JTextField txtDir;
  JButton oky;

  public MakeDirectoryFrame() {

    setSize(new Dimension(300, 150));
```

```
    setTitle("Enter the name of the Directory");
    setLocation(300, 300);
    this.getContentPane().setLayout(null);
    txtDir = new JTextField(10);
    oky = new JButton("OK");
    txtDir.setBounds(80, 30, 130, 25);
    oky.setBounds(105, 70, 80, 20);
    this.getContentPane().add(txtDir);
    this.getContentPane().add(oky);

    oky.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        ok_action_performed(ae);
      }
    });

  }

  private void ok_action_performed(ActionEvent ae) {

    String DirName = txtDir.getText();
    RemoteFileManagerFrame.DirName = DirName;
    this.dispose();

    RemoteFileManagerFrame.createDirectory();
  }
}
```

NetlistEditorFrame.java

```
 package ECS.GUIFrameWindows;

import java.util.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;
import ECS.GUPanels.*;
```

```
public class NetlistEditorFrame
    extends CloseableFrame {

  JScrollPane txtScroll;
  JTextArea txtArea;
  JButton okBtn, cancelBtn;
  GridCanvas gridCanvas;
  mainApplet myApplet;
  Vector lines = new Vector();
  String line = "";

  public NetlistEditorFrame(mainApplet applet) {
    this.getContentPane().setBackground(new Color(91, 141, 255));
    this.setTitle("Netlist Editor");
    this.setSize(370, 300);

    myApplet = applet;

    txtScroll = new JScrollPane();
    txtScroll.setBounds(5, 10, 350, 200);

    txtArea = new JTextArea();
    txtArea.setText(myApplet.cirMLString);
    txtArea.addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent ke) {
        textArea_keyPressed_action(ke);
      }
    }
    );
    okBtn = new JButton("ok");
    okBtn.setBounds(120, 220, 70, 25);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        ok_actionPerformed(ae);
      }
    }
    );

    cancelBtn = new JButton("Cancel");
    cancelBtn.setBounds(190, 220, 70, 25);
    cancelBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        cancel_actionPerformed(ae);
      }
    }
```

```
    );

    txtScroll.getViewport().add(txtArea);
    this.getContentPane().add(txtScroll);
    this.getContentPane().add(okBtn);
    this.getContentPane().add(cancelBtn);
    this.backgroundLbl.setBounds(0, 0, 370, 300);
    this.getContentPane().add(this.backgroundLbl);
  }

  public void setText(String str) {
    txtArea.setText(str);
  }

  public String getText() {
    return txtArea.getText();
  }

  private void ok_actionPerformed(ActionEvent ae) {
    myApplet.cirMLString = txtArea.getText();
    ActiveState.textEntry = true;
    this.dispose();
  }

  private void cancel_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

  private void textArea_keyPressed_action(KeyEvent ke) {

  }

}
```

OutputFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
import ECS.*;

public class OutputFrame
    extends CloseableFrame {

  JScrollPane txtScroll;
  JTextArea txtArea;
  JButton okBtn;
  JLabel background;

  public OutputFrame(String title, String outputString) {
    this.getContentPane().setBackground(new Color(91, 141, 255));
    this.setTitle(title);
    this.setSize(370, 300);

    txtScroll = new JScrollPane();
    txtScroll.setBounds(7, 10, 350, 200);

    txtArea = new JTextArea();
    txtArea.setEditable(false);
    txtArea.setText(outputString);
    okBtn = new JButton("ok");
    okBtn.setBounds(130, 220, 60, 20);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        ok_actionPerformed(ae);
      }
    }
    );

    txtScroll.getViewport().add(txtArea);
    this.getContentPane().add(txtScroll);
    this.getContentPane().add(okBtn);
    this.backgroundLbl.setBounds(0, 0, 370, 300);
    this.getContentPane().add(this.backgroundLbl);

  }

  private void ok_actionPerformed(ActionEvent ae) {

    this.dispose();
  }

}
```

PassiveElementsInputFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

import ECS.*;
import ECS.CircuitComponents.*;

public class PassiveElementsInputFrame
    extends CloseableFrame {

  JLabel lblLabel, lblValue, lblUnits;
  JTextField txtLabel, txtValue;
  JComboBox choiceUnits;
  JButton oky;
  Components curComp;
  mainApplet myApplet;
  public PassiveElementsInputFrame() {
    this.getContentPane().setBackground(new Color(91, 141, 255));

    Border border = BorderFactory.createEtchedBorder(Color.white, Color.blue);

    this.setSize(320, 210);
    lblLabel = new JLabel("Label:");
    lblLabel.setBackground(SystemColor.control);
    lblLabel.setForeground(Color.blue);

    lblValue = new JLabel("Value");
    lblValue.setBackground(SystemColor.control);
    lblValue.setForeground(Color.blue);

    lblUnits = new JLabel("Units:");
    lblUnits.setBackground(SystemColor.control);
    lblUnits.setForeground(Color.blue);
    lblUnits.setPreferredSize(new Dimension(18, 15));
```

```
  txtLabel = new JTextField();

  txtValue = new JTextField();

  choiceUnits = new JComboBox();
  choiceUnits.setBackground(Color.WHITE);
  oky = new JButton("OK");
  oky.setBackground(Color.white);
  oky.setForeground(Color.blue);
  oky.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent me) {
      oky_mouseClicked(me);
    }
  }
  );
  oky.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });
  txtLabel.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });
  txtValue.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });
  choiceUnits.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent ke) {
      oky_keyPressed(ke);
    }
  });

  add(lblLabel, 10, 20, 34, 13);
  add(lblValue, 10, 60, 34, 15);
  add(lblUnits, 150, 60, 34, 14);
  add(txtLabel, 60, 20, 86, 21);
  add(txtValue, 60, 60, 86, 21);
  add(choiceUnits, 190, 60, 100, 21);
  add(oky, 115, 110, 73, 25);
  add(this.backgroundLbl, -100, -250, 500, 650);
}
```

```
  public void setComponent(Components comp, mainApplet applet) {
    curComp = comp;
    myApplet = applet;
    this.setTitle(comp.getGenericName());
    txtLabel.setText(comp.getComponentName());
    Double value = new Double(comp.getValue());
    txtValue.setText(value.toString());
    String[] units = comp.getAllInputUnits();
    for (int i = 0; i < units.length; i++) {
      choiceUnits.addItem(units[i]);
    }
    choiceUnits.setSelectedItem(comp.getUnit());
  }

  private void oky_mouseClicked(MouseEvent me) {
    this.setVisible(false);
    String txtlbl = txtLabel.getText();
    Double txtval = new Double(txtValue.getText());
    curComp.setComponentName(txtlbl);
    curComp.setValue(txtval.doubleValue());
    String[] units = curComp.getAllUnits();
    int unitIndex = (int) choiceUnits.getSelectedIndex();
    curComp.setUnit(units[unitIndex]);
    myApplet.redrawGridCanvas();
  }

  private void oky_keyPressed(KeyEvent ke) {

    if (ke.getKeyCode() == 10) {
      this.setVisible(false);
      String txtlbl = txtLabel.getText();
      Double txtval = new Double(txtValue.getText());
      curComp.setComponentName(txtlbl);
      curComp.setValue(txtval.doubleValue());
      String unit = new String( (String) choiceUnits.getSelectedItem());
      curComp.setUnit(unit);
      myApplet.gridCanvas.gridImage.clearRect(0, 0, 650, 550);
      myApplet.gridCanvas.drawGridLayout(ActiveState.startPoint);
      myApplet.drawAllComps();
      myApplet.gridCanvas.redraw();
    }
  }
}
```

RemoteFileManagerFrame.java

```
 package ECS.GUIFrameWindows;

import java.io.*;
import java.net.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;

import ECS.*;

public class RemoteFileManagerFrame
    extends JFrame {

//Variable declaration//
  static Socket commandClient, dataClient;
//JPanel mainPanel;
  static JScrollPane scrollPanel;
  static ObjectInputStream OIS;
  static ObjectOutputStream OOS;
  JButton connect, upload, download, disconnect, mkDir, deltFile, exit
      , loadSchematic, saveSchematic, renameFile;
  File[] filesList;
  static JTree tree;
  static DefaultMutableTreeNode root;
  JFrame frameChooser;
  JFileChooser fc;
  File uploadedFile;
  DataOutputStream DOS;
  DataInputStream DIS;
  InputStream FIS;
  OutputStream FOS;
  static String downloadedFilename = "", uploadedFolder = "",
      renamedFilename = "", newFileName = "";
  static String DirName = "", selectedFolder = "";
  boolean open = false, save = false;

  public final MouseListener mouseListener = new java.awt.event.MouseAdapter() {
    public void mouseEntered(MouseEvent me) {
```

```
    mouse_entered_action(me);
  }

  public void mouseExited(MouseEvent me) {
    mouse_exited_action(me);
  }

};
Font myFont = new Font("Serif", Font.PLAIN, 13);

public RemoteFileManagerFrame() {

  this.setSize(370, 430);
  this.setLocation(200, 150);
  this.setTitle("Remote Server File Manager");
  this.getContentPane().setBackground(new Color(117, 159, 255));

  this.getContentPane().setLayout(null);
  scrollPanel = new JScrollPane();

  scrollPanel.setBounds(20, 10, 230, 350);
  this.getContentPane().add(scrollPanel, null);

  Image img = ActiveState.diffImages[ActiveState.SIMULATIONBACKGROUND];
  JLabel background = new JLabel(new ImageIcon(img));
  background.setBounds(0, 0, 550, 400);

  img = ActiveState.btnsExited[ActiveState.CONNECTFM];
  connect = new JButton(new ImageIcon(img));
  connect.setFont(myFont);
  connect.setBounds(255, 53, 100, 30);
  connect.setActionCommand("connect");
  connect.setToolTipText("Connect to Server");
  connect.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      connect_action_performed(ae);
    }
  });
  connect.setBackground(new Color(179, 186, 241));
  connect.setBorder(BorderFactory.createLineBorder(Color.WHITE));
  connect.setForeground(Color.WHITE);
  connect.addMouseListener(mouseListener);

  img = ActiveState.btnsExited[ActiveState.DISCONNECTFM];
  disconnect = new JButton(new ImageIcon(img));
```

```
disconnect.setBounds(255, 88, 100, 30);
disconnect.setFont(myFont);
disconnect.setActionCommand("disconnect");
disconnect.setToolTipText("Disconnect from Server");
disconnect.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    disconnect_action_performed(ae);
  }
}
);
disconnect.setBackground(new Color(179, 186, 241));
disconnect.setBorder(BorderFactory.createLineBorder(Color.WHITE));
disconnect.setForeground(Color.WHITE);
disconnect.addMouseListener(mouseListener);

img = ActiveState.btnsExited[ActiveState.LOADSCHEMATICFM];
loadSchematic = new JButton(new ImageIcon(img));
loadSchematic.setBounds(255, 123, 100, 30);
loadSchematic.setFont(myFont);
loadSchematic.setActionCommand("loadSchematic");
loadSchematic.setToolTipText("load Schematic Page");
loadSchematic.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    loadSchematic_action_performed(ae);
  }
}
);
loadSchematic.setBackground(new Color(179, 186, 241));
loadSchematic.setBorder(BorderFactory.createLineBorder(Color.WHITE));
loadSchematic.setForeground(Color.WHITE);
loadSchematic.addMouseListener(mouseListener);

img = ActiveState.btnsExited[ActiveState.SAVESCHEMATICFM];
saveSchematic = new JButton(new ImageIcon(img));
saveSchematic.setBounds(255, 158, 100, 30);
saveSchematic.setFont(myFont);
saveSchematic.setActionCommand("saveSchematic");
saveSchematic.setToolTipText("Save Schematic");
saveSchematic.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    saveSchematic_action_performed(ae);
  }
}
);
saveSchematic.setBackground(new Color(179, 196, 241));
```

```
saveSchematic.setBorder(BorderFactory.createLineBorder(Color.WHITE));
saveSchematic.setForeground(Color.WHITE);
saveSchematic.addMouseListener(mouseListener);

img = ActiveState.btnsExited[ActiveState.DOWNLOADFM];
download = new JButton(new ImageIcon(img));
download.setBounds(255, 158, 100, 30);
download.setFont(myFont);
download.setActionCommand("download");
download.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    download_action_performed(ae);
  }
}
);
download.setBackground(new Color(179, 186, 241));
download.setBorder(BorderFactory.createLineBorder(Color.WHITE));
download.setForeground(Color.WHITE);
download.addMouseListener(mouseListener);

img = ActiveState.btnsExited[ActiveState.UPLOADFM];
upload = new JButton(new ImageIcon(img));
upload.setBounds(255, 123, 100, 30);
upload.setFont(myFont);
upload.setActionCommand("upload");
upload.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    upload_action_performed(ae);
  }
});
upload.setBackground(new Color(179, 186, 241));
upload.setBorder(BorderFactory.createLineBorder(Color.WHITE));
upload.setForeground(Color.WHITE);
upload.addMouseListener(mouseListener);

img = ActiveState.btnsExited[ActiveState.MKDIRFM];
mkDir = new JButton(new ImageIcon(img));
mkDir.setBounds(255, 193, 100, 30);
mkDir.setFont(myFont);
mkDir.setActionCommand("mkDirectory");
mkDir.setToolTipText("Make Directory");
mkDir.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    mkDir_action_performed(ae);
  }
```

```
}
);
mkDir.setBackground(new Color(179, 176, 241));
mkDir.setBorder(BorderFactory.createLineBorder(Color.WHITE));
mkDir.setForeground(Color.WHITE);
mkDir.addMouseListener(mouseListener);

img = ActiveState.btnsExited[ActiveState.FILERENAMEFM];
renameFile = new JButton(new ImageIcon(img));
renameFile.setBounds(255, 228, 100, 30);
renameFile.setFont(myFont);
renameFile.setActionCommand("rename");
renameFile.setToolTipText("Rename");
renameFile.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    rename_action_performed(ae);
  }
}
);
renameFile.setBackground(new Color(179, 186, 241));
renameFile.setBorder(BorderFactory.createLineBorder(Color.WHITE));
renameFile.setForeground(Color.WHITE);
renameFile.addMouseListener(mouseListener);

img = ActiveState.btnsExited[ActiveState.FILEDELETEFM];
deltFile = new JButton(new ImageIcon(img));
deltFile.setBounds(255, 263, 100, 30);
deltFile.setActionCommand("delete");
deltFile.setFont(myFont);
deltFile.setToolTipText("Delete");
deltFile.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    deltFile_action_performed(ae);
  }
}
);
deltFile.setBackground(new Color(179, 186, 241));
deltFile.setBorder(BorderFactory.createLineBorder(Color.WHITE));
deltFile.setForeground(Color.WHITE);
deltFile.addMouseListener(mouseListener);

img = ActiveState.btnsExited[ActiveState.EXITFM];
exit = new JButton(new ImageIcon(img));
exit.setBounds(255, 298, 100, 30);
exit.setFont(myFont);
```

```
    exit.setActionCommand("exit");
    exit.setToolTipText("Exit File Manager");
    exit.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        exit_action_performed(ae);
      }
    }
    );
    exit.setBackground(new Color(179, 186, 241));
    exit.setBorder(BorderFactory.createLineBorder(Color.WHITE));
    exit.setForeground(Color.WHITE);
    exit.addMouseListener(mouseListener);

    this.getContentPane().add(scrollPanel);
    this.getContentPane().add(connect);
    this.getContentPane().add(upload);
    this.getContentPane().add(download);
    this.getContentPane().add(disconnect);
    //this.getContentPane().add(saveSchematic);
    //this.getContentPane().add(loadSchematic);
    this.getContentPane().add(mkDir);
    this.getContentPane().add(renameFile);
    this.getContentPane().add(deltFile);
    this.getContentPane().add(exit);

//root=new DefaultMutableTreeNode();
    this.getContentPane().add(background);

    tree = new JTree(new DefaultMutableTreeNode("/"));
    scrollPanel.getViewport().add(tree);

  }

////////////////////////////////////////////////////////////////////////////
  private void connect_action_performed(ActionEvent ae) {

    connectToServer();

    try {
      OOS = new ObjectOutputStream(commandClient.getOutputStream());
      OOS.writeObject(new String("list"));

      OIS = new ObjectInputStream(commandClient.getInputStream());
      try {
        root = (DefaultMutableTreeNode) OIS.readObject();
```

```
      }
      catch (ClassNotFoundException cnfe) {
        System.out.println(cnfe);
      }
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    tree = new JTree(root);
    tree.setBounds(0, 0, 250, 400);
    tree.addTreeSelectionListener(new TreeSelectionListener() {
      public void valueChanged(TreeSelectionEvent event) {
        tree_value_changed(event);
      }
    });
    scrollPanel.getViewport().add(tree, null);

    //mainPanel.add(tree);

    /*for(int i=0;i<filesList.length;i++)
     {
     System.out.println(filesList[i].getName());
     }
     addNode(root,filesList);
     */

//tree.putClientProperty("JTree.lineStyle", "Horizontal");
    tree.expandRow(0);

    tree.repaint();

  }

////////////////////////////////////////////////////////////////////////////
  private void upload_action_performed(ActionEvent ae) {
    open = true;
    fc = new JFileChooser(new File("/"));
    fc.setSize(300, 400);
    fc.setEnabled(true);
    fc.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        fc_action_performed(ae);
      }
```

```
    });

    fc.showOpenDialog(this.getContentPane());

  }

////////////////////////////////////////////////////////////////////////////
  private void download_action_performed(ActionEvent ae) {
    if (downloadedFilename != "") {
      save = true;
      fc = new JFileChooser(new File("/"));
      fc.setSize(300, 400);
      fc.setEnabled(true);
      fc.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
          fc_action_performed(ae);
        }

      });

      fc.showSaveDialog(this.getContentPane());
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void mkDir_action_performed(ActionEvent ae) {
    MakeDirectoryFrame mkDir = new MakeDirectoryFrame();
    mkDir.setLocation(350, 300);
    mkDir.show();
  }

////////////////////////////////////////////////////////////////////////////
  private void rmDir_action_performed(ActionEvent ae) {
    /*  String path="";
      if(uploadedFolder.equals(null))
      {path=DirName;}
      else
      {path=uploadedFolder+"/"+DirName;}
       System.out.println(path);
     */

    try {
      OOS = new ObjectOutputStream(commandClient.getOutputStream());
      OOS.writeObject(new String("rmDir"));
```

```
      String path = uploadedFolder;
      OOS.writeObject(path);

    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    try {
      OIS = new ObjectInputStream(commandClient.getInputStream());
      try {
        root = (DefaultMutableTreeNode) OIS.readObject();

      }
      catch (ClassNotFoundException cnfe) {
        System.out.println(cnfe);
      }
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    scrollPanel.getViewport().remove(tree);
    scrollPanel.getViewport().repaint();
    tree = new JTree(root);
    tree.addTreeSelectionListener(new TreeSelectionListener() {
      public void valueChanged(TreeSelectionEvent event) {
        tree_value_changed(event);
      }
    });
    scrollPanel.getViewport().add(tree);

  }

////////////////////////////////////////////////////////////////////////////
  private void deltFile_action_performed(ActionEvent ae) {

    if (downloadedFilename != "") {
      try {
        OOS = new ObjectOutputStream(commandClient.getOutputStream());
        OOS.writeObject(new String("delete"));

        OOS.writeObject(new String(downloadedFilename));

      }
```

```
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    try {
      OIS = new ObjectInputStream(commandClient.getInputStream());
      try {
        root = (DefaultMutableTreeNode) OIS.readObject();

      }
      catch (ClassNotFoundException cnfe) {
        System.out.println(cnfe);
      }
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    scrollPanel.getViewport().remove(tree);
    scrollPanel.getViewport().repaint();
    tree = new JTree(root);
    tree.addTreeSelectionListener(new TreeSelectionListener() {
      public void valueChanged(TreeSelectionEvent event) {
        tree_value_changed(event);
      }
    });
    scrollPanel.getViewport().add(tree);
  }
 }

////////////////////////////////////////////////////////////////////////////
 private void rename_action_performed(ActionEvent ae) {
  if (downloadedFilename != "") {
    RenameFileFrame renameFile = new RenameFileFrame();
    renameFile.show();
  }
 }

////////////////////////////////////////////////////////////////////////////
 public static void rename() {
  if (downloadedFilename != "") {
    try {
      OOS = new ObjectOutputStream(commandClient.getOutputStream());
      OOS.writeObject(new String("rename"));
```

```
      OOS.writeObject(new String(downloadedFilename));

      OOS.writeObject(new String(selectedFolder + newFileName));

    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    try {
      OIS = new ObjectInputStream(commandClient.getInputStream());
      try {
        root = (DefaultMutableTreeNode) OIS.readObject();

      }
      catch (ClassNotFoundException cnfe) {
        System.out.println(cnfe);
      }
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    scrollPanel.getViewport().remove(tree);
    scrollPanel.getViewport().repaint();
    tree = new JTree(root);
    tree.addTreeSelectionListener(new TreeSelectionListener() {
      public void valueChanged(TreeSelectionEvent event) {
        tree_value_changed(event);
      }
    });
    scrollPanel.getViewport().add(tree);
  }

 }

////////////////////////////////////////////////////////////////////////////
  private void disconnect_action_performed(ActionEvent ae) {
    try {
      OOS = new ObjectOutputStream(commandClient.getOutputStream());
      OOS.writeObject(new String("disconnect"));
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }
```

```
    tree = new JTree();

    scrollPanel.getViewport().remove(tree);
    scrollPanel.getViewport().repaint();
    tree = new JTree(new DefaultMutableTreeNode("/"));
    scrollPanel.getViewport().add(tree);

  }

////////////////////////////////////////////////////////////////////////////
  private void loadSchematic_action_performed(ActionEvent ae) {

  }

////////////////////////////////////////////////////////////////////////////
  private void saveSchematic_action_performed(ActionEvent ae) {
    open = true;
    fc = new JFileChooser(new File("/"));
    fc.setSize(300, 400);
    fc.setEnabled(true);
    fc.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        fc_action_performed(ae);
      }

    });

    fc.showOpenDialog(this.getContentPane());

  }

////////////////////////////////////////////////////////////////////////////
  private void connectToServer() {
    try {
      commandClient = new Socket("localhost", 3500);
      dataClient = new Socket("localhost", 3600);

    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }
  }

////////////////////////////////////////////////////////////////////////////
  private void fc_action_performed(ActionEvent ae) {
```

```
disconnect();
connectToServer();

if (open) {
  try {
    OOS = new ObjectOutputStream(commandClient.getOutputStream());
    OOS.writeObject(new String("upload"));
  }
  catch (IOException ioe) {
    System.out.println(ioe);
  }

  uploadedFile = fc.getSelectedFile();

  try {
    OOS = new ObjectOutputStream(dataClient.getOutputStream());
    String path;
    if (uploadedFolder == "") {
      path = uploadedFile.getName();
    }
    else {
      path = uploadedFolder + "/" + uploadedFile.getName();

    }
    OOS.writeObject(path);
  }
  catch (IOException ioe) {
    System.out.println(ioe);
  }

  try {
    DOS = new DataOutputStream(dataClient.getOutputStream());
    FIS = new FileInputStream(uploadedFile.getPath());

    int c = FIS.read();
    while (c != -1) {

      DOS.write(c);
      c = FIS.read();
    }
    DOS.close();
    FIS.close();
  }
  catch (IOException ioe) {
    System.out.println(ioe);
```

```
  }
  open = false;

  try {

    OIS = new ObjectInputStream(commandClient.getInputStream());
    try {
      root = (DefaultMutableTreeNode) OIS.readObject();

    }
    catch (ClassNotFoundException cnfe) {
      System.out.println(cnfe);
    }
  }
  catch (IOException ioe) {
    System.out.println(ioe);
  }

  scrollPanel.getViewport().remove(tree);
  scrollPanel.getViewport().repaint();
  tree = new JTree(root);
  tree.addTreeSelectionListener(new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent event) {
      tree_value_changed(event);
    }
  });

  scrollPanel.getViewport().add(tree);

}

if (save) {
  if (downloadedFilename != "") {
    try {
      OOS = new ObjectOutputStream(commandClient.getOutputStream());
      OOS.writeObject(new String("download"));
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    try {
      OOS = new ObjectOutputStream(dataClient.getOutputStream());
      OOS.writeObject(new String(downloadedFilename));
```

```
        DIS = new DataInputStream(dataClient.getInputStream());
        String filename = fc.getSelectedFile().getPath();
        FOS = new FileOutputStream(filename);

        int c = DIS.read();
        while (c != -1) {
          FOS.write(c);
          c = DIS.read();
        }

        FOS.close();
        DIS.close();
        OOS.close();
      }
      catch (IOException ioe) {
        System.out.println();
      }
      save = false;

    }
  }

}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
 private static void tree_value_changed(TreeSelectionEvent event) {
  int c = tree.getSelectionPath().getPathCount();

  downloadedFilename = "";
  uploadedFolder = "";
  selectedFolder = "";
  for (int i = 1; i < c; i++) {
    if (i == c - 1) {
      downloadedFilename += tree.getSelectionPath().getPathComponent(i).
          toString();
      uploadedFolder += tree.getSelectionPath().getPathComponent(i).toString();
    }
    else {
      downloadedFilename += tree.getSelectionPath().getPathComponent(i).
          toString() + "/";
      uploadedFolder += tree.getSelectionPath().getPathComponent(i).toString() +
          "/";
    }
  }
```

```
    for (int i = 1; i < c - 1; i++) {
      if (i == c - 1) {
        selectedFolder += tree.getSelectionPath().getPathComponent(i).
            toString();

      }
      else {
        selectedFolder += tree.getSelectionPath().getPathComponent(i).
            toString() + "/";

      }
    }

  }

//////////////////////////////////////////////////////////////////////////////

  private void addNode(DefaultMutableTreeNode parent, File[] filesList) {
    for (int i = 0; i < filesList.length; i++) {

      if (filesList[i].isDirectory()) {
        DefaultMutableTreeNode subDir = new DefaultMutableTreeNode(filesList[i].
            getName());

        File[] subDirList = filesList[i].listFiles();
        addNode(subDir, subDirList);
        parent.add(subDir);
      }
      else {
        DefaultMutableTreeNode node = new DefaultMutableTreeNode(filesList[i].
            getName());
        parent.add(node);
      }
    }

  }

//////////////////////////////////////////////////////////////////////////////

  private void disconnect() {
    try {
      OOS = new ObjectOutputStream(commandClient.getOutputStream());
      OOS.writeObject(new String("disconnect"));
    }
```

```
    catch (IOException ioe) {
      System.out.println(ioe);
    }

  }

////////////////////////////////////////////////////////////////////////////
  public static void createDirectory() {
    /*  String path="";
      if(uploadedFolder.equals(null))
      {path=DirName;}
      else
      {path=uploadedFolder+"/"+DirName;}
       System.out.println(path);
     */

    try {
      OOS = new ObjectOutputStream(commandClient.getOutputStream());
      OOS.writeObject(new String("mkDir"));
      String path;
      if (uploadedFolder.equals("")) {
        path = DirName;
      }
      else {
        path = uploadedFolder + "/" + DirName;

      }
      OOS.writeObject(path);

    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    try {
      OIS = new ObjectInputStream(commandClient.getInputStream());
      try {
        root = (DefaultMutableTreeNode) OIS.readObject();

      }
      catch (ClassNotFoundException cnfe) {
        System.out.println(cnfe);
      }
    }
    catch (IOException ioe) {
```

```
      System.out.println(ioe);
    }

    scrollPanel.getViewport().remove(tree);
    scrollPanel.getViewport().repaint();
    tree = new JTree(root);
    tree.addTreeSelectionListener(new TreeSelectionListener() {
      public void valueChanged(TreeSelectionEvent event) {
        tree_value_changed(event);
      }
    });
    scrollPanel.getViewport().add(tree);
  }

////////////////////////////////////////////////////////////////////////////
  private void mouse_entered_action(MouseEvent me) {
    JButton btn = (JButton) me.getSource();
    String cmd = btn.getActionCommand();
    if (cmd.equals("connect")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.CONNECTFM]));
    }
    else if (cmd.equals("disconnect")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                        DISCONNECTFM]));
    }
    else if (cmd.equals("download")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.DOWNLOADFM]));
    }
    else if (cmd.equals("upload")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.UPLOADFM]));
    }
    else if (cmd.equals("mkDirectory")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.MKDIRFM]));
    }
    else if (cmd.equals("rename")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                        FILERENAMEFM]));
    }
    else if (cmd.equals("delete")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.
                        FILEDELETEFM]));
    }
    else if (cmd.equals("exit")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsEntered[ActiveState.EXITFM]));
```

```
    }

    btn.setBorder(BorderFactory.createLineBorder(Color.WHITE));
  }

////////////////////////////////////////////////////////////////////////////
  private void mouse_exited_action(MouseEvent me) {
    JButton btn = (JButton) me.getSource();
    btn.setBackground(new Color(179, 186, 241));
    String cmd = btn.getActionCommand();
    if (cmd.equals("connect")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.CONNECTFM]));
    }
    else if (cmd.equals("disconnect")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.DISCONNECTFM]));
    }
    else if (cmd.equals("download")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.DOWNLOADFM]));
    }
    else if (cmd.equals("upload")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.UPLOADFM]));
    }
    else if (cmd.equals("mkDirectory")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.MKDIRFM]));
    }
    else if (cmd.equals("rename")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.FILERENAMEFM]));
    }
    else if (cmd.equals("delete")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.FILEDELETEFM]));
    }
    else if (cmd.equals("exit")) {
      btn.setIcon(new ImageIcon(ActiveState.btnsExited[ActiveState.EXITFM]));

    }

  }

////////////////////////////////////////////////////////////////////////////
  private void exit_action_performed(ActionEvent ae) {

    disconnect();
    this.dispose();
  }
```

```
}
```

RenameFileFrame.java

```
package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class RenameFileFrame
   extends JDialog {

  JTextField txtDir;
  JButton oky;

  public RenameFileFrame() {

    setSize(new Dimension(300, 150));
    setTitle("Enter new name");
    setLocation(300, 300);
    this.getContentPane().setLayout(null);
    txtDir = new JTextField(10);
    oky = new JButton("OK");
    txtDir.setBounds(80, 30, 130, 25);
    oky.setBounds(105, 70, 80, 20);
    this.getContentPane().add(txtDir);
    this.getContentPane().add(oky);

    oky.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        ok_action_performed(ae);
      }
    });

  }

  private void ok_action_performed(ActionEvent ae) {

    String fileName = txtDir.getText();
    RemoteFileManagerFrame.newFileName = fileName;
    RemoteFileManagerFrame.rename();
```

```
    this.dispose();
  }
}
```

SelectPageFrame.java

```
package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import java.lang.reflect.*;
import java.util.*;

import ECS.ActiveState;
import ECS.mainApplet;

public class SelectPageFrame
    extends CloseableFrame {
  public JList pagesList;
  JLabel header;
  JButton okBtn, cancelBtn;
  Vector list;
  mainApplet myApplet;

  public SelectPageFrame(mainApplet applet) {
    this.setSize(200, 200);
    this.setTitle("Pages List");
    this.getContentPane().setLayout(null);
    list = new Vector();
    myApplet = applet;

    switch (ActiveState.numberOfPages) {
      case 1:
        list.add("    1             " + mainApplet.page1.getPageTitle());
        break;
      case 2:
        list.add("    1             " + mainApplet.page1.getPageTitle());
        list.add("    2             " + mainApplet.page2.getPageTitle());
        break;
      case 3:
        list.add("    1             " + mainApplet.page1.getPageTitle());
```

```
    list.add("   2            " + mainApplet.page2.getPageTitle());
    list.add("   3            " + mainApplet.page3.getPageTitle());
    break;
  case 4:
    list.add("   1            " + mainApplet.page1.getPageTitle());
    list.add("   2            " + mainApplet.page2.getPageTitle());
    list.add("   3            " + mainApplet.page3.getPageTitle());
    list.add("   4            " + mainApplet.page4.getPageTitle());
    break;
  case 5:
    list.add("   1            " + mainApplet.page1.getPageTitle());
    list.add("   2            " + mainApplet.page2.getPageTitle());
    list.add("   3            " + mainApplet.page3.getPageTitle());
    list.add("   4            " + mainApplet.page4.getPageTitle());
    list.add("   5            " + mainApplet.page5.getPageTitle());
    break;

}
;

header = new JLabel("Page#            Title");
pagesList = new JList();
pagesList.setListData(list);
pagesList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

switch (ActiveState.currentPageNumber) {
  case 1:
    pagesList.setSelectedIndex(0);
    break;
  case 2:
    pagesList.setSelectedIndex(1);
    break;
  case 3:
    pagesList.setSelectedIndex(2);
    break;
  case 4:
    pagesList.setSelectedIndex(3);
    break;
  case 5:
    pagesList.setSelectedIndex(4);
    break;

}

okBtn = new JButton("OK");
```

```
  okBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      okBtn_actionPerformed(ae);
    }
  });

  cancelBtn = new JButton("Cancel");
  cancelBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      cancelBtn_actionPerformed(ae);
    }
  });

  pagesList.setBounds(0, 30, 200, 90);
  header.setBounds(5, 5, 200, 20);
  okBtn.setBounds(30, 140, 70, 20);
  cancelBtn.setBounds(100, 140, 70, 20);

  this.getContentPane().add(pagesList);
  this.getContentPane().add(header);
  this.getContentPane().add(okBtn);
  this.getContentPane().add(cancelBtn);

  this.backgroundLbl.setBounds(0, 0, 300, 300);
  this.getContentPane().add(this.backgroundLbl);
}

private void okBtn_actionPerformed(ActionEvent ae) {

  switch (pagesList.getSelectedIndex()) {
    case 0:
      myApplet.activatePage1();
      break;
    case 1:
      myApplet.activatePage2();
      break;
    case 2:
      myApplet.activatePage3();
      break;
    case 3:
      myApplet.activatePage4();
      break;
    case 4:
      myApplet.activatePage5();
      break;
```

```
    }
    ;
    this.dispose();
  }

  private void cancelBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

SimulationResultFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;

public class SimulationResultFrame
    extends CloseableFrame {
  public JTextArea result;
  JScrollPane scroll;
  JButton okBtn;
  JLabel background;
  public SimulationResultFrame() {
    this.setSize(550, 600);
    this.setTitle("Simulation Result Display");
    this.getContentPane().setLayout(null);
    scroll = new JScrollPane();
    result = new JTextArea();
    result.setEditable(false);

    add(scroll, 20, 20, 500, 500);
    scroll.getViewport().add(result);

    okBtn = new JButton("Ok");
    okBtn.setBackground(Color.WHITE);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
```

```
        okBtn_actionPerformed(ae);
      }
    }
    );
    add(okBtn, 250, 520, 60, 30);
    Image img = ActiveState.diffImages[ActiveState.SIMULATIONBACKGROUND];
    background = new JLabel(new ImageIcon(img));
    add(background, 0, 0, 550, 600);
  }

  private void okBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

SimulationSettingsFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;

public class SimulationSettingsFrame
    extends CloseableFrame {
  JLabel lblStartTime, lblEndTime, lblStepTime, lblStartSeconds,
      lblEndSeconds, lblStepSeconds, lblTitle;
  JTextField txtStartTime, txtEndTime, txtStepTime;
  JButton okBtn;
  String startTime, endTime, stepTime;
  mainApplet myApplet;
  public SimulationSettingsFrame(mainApplet applet) {
    this.getContentPane().setLayout(null);
    this.getContentPane().setBackground(new Color(91, 141, 255));
    this.setTitle("Simulation Settings");

    myApplet = applet;
```

```
okBtn = new JButton("Apply");
okBtn.setBackground(Color.WHITE);
okBtn.setForeground(Color.BLUE);
okBtn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    okBtn_actionPerformed(ae);
  }
});

lblStartTime = new JLabel("Start Time");
lblEndTime = new JLabel("End Time");
lblStepTime = new JLabel("Step Time");
lblTitle = new JLabel("Time Domain(Transient)");

txtStartTime = new JTextField();
txtEndTime = new JTextField();
txtStepTime = new JTextField();

lblStartSeconds = new JLabel("Seconds");
lblEndSeconds = new JLabel("Seconds");
lblStepSeconds = new JLabel("Seconds");

lblStartTime.setForeground(Color.BLUE);
lblEndTime.setForeground(Color.BLUE);
lblStepTime.setForeground(Color.BLUE);
lblStartSeconds.setForeground(Color.BLUE);
lblEndSeconds.setForeground(Color.BLUE);
lblStepSeconds.setForeground(Color.BLUE);
lblStartSeconds.setBackground(Color.GREEN);
lblEndSeconds.setBackground(Color.GREEN);
lblStepSeconds.setBackground(Color.GREEN);

add(lblTitle, 60, 10, 200, 10);
add(lblStartTime, 40, 40, 70, 20);
add(lblEndTime, 40, 70, 70, 20);
add(lblStepTime, 40, 100, 70, 20);
add(txtStartTime, 120, 40, 50, 20);
add(txtEndTime, 120, 70, 50, 20);
add(txtStepTime, 120, 100, 50, 20);
add(lblStartSeconds, 180, 40, 50, 20);
add(lblEndSeconds, 180, 70, 50, 20);
add(lblStepSeconds, 180, 100, 50, 20);
add(okBtn, 80, 140, 100, 20);
add(this.backgroundLbl, -100, -250, 500, 650);
txtStartTime.setText(myApplet.startTime);
```

```
    txtEndTime.setText(myApplet.endTime);
    txtStepTime.setText(myApplet.stepTime);
  }

  private void okBtn_actionPerformed(ActionEvent ae) {
    startTime = txtStartTime.getText();
    endTime = txtEndTime.getText();
    stepTime = txtStepTime.getText();
    myApplet.setSimulationSettings(startTime, endTime, stepTime);
    this.dispose();
  }
}
```

TransformerInputFrame.java

```
 package ECS.GUIFrameWindows;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

import ECS.*;
import ECS.CircuitComponents.*;

//*************************************************************************/
/
//                        TransformerInput Class
//*************************************************************************/
/
public class TransformerInputFrame
    extends CloseableFrame {

  JLabel lblLabel, lbln1, lbln2;
  JTextField txtLabel, txtn1, txtn2;
  JButton oky;
  Components curComp;
  mainApplet myApplet;
  public TransformerInputFrame() {
    Border border = BorderFactory.createEtchedBorder(Color.white, Color.blue);
```

```
this.setSize(300, 210);
lblLabel = new JLabel("Label:");
lblLabel.setBackground(SystemColor.control);
lblLabel.setForeground(Color.blue);

lbln1 = new JLabel("n1");
lbln1.setBackground(SystemColor.control);
lbln1.setForeground(Color.blue);

lbln2 = new JLabel("n2");
lbln2.setBackground(SystemColor.control);
lbln2.setForeground(Color.blue);
lbln2.setPreferredSize(new Dimension(18, 15));

txtLabel = new JTextField();

txtn1 = new JTextField();

txtn2 = new JTextField();

oky = new JButton("OK");
oky.setBackground(Color.white);
oky.setForeground(Color.blue);
oky.addMouseListener(new MouseAdapter() {
  public void mouseClicked(MouseEvent me) {
    oky_mouseClicked(me);
  }
});
oky.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
txtLabel.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
txtn1.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
    oky_keyPressed(ke);
  }
});
txtn2.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent ke) {
```

```
      oky_keyPressed(ke);
    }
   });

   add(lblLabel, 10, 20, 34, 13);
   add(lbln1, 10, 60, 34, 15);
   add(lbln2, 150, 60, 34, 14);
   add(txtLabel, 60, 20, 86, 21);
   add(txtn1, 60, 60, 86, 21);
   add(txtn2, 180, 60, 82, 21);
   add(oky, 100, 100, 73, 25);
   add(this.backgroundLbl, -100, -250, 500, 650);
  }

  public void setComponent(Components comp, mainApplet applet) {
   curComp = comp;
   myApplet = applet;
   this.setTitle(comp.getGenericName());
   txtLabel.setText(comp.getComponentName());
   Double n1 = new Double(comp.getN1());
   Double n2 = new Double(comp.getN2());
   txtn1.setText(n1.toString());
   txtn2.setText(n2.toString());
  }

  private void oky_mouseClicked(MouseEvent me) {
   this.setVisible(false);
   curComp.setComponentName(txtLabel.getText());
   Double n1 = new Double(txtn1.getText());
   Double n2 = new Double(txtn2.getText());
   curComp.setN1(n1.doubleValue());
   curComp.setN2(n2.doubleValue());
   curComp.setValue(n1.doubleValue() / n2.doubleValue());
   myApplet.gridCanvas.gridImage.clearRect(0, 0, 650, 550);
   myApplet.gridCanvas.drawGridLayout(ActiveState.startPoint);
   myApplet.drawAllComps();
   myApplet.gridCanvas.redraw();

  }

////////////////////////////////////////////////////////////////////////////
  private void oky_keyPressed(KeyEvent ke) {

   if (ke.getKeyCode() == 10) {
    this.setVisible(false);
```

```
      curComp.setComponentName(txtLabel.getText());
      Double n1 = new Double(txtn1.getText());
      Double n2 = new Double(txtn2.getText());
      curComp.setN1(n1.doubleValue());
      curComp.setN2(n2.doubleValue());
      curComp.setValue(n1.doubleValue() / n2.doubleValue());
      myApplet.redrawGridCanvas();
    }
  }

}
//*************************************************************************/
/
//                          END                              //
//*************************************************************************/
/
```

AlertMessage.java

```
 package ECS.GUIMessages;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;

public class AlertMessage
    extends JDialog {

  JLabel msgLbl, backgroundLbl;
  JButton okBtn;
  public AlertMessage(String msg) {
    this.setTitle("Alert");
    this.setSize(325, 160);
    this.setLocation(400, 400);
    this.getContentPane().setBackground(Color.WHITE);
    this.getContentPane().setLayout(null);
    Image img = ActiveState.diffImages[ActiveState.ALERTMESSAGEBACKGROUND];
    backgroundLbl = new JLabel(new ImageIcon(img));
```

```
    msgLbl = new JLabel(msg);
    okBtn = new JButton("Ok");
    okBtn.setBackground(Color.WHITE);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });

    add(msgLbl, 50, 25, 270, 35);
    add(okBtn, 125, 75, 60, 20);
    add(backgroundLbl, -5, -15, 325, 150);
  }

  public void add(Component comp, int x, int y, int width, int height) {
    comp.setBounds(x, y, width, height);
    this.getContentPane().add(comp);
  }

  private void okBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

ErrorMessage.java

```
 package ECS.GUIMessages;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;

public class ErrorMessage
    extends JDialog {

  JLabel msgLbl, backgroundLbl;
  JButton okBtn;
  public ErrorMessage(String msg) {
```

```
    this.setTitle("Error");
    this.setSize(325, 160);
    this.setLocation(400, 400);
    this.getContentPane().setBackground(Color.WHITE);
    this.getContentPane().setLayout(null);
    Image img = ActiveState.diffImages[ActiveState.ERRORMESSAGEBACKGROUND];
    backgroundLbl = new JLabel(new ImageIcon(img));

    msgLbl = new JLabel(msg);
    okBtn = new JButton("Ok");
    okBtn.setBackground(Color.WHITE);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });

    add(msgLbl, 85, 25, 250, 35);
    add(okBtn, 125, 75, 60, 20);
    add(backgroundLbl, -5, -15, 325, 150);
  }

  public void add(Component comp, int x, int y, int width, int height) {
    comp.setBounds(x, y, width, height);
    this.getContentPane().add(comp);
  }

  private void okBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

Acknowledgments.java

```
 package ECS.GUIMessages;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
import ECS.ActiveState;

public class Acknowledgments
    extends JDialog {

  JLabel lbl;
  JButton okBtn;
  public Acknowledgments() {
    this.setSize(400, 270);
    this.setTitle("Acknowledgments");
    this.getContentPane().setLayout(null);
    this.setLocation(350, 350);

    this.getContentPane().setBackground(new Color(91, 141, 255));
    lbl = new JLabel(new ImageIcon(ActiveState.diffImages[ActiveState.
                                ACKNOWLEDGMENT]));
    okBtn = new JButton("OK");
    okBtn.setBackground(Color.WHITE);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });

    lbl.setBounds(0, 0, 400, 230);
    okBtn.setBounds(145, 180, 70, 20);

    this.getContentPane().add(okBtn);
    this.getContentPane().add(lbl);
  }

  private void okBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }
}
```

InfoMessage.java

```
 package ECS.GUIMessages;

import java.awt.*;
```

```
import java.awt.event.*;
import javax.swing.*;

import ECS.*;

public class InfoMessage
    extends JDialog {

  JLabel msgLbl, backgroundLbl;
  JButton okBtn;
  public InfoMessage(String msg) {
    this.setTitle("Tip");
    this.setSize(355, 165);
    this.setLocation(400, 400);
    this.getContentPane().setBackground(Color.WHITE);
    this.getContentPane().setLayout(null);
    Image img = ActiveState.diffImages[ActiveState.INFOMESSAGEBACKGROUND];
    backgroundLbl = new JLabel(new ImageIcon(img));

    msgLbl = new JLabel(msg);
    okBtn = new JButton("Ok");
    okBtn.setBackground(Color.WHITE);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });

    add(msgLbl, 55, 25, 300, 35);
    add(okBtn, 140, 75, 60, 20);
    add(backgroundLbl, -5, -15, 325, 150);
  }

  public void add(Component comp, int x, int y, int width, int height) {
    comp.setBounds(x, y, width, height);
    this.getContentPane().add(comp);
  }

  private void okBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

QuestionMessage.java

```
 package ECS.GUIMessages;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;

public class QuestionMessage
    extends JDialog {

  JLabel backgroundLbl;
  JTextArea msgTxt;
  JButton saveBtn, cancelBtn;
  mainApplet myApplet;
  public QuestionMessage(String msg, mainApplet applet) {
    this.setTitle("Question");
    this.setSize(325, 160);
    this.setLocation(400, 400);
    this.getContentPane().setBackground(Color.WHITE);
    this.getContentPane().setLayout(null);
    Image img = ActiveState.diffImages[ActiveState.QUESTION];
    myApplet = applet;
    //alertLbl=new JLabel(new ImageIcon(img));
    msgTxt = new JTextArea(msg);
    msgTxt.setEditable(false);
    saveBtn = new JButton("Save");
    saveBtn.setBackground(Color.WHITE);
    saveBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        saveBtn_actionPerformed(ae);
      }
    });
    cancelBtn = new JButton("Cancel");
    cancelBtn.setBackground(Color.WHITE);
    cancelBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        cancelBtn_actionPerformed(ae);
      }
    });
    img = ActiveState.diffImages[ActiveState.QUESTIONBACKGROUND];
```

```
    backgroundLbl = new JLabel(new ImageIcon(img));
    add(saveBtn, 90, 70, 80, 20);
    add(cancelBtn, 170, 70, 80, 20);
    add(backgroundLbl, -5, -60, 325, 210);
  }

  public void add(Component comp, int x, int y, int width, int height) {
    comp.setBounds(x, y, width, height);
    this.getContentPane().add(comp);
  }

  private void saveBtn_actionPerformed(ActionEvent ae) {
    myApplet.saveSchematicToLocal();
    myApplet.removeCurrentPage();
    this.dispose();
  }

  private void cancelBtn_actionPerformed(ActionEvent ae) {
    myApplet.removeCurrentPage();
    this.dispose();
  }
}
```

ContactUS.java

```
package ECS.GUIMessages;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.ActiveState;

public class ContactUS
    extends JDialog {

  JLabel lbl;
  JButton okBtn;
  public ContactUS() {
    this.setSize(300, 350);
    this.setLocation(350, 350);
    this.setTitle("Contact Us");
```

```
    this.getContentPane().setLayout(null);

    this.getContentPane().setBackground(new Color(91, 141, 255));
    lbl = new JLabel(new ImageIcon(ActiveState.diffImages[ActiveState.
                              OURCONTACT]));
    okBtn = new JButton("OK");
    okBtn.setBackground(Color.WHITE);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });

    lbl.setBounds(0, 0, 300, 280);
    okBtn.setBounds(115, 260, 70, 20);

    this.getContentPane().add(lbl);
    this.getContentPane().add(okBtn);

  }

  private void okBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }
}
```

WelcomeMessage.java

```
 package ECS.GUIMessages;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import ECS.*;

public class WelcomeMessage
    extends JDialog {
  JLabel msg;
  JButton okBtn;

  public WelcomeMessage() {
```

```
    this.getContentPane().setBackground(Color.WHITE);
    this.getContentPane().setLayout(null);
    this.setSize(500, 395);
    this.setLocation(400, 400);
    this.setDefaultLookAndFeelDecorated(true);
    this.setTitle("Welcome to the Electrical Circuit Solver");
    Image img = ActiveState.diffImages[ActiveState.WELCOMEMESSAGE];
    msg = new JLabel(new ImageIcon(img));
    msg.setBounds(0, -60, 500, 450);
    okBtn = new JButton("Ok");
    okBtn.setBounds(230, 310, 50, 20);
    okBtn.setBackground(Color.WHITE);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });
    this.getContentPane().add(okBtn);

    this.getContentPane().add(msg);

  }

  private void okBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

ColorPanel.java

```
 package ECS.GUPanels;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

import ECS.*;
import ECS.GUIFrameWindows.*;
```

```
public class ColorPanel
    extends JFrame {

  JLabel backgroundLbl;
  JColorChooser cs;
  mainApplet myApplet;
  Color color;
  JButton okBtn, cancelBtn;
  String name;
  public ColorPanel(String str, mainApplet applet) {

    Image img = ActiveState.diffImages[ActiveState.MSGBACKGROUND];
    backgroundLbl = new JLabel(new ImageIcon(img));
    backgroundLbl.setBounds(0, 0, 440, 420);

    name = str;
    this.getContentPane().setLayout(null);
    this.getContentPane().setBackground(new Color(91, 141, 255));

    this.setSize(440, 440);

    myApplet = applet;
    cs = new JColorChooser();
    cs.getSelectionModel().addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent ce) {

        changeColor();
      }
    }
    );
    cs.setBounds(5, 0, 440, 350);
    cs.setBackground(Color.BLUE);
    cs.setForeground(Color.BLUE);
    this.getContentPane().add(cs);
    okBtn = new JButton("Ok");
    okBtn.setBounds(150, 355, 80, 20);
    okBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        okBtn_actionPerformed(ae);
      }
    });
    cancelBtn = new JButton("Cancel");
    cancelBtn.setBounds(230, 355, 80, 20);
    cancelBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
```

```
      cancelBtn_actionPerformed(ae);
    }
  });
  this.getContentPane().add(okBtn);
  this.getContentPane().add(cancelBtn);

  if (name.equals("drawingFillColor")) {
    cs.setColor(applet.drawingFillColor);
    this.setTitle("Fill Color");
  }
  else if (name.equals("drawingLineColor")) {
    cs.setColor(applet.drawingLineColor);
    this.setTitle("Line Color");
  }
  else if (name.equals("drawingTextColor")) {
    cs.setColor(applet.drawingTextColor);
    this.setTitle("Text Color");
  }
  else if (name.equals("canvasForgroundColor")) {
    cs.setColor(applet.canvasForegroundColor);
    this.setTitle("Forground Color");
  }
  else if (name.equals("canvasBackgroundColor")) {
    cs.setColor(applet.canvasBackgroundColor);
    this.setTitle("Background Color");
  }

}

private void changeColor() {
  color = cs.getColor();

}

private void okBtn_actionPerformed(ActionEvent ae) {

  if (name.equals("drawingFillColor")) {
    myApplet.drawingFillColor = color;
    myApplet.redrawGridCanvas();
  }
  else if (name.equals("drawingLineColor")) {
    myApplet.drawingLineColor = color;
    myApplet.redrawGridCanvas();
  }
  else if (name.equals("drawingTextColor")) {
```

```
      myApplet.drawingTextColor = color;
      myApplet.redrawGridCanvas();
    }
    else if (name.equals("canvasForgroundColor")) {
      CanvasPropertiesFrame.canvasForColorTxt.setBackground(color);
      myApplet.redrawGridCanvas();
    }
    else if (name.equals("canvasBackgroundColor")) {
      CanvasPropertiesFrame.canvasBackColorTxt.setBackground(color);
      myApplet.redrawGridCanvas();
    }

    this.dispose();
  }

  private void cancelBtn_actionPerformed(ActionEvent ae) {
    this.dispose();
  }

}
```

GraphicalPanel.java

```
package ECS.GUPanels;

import java.awt.*;
import javax.swing.*;

import ECS.*;

////////////////////////////////////////////////////////////////////////////
public class GraphicalPanel
    extends JPanel {

  public Graphics gridImage, picImage;
  Image img, imgcomp;
  int xpos, ypos;
  boolean init = false;
  Image gImg;
//*************************************//
//*************************************//
```

```
  public GraphicalPanel() {

  }

//************************************//
  public void paintComponent(Graphics g) {

    g.drawImage(ActiveState.diffImages[ActiveState.BACKGROUND], 0, 0, this);

  }

//**********************************//
  public void redraw() {
    repaint();
  }

}
////////////////////////////////////////////////////////////////////////////
```

GraphicalResultPanel.java

```
 package ECS.GUPanels;

import java.awt.*;
import javax.swing.*;

import ECS.*;

////////////////////////////////////////////////////////////////////////////
public class GraphicalResultPanel
    extends JPanel {

  public Graphics picImage;
  boolean init = false;

//************************************//
//************************************//
  public GraphicalResultPanel() {

    this.setBackground(Color.blue);
    this.setBounds(new Rectangle(385, 590, 375, 210));
    picImage = this.getGraphics();
```

```
  }

//*************************************//
  public void drawGridArea(Graphics g) {

    g.setColor(Color.BLUE);
    g.drawLine(35, 175, 345, 175);
    g.drawLine(40, 20, 40, 175);
    g.setColor(Color.black);
    for (int y = 25; y < 175; y += 25) {
      for (int x = 35; x < 350; x += 5) {
        g.drawLine(x, y, x, y);
      }
    }
    for (int x = 40; x < 360; x += 25) {
      for (int y = 20; y < 185; y += 5) {
        g.drawLine(x, y, x, y);
      }
    }

    g.setColor(Color.BLACK);
    g.drawString("Time", 180, 207);
    g.drawString("V/I", 1, 100);

  }

  private void init_paint() {
    picImage = this.getGraphics();

  }

//*************************************//
  public void paintComponent(Graphics g) {

    if (init != true) {
      init = true;
      init_paint();
    }

    g.drawImage(ActiveState.diffImages[ActiveState.PANELBACKGROUND], 0, 0, this);
  }

//*************************************//
  public void redraw() {
    repaint();
```

```
  }

//************************************//

}
////////////////////////////////////////////////////////////////////////////
```

GridCanvas.java

```
 package ECS.GUPanels;

import java.lang.reflect.*;

import java.awt.*;
import javax.swing.*;

import ECS.*;

////////////////////////////////////////////////////////////////////////////
public class GridCanvas
    extends JPanel {

  public Graphics gridImage, picImage;
  Image img, imgcomp;
  int xpos, ypos;
  boolean init = false;
  public static int width = 2500;
  public static int height = 2500;
  public static Color backColor, foreColor;
//************************************//
//************************************//
  public GridCanvas(int w, int h, Color b, Color f) {

    this.setPreferredSize(new Dimension(w, h));
    width = w;
    height = h;
    backColor = b;
    foreColor = f;
  }

//************************************//
```

```
  public void drawGridLayout(int[] point) {
    for (int x = point[0]; x < point[0] + width; x += 25) {
      for (int y = point[1]; y < point[1] + height; y += 25) {
        gridImage.drawLine(x, y, x, y);
      }
    }
  }

//************************************//
  private void init_paint() {
    img = this.createImage(width, height);

    //imgcomp=this.createImage(50,50);
    gridImage = img.getGraphics();

    //picImage=img.getGraphics();
    for (int i = 0; i < Array.getLength(ActiveState.StrCompGifs); i++) {
      gridImage.drawImage(ActiveState.ImageNormal[i], 50, 50, this);
      gridImage.drawImage(ActiveState.ImageActive[i], 50, 50, this);
    }

    gridImage.clearRect(0, 0, width, height);
    gridImage.setColor(backColor);
    gridImage.fillRect(0, 0, width, height);
    gridImage.setColor(foreColor);
    int[] p = {
        0, 0};
    drawGridLayout(p);

  }

//************************************//
  public void paintComponent(Graphics g) {

    if (init != true) {
      init = true;
      init_paint();
    }

    g.drawImage(img, 0, 0, this);

  }

//************************************//
  public void redraw() {
```

```
    repaint();
  }

//***********************************//
  public static void setGridWidth(int w) {
    width = w;
  }

  public static int getGridWidth() {
    return width;
  }

  public static void setGridHeight(int h) {
    height = h;
  }

  public static int getGridHeight() {
    return height;
  }

  public void setGridSize(int w, int h) {
    width = w;
    height = h;
    repaint();
  }

//************************************
   public void resizeCanvas(int w, int h) {
     this.setPreferredSize(new Dimension(w, h));
     width = w;
     height = h;
     this.setSize(w, h);
     init_paint();
     redraw();
   }

}
////////////////////////////////////////////////////////////////////////////////
```

NetlistOutputPanel.java

```
 package ECS.GUPanels;

import java.awt.*;
import javax.swing.*;

import ECS.*;

public class NetlistOutputPanel
    extends JPanel {
  JScrollPane scrollPanel;
  public JTextArea netText;

  public NetlistOutputPanel() {
    this.setBounds(new Rectangle(5, 590, 375, 210));
    this.setLayout(null);
    this.setBackground(new Color(213, 224, 254));

    scrollPanel = new JScrollPane();
    scrollPanel.setBounds(25, 25, 325, 160);

    netText = new JTextArea();
    netText.setEditable(false);
    netText.setBounds(0, 0, 390, 210);
    netText.setForeground(Color.BLACK);
    netText.setText("Simulation Results");
    netText.setBorder(null);
    netText.setBackground(new Color(213, 224, 254));
    scrollPanel.getViewport().add(netText, null);

    this.add(scrollPanel);

  }

//*********************************//
  public void paintComponent(Graphics g) {
```

```
    g.drawImage(ActiveState.diffImages[ActiveState.NETLISTPANELBACKGROUND], 0,
            0, this);

  }

}
```

ShowNetlistPanel.java

```
 package ECS.GUPanels;

import java.awt.*;
import javax.swing.*;

import ECS.*;

public class ShowNetlistPanel
    extends JPanel {

  JScrollPane scrollPanel;
  public JTextArea netText;

  ShowNetlistPanel() {

    this.setBounds(new Rectangle(5, 590, 375, 210));
    this.setLayout(null);
    this.setBackground(new Color(213, 224, 254));

    scrollPanel = new JScrollPane();
    scrollPanel.setBounds(25, 25, 325, 160);

    netText = new JTextArea();
    netText.setEditable(false);
    netText.setBounds(0, 0, 390, 210);
    netText.setForeground(Color.BLACK);
    netText.setText("*Schematic Netlist");
    netText.setBorder(null);
```

```
    netText.setBackground(new Color(213, 224, 254));
    scrollPanel.getViewport().add(netText, null);

    this.add(scrollPanel);

  }

//************************************//
  public void paintComponent(Graphics g) {

    g.drawImage(ActiveState.diffImages[ActiveState.NETLISTPANELBACKGROUND], 0,
            0, this);

  }

}
```

StatusBarPanel.java

```
 package ECS.GUPanels;

import java.awt.*;
import javax.swing.*;

import ECS.*;

////////////////////////////////////////////////////////////////////////////
public class StatusBarPanel
    extends JPanel {

  public Graphics gridImage, picImage;
  Image img, imgcomp;
  int xpos, ypos;
  boolean init = false;
//************************************//
//************************************//
  public StatusBarPanel() {

    //  this.setPreferredSize(new Dimension(2500,2500));
    //this.setSize(new Dimension(3000,3000));
```

```
  }

//************************************//
  private void init_paint() {
    img = this.createImage(500, 25);

    //imgcomp=this.createImage(50,50);
    gridImage = img.getGraphics();

    //picImage=img.getGraphics();
    gridImage.drawImage(ActiveState.diffImages[ActiveState.STATUSBAR], 0, 0, this);
  }

//************************************//
  public void paintComponent(Graphics g) {

    //Graphics2D gin = (Graphics2D) g;
    //AffineTransform origTransform = gin.getTransform();

    //gin.setTransform(origTransform);

    if (init != true) {
      init = true;
      init_paint();
    }

    g.drawImage(img, 0, 0, this);

  }

//************************************//
  public void redraw() {
    repaint();
  }

}
////////////////////////////////////////////////////////////////////////////////
```

ServerApplication.java

```
 import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import javax.swing.tree.*;

public class ServerApplication {

  static ServerSocket commandServer, dataServer, simulateServer;
  static Socket commandClient, dataClient, simulateClient;
  static ObjectInputStream inputCommand, inputData;
  static ObjectOutputStream outputCommand;
  static String strCommand;
  static DataOutputStream DOS;
  static DataInputStream DIS;
  static FileOutputStream FOS;
  static InputStream FIS;
  static DefaultMutableTreeNode root;

  static class fileManager
     implements Runnable {

    public void run() {

      while (true) {

        try {
          commandServer = new ServerSocket(3500);
          dataServer = new ServerSocket(3600);
          System.out.println("Server is listening...");
          commandClient = commandServer.accept();
          dataClient = dataServer.accept();
          System.out.println("Client is connected");
          root = new DefaultMutableTreeNode();
          File currFolder = new File(".");
          File[] ListOfFiles = currFolder.listFiles();
          //addNode(root,ListOfFiles);
```

```
commandClient.setReuseAddress(true);
dataClient.setReuseAddress(true);
while (!commandClient.isClosed()) {
  System.out.println("waiting other command");
  inputCommand = new ObjectInputStream(commandClient.getInputStream());
  try {
    strCommand = (String) inputCommand.readObject();
    System.out.println(strCommand);

    if (strCommand.equals("disconnect")) {
      commandServer.close();
      dataServer.close();
      commandClient.close();
      dataClient.close();

    }
    else if (strCommand.equals("list")) {
      outputCommand = new ObjectOutputStream(commandClient.
          getOutputStream());
      File currDir = new File(".");
      File[] filesList = currDir.listFiles();
      //DefaultMutableTreeNode r=new DefaultMutableTreeNode("SHADI");
      //DefaultMutableTreeNode r1=new DefaultMutableTreeNode("SH");
      //r.add(r1);
      root = new DefaultMutableTreeNode();
      addNode(root, filesList);
      outputCommand.writeObject(root);

      /* JFrame f=new JFrame("HI");
       f.setSize(700,700);
       f.getContentPane().setLayout(null);
       JTree t=new JTree(root);
       t.setBounds(0,0,300,400);
       f.getContentPane().add(t);
       f.show();*/

    }
    else if (strCommand.equals("download")) {
      try {

        inputCommand = new ObjectInputStream(dataClient.
            getInputStream());
        String filename = (String) inputCommand.readObject();

        DOS = new DataOutputStream(dataClient.getOutputStream());
```

```
    FIS = new FileInputStream(filename);

    int c = FIS.read();
    while (c != -1) {

      DOS.write(c);
      c = FIS.read();
    }

    DOS.close();
    FIS.close();
  }
  catch (SocketException se) {
    System.out.println(se.getMessage());
  }
}
else if (strCommand.equals("upload")) {

  inputData = new ObjectInputStream(dataClient.getInputStream());
  String fileName = (String) inputData.readObject();

  DIS = new DataInputStream(dataClient.getInputStream());
  FOS = new FileOutputStream(fileName);
  int c = DIS.read();
  while (c != -1) {
    System.out.println(c);
    FOS.write(c);
    c = DIS.read();
  }
  FOS.close();
  DIS.close();

  outputCommand = new ObjectOutputStream(commandClient.
      getOutputStream());
  File currDir = new File(".");
  File[] filesList = currDir.listFiles();
  //DefaultMutableTreeNode r=new DefaultMutableTreeNode("SHADI");
  //DefaultMutableTreeNode r1=new DefaultMutableTreeNode("SH");
  //r.add(r1);
  root = new DefaultMutableTreeNode();
  addNode(root, filesList);
  outputCommand.writeObject(root);

}
else if (strCommand.equals("delete")) {
```

```
  String filename = (String) inputCommand.readObject();

  File file = new File(filename);
  file.delete();

  outputCommand = new ObjectOutputStream(commandClient.
     getOutputStream());
  File currDir = new File(".");
  File[] filesList = currDir.listFiles();
  //DefaultMutableTreeNode r=new DefaultMutableTreeNode("SHADI");
  //DefaultMutableTreeNode r1=new DefaultMutableTreeNode("SH");
  //r.add(r1);
  root = new DefaultMutableTreeNode();
  addNode(root, filesList);
  outputCommand.writeObject(root);

 }
 else if (strCommand.equals("mkDir")) {
  String filename = (String) inputCommand.readObject();

  File file = new File(filename);
  file.mkdir();

  outputCommand = new ObjectOutputStream(commandClient.
     getOutputStream());
  File currDir = new File(".");
  File[] filesList = currDir.listFiles();
  //DefaultMutableTreeNode r=new DefaultMutableTreeNode("SHADI");
  //DefaultMutableTreeNode r1=new DefaultMutableTreeNode("SH");
  //r.add(r1);
  root = new DefaultMutableTreeNode();
  addNode(root, filesList);
  outputCommand.writeObject(root);

 }
 else if (strCommand.equals("rmDir")) {
  String filename = (String) inputCommand.readObject();

  File file = new File(filename);
  File[] list = file.listFiles();
  deleteFiles(list);
  list = file.listFiles();

  while (list.length != 0) {
    deleteDirectories(list);
```

```
    list = file.listFiles();
   }
   file.delete();

   outputCommand = new ObjectOutputStream(commandClient.
      getOutputStream());
   File currDir = new File(".");
   File[] filesList = currDir.listFiles();
   //DefaultMutableTreeNode r=new DefaultMutableTreeNode("SHADI");
   //DefaultMutableTreeNode r1=new DefaultMutableTreeNode("SH");
   //r.add(r1);
   root = new DefaultMutableTreeNode();
   addNode(root, filesList);
   outputCommand.writeObject(root);

  }
  else if (strCommand.equals("rename")) {
   String filename = (String) inputCommand.readObject();

   File file = new File(filename);
   String newFileName = (String) inputCommand.readObject();
   file.renameTo(new File(newFileName));

   outputCommand = new ObjectOutputStream(commandClient.
      getOutputStream());
   File currDir = new File(".");
   File[] filesList = currDir.listFiles();
   //DefaultMutableTreeNode r=new DefaultMutableTreeNode("SHADI");
   //DefaultMutableTreeNode r1=new DefaultMutableTreeNode("SH");
   //r.add(r1);
   root = new DefaultMutableTreeNode();
   addNode(root, filesList);
   outputCommand.writeObject(root);

  }
}
catch (ClassNotFoundException cnfe) {
  System.out.println(cnfe);
}
if (commandServer.isClosed()) {
  System.out.println("commandServer CLOSED");
}
else if (dataServer.isClosed()) {
  System.out.println("dataServer CLOSED");
}
```

```
        else if (commandClient.isClosed()) {
          System.out.println("commandClient CLOSED");
        }
        else if (dataClient.isClosed()) {
          System.out.println("dataClient CLOSED");
        }
      }
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }

    try {
      commandServer.close();
      dataServer.close();
    }
    catch (IOException ioe) {
      System.out.println(ioe);
    }
   }

  }

}

static class simulate
   implements Runnable {

  public void run() {

    while (true) {

      Process pspiceProcess = null;
      Runtime runPSpice = null;
      try {
        simulateServer = new ServerSocket(3300);

        System.out.println("Server Simulation is waiting for listening");
        simulateClient = simulateServer.accept();
        System.out.println("Client is connected");

        File cirFile = new File("PSpice/circuit.cir");
        if (cirFile.exists()) {
          cirFile.delete();
        }
```

```
        File outFile = new File("PSpice/circuit.out");
        if (outFile.exists()) {
          outFile.delete();
        }
        File csdFile = new File("PSpice/circuit.csd");
        if (csdFile.exists()) {
          csdFile.delete();

        }

        FileOutputStream FOS = new FileOutputStream("PSpice/circuit.cir");
        ObjectInputStream OIS = new ObjectInputStream(simulateClient.
            getInputStream());

        String cirML = "";
        try {
          cirML = (String) OIS.readObject();
        }
        catch (ClassNotFoundException cnfe) {
          System.out.println(cnfe);
        }

        for (int i = 0; i < cirML.length(); i++) {
          FOS.write(cirML.charAt(i));
        }
        FOS.close();

/////
        String command = "runPSpice.bat";
        System.out.println(command);
        pspiceProcess = null;
        runPSpice = Runtime.getRuntime();
        pspiceProcess = runPSpice.exec(command);
        try {
          Thread.sleep(1500);
          pspiceProcess.waitFor();
        }
        catch (InterruptedException ie) {
          ;
        }
        runPSpice.gc();
        pspiceProcess.destroy();

      }
      catch (IOException ioe) {
```

```
        System.out.println(ioe);
      }

      try {
        try {

          ObjectOutputStream OOS = new ObjectOutputStream(simulateClient.
            getOutputStream());
          InputStream FIS1 = new FileInputStream("PSpice/circuit.out");
          File outputFile = new File("PSpice/circuit.out");

          Long x = new Long(outputFile.length());
          String s = x.toString();
          Integer o = new Integer(s);
          System.out.println(o);

          byte buff1[] = new byte[o.intValue()];

          int m = FIS1.read(buff1);
          String resultStr = new String(buff1);
          System.out.println(resultStr);
          OOS.writeObject(resultStr);
          System.out.println(resultStr);

//// SEND GRAPHIC DATA.....
          if (resultStr.indexOf("Error") == -1) {
            File csdFile = new File("PSpice/circuit.csd");
            if (csdFile.exists()) {
              InputStream FIS2 = new FileInputStream("PSpice/circuit.csd");

              Long len = new Long(csdFile.length());
              String lenstr = len.toString();
              Integer lenint = new Integer(lenstr);
              byte buff2[] = new byte[lenint.intValue()];

              int k = FIS2.read(buff2);
              String csdStr = new String(buff2);

              OOS.writeObject(csdStr);
              System.out.println(csdStr);
              FIS2.close();
            }
            else {
              OOS.writeObject(new String(""));
            }
```

```
          }

          OOS.close();
          FIS1.close();
          ;
          simulateServer.close();

          try {
            Thread.sleep(1500);
          }
          catch (InterruptedException ie) {}
          runPSpice.gc();
          pspiceProcess.destroy();

        }
        catch (FileNotFoundException fne) {
          System.out.println(fne);
        }
      }
      catch (IOException ioe) {
        System.out.println(ioe);
      }

      //kill PSPICE process.....
      //pspiceProcess.destroy();

    }

  }

}

////////////////////////////////////////////////////////////////////////

  public static void main(String[] args) {

    Thread FM = new Thread(new fileManager());
    FM.start();
    Thread SM = new Thread(new simulate());
    SM.start();

  }

////////////////////////////////////////////////////////////////////////
```

```
  private static void addNode(DefaultMutableTreeNode parent, File[] filesList) {
    for (int i = 0; i < filesList.length; i++) {

      if (filesList[i].isDirectory()) {
        DefaultMutableTreeNode subDir = new DefaultMutableTreeNode(filesList[i].
            getName());

        File[] subDirList = filesList[i].listFiles();
        addNode(subDir, subDirList);
        parent.add(subDir);
      }
      else {
        DefaultMutableTreeNode node = new DefaultMutableTreeNode(filesList[i].
            getName());
        parent.add(node);
      }
    }

  }

////////////////////////////////////////////////////////////////////////////
  public static void deleteFiles(File[] filesList) {
    for (int i = 0; i < filesList.length; i++) {
      if (filesList[i].isDirectory()) {
        File[] list = filesList[i].listFiles();
        deleteFiles(list);

      }
      else {
        filesList[i].delete();
      }
    }
  }

////////////////////////////////////////////////////////////////////////////
  public static void deleteDirectories(File[] filesList) {
    for (int i = 0; i < filesList.length; i++) {
      if (filesList[i].isDirectory()) {
        File[] list = filesList[i].listFiles();

        if (list.length == 0) {
          filesList[i].delete();
        }
        else {
          deleteDirectories(list);
```

```
        }
      }
    }
  }
////////////////////////////////////////////////////////////////////////////////

}
```

LIST OF REFERENCES

[1] Qiong Zhang, “Design and Evaluation of an Internet-based Circuit Design Package used in an undergraduate Engineering Circuit Course”, Thesis, University of Central Florida, Spring 2003

[2] Sayed Raihan, “Evaluation and Improvement of an Internet-based Circuit Design Package”, Thesis, University of Central Florida, 2001

[3] Smitha Reddy, “Online PSPICE Based Simulation of DC Electrical Engineering Circuits”, Thesis, University of Central Florida, 2001

[4] H.M.Deitel, P,J Deitel “Java How to Program” ISBN 0-13-012507-5

[5] Arthur Griffith “Java Master Reference” ISBN 0-7645-3084-4