3

Graphical Tools for Modelling Physical Systems

by

Richard Kahil-Lateef Branton

Submitted to the

Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements
for the degrees of

Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

May, 1995

Author _____

Department of Electrical Engineering and Computer Science
May, 1995

Certified by _____

Dr. Kamal Youcef-Toumi
Associate Professor
Thesis Supervisor

Accepted by _____

F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Graphical Tools for Modelling Physical Systems

by
Richard K. Branton

Submitted to the Department of Electrical Engineering and Computer Science

May, 1995

In partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

There is in the modelling of physical systems a representation referred to as bond graph. This representation, devised at the Massachusetts Institute of Technology (MIT), takes into account energy interactions between systems and subsystems as well as the causality of the interactions between these systems. This thesis presents a graphical application which allows users to build models of physical systems using the bond graph representation. This application can then generate data necessary to analyze and simulate the user's model with a separate program. The modeler runs on a DOS based machine running Microsoft Windows 3.1.

Thesis Supervisor:Dr. Kamal Youcef-Toumi
Title: Associate Professor of Mechanical Engineering

Acknowledgments

I would like to thank Professor Kamal Youcef-Toumi, my thesis advisor, for his help, support, encouragement, enthusiasm, and active involvement throughout my work on this thesis. I would also like to thank Shih-Ying Huang for his support, help, and suggestions. Finally, I would like to thank my parents without whom I would not be in the position I am today.

Contents

List of Figures

1 Introduction

1.1 Motivation of Thesis

Many features which were considered luxuries in products just a few years ago are now common place. The increase in the quality of what is standard for a product has led to new requirements for system design which emphasizes the systems' approach, including the integration of design, modelling and control.

The demand for high performance systems, as emphasized by a variety of applications, has increased the difficulty of the system design problem. For these applications, systems involving several energy domains, such as electrical, mechanical, magnetic, fluidic, thermal, chemical, etc. are involved. Thus, the task of designing, modelling and analyzing a system becomes more critical when considering the severe system performance requirements and economic competitiveness.

For multi-energy domain systems, the properties and constraints from all the domains need to be related. The tools used to represent these systems need to use a unified language or representation in order to deal with the multiple energy domains. In addition, the representation needs to have several important properties. The energy method, which provides a systematic procedure for dealing with multidisciplinary applications and systems, is ideal for revealing information to help in the understanding of interactions between components and subsystems. It also contains information which helps in the determination of the system limitations and helps in identifying ways of improving the system design. Implementing the energy method as software can provide a useful rapid prototyping tool, which can allow the user to do analysis and design correctly and quickly[1].

Bond graphs, a representation devised at the Massachusetts Institute of Technology (M.I.T.), is an energy method which takes into account energy interactions between systems and subsystems as well as the causality of the interactions between these systems.

Information pertaining to mass balance, momentum balance, energy balance, as well as structural information is included in this graphical representation. This structural information, which includes the physical interconnection between components and subsystems and their energy causal relations or energy interactions, plays a vital role in system analysis, synthesis and process design. From the bond graph model it is possible to derive a set of state variables easily and, as such, the state equations can be generated systematically [5,6].

Currently, the software packages which have been developed for the purpose of simulating dynamic systems may be divided into three categories. The first set deals with simulating mathematical models of systems and sometimes providing additional tools for the analysis and processing of the results. The information provided to these applications can be in the form of line code and/or graphic form. The second set includes packages which were designed for modelling purposes. Their information is graphical in the form of a bond graph. These packages generate the system's equations from the model and then simulates it. The last set of packages only provide the state equations of a system. Many of these programs have been summarized in the survey paper [7].

The result of this thesis is a complete graphical software package for rapidly modelling physical systems using the bond graph representation. As part of the larger project, to model, analyze and simulate physical systems, this thesis provides the initial implementation of the modelling system. With this system a user may rapidly develop and modify models to be simulated and analyzed for performance evaluation. As well, the user may easily choose from a data base of pre-defined models.

This system will allow engineers in academia and industries to be very efficient and focus more on design and conceptual formulation rather than the grammar of a programming language, data structures, and variables, equation derivations, etc.

1.2 Contents of Thesis

In this thesis a graphical tool to assist in the modelling of physical systems is presented. This tool uses a graphical lumped-parameter model of the energetic interactions between the systems and subsystems being modelled. Specifically, the bond graph representation representation, which provides support, in its generality, for modelling a wide variety of

systems, is used. The modelling tool which is intended to be part of a larger project, to model, simulate, and analyze physical systems, is a complete system within itself.

This thesis is organized as follows. Chapter 2 provides a brief introduction to the bond graph representation of physical systems. An overview of the global structure and general functions of the modelling tool produced is presented in Chapter 3. Chapter 4 discusses specific design details of the modelling tool including data structures and programming techniques used in the implementation. Chapter 5 presents the various screens/parts of the modeler. The conclusions and recommendations for future additions and research are summarized in Chapter 6. The Appendices contains complete source code listings, instructions on program compilation, and an overview of the files included and necessary to run or compile the software.

2 Bond Graphs

2.1 Introduction

A lumped-parameter model describes real physical phenomena by using combinations of idealized fictional elements. This combination of fictional elements may behave closer to reality than any of the individual fictional elements themselves. It is important to remember that although this combination behaves closer to the real physical phenomena, it is still only a fictional model. However, lumped-parameter models help one to visualize the qualitative understanding of a physical phenomenon which may be obscured by systems of equations.

Energy systems assume that no physical system may violate the laws of thermodynamics, in particular the principle of conservation of energy. These systems can be described with a state-determined representation and cover a broad class of systems. For energetic systems, the movement of energy within and across the physical system is tracked. Energy may be stored, added or removed from the system, but never destroyed.

Bond graphs are lumped-parameter models which can be applied to all energetic systems. The bond graph representation was developed at M.I.T. in 1959 by Henry M. Paynter in the Mechanical Engineering Department. Bond graph modelling takes into account energy interactions between systems and subsystems as well as causality of the interactions between these systems and subsystems.

The modeler presented in this thesis uses the bond graph representation to model physical systems. Because bond graphs may be applied to describe all energetic systems, the modeler can be used to design and model a wide range of systems. These systems may be in the mechanical, electrical, magnetic, fluidic, or thermal domains. Non-energetic systems may also be modelled if it is possible to identify a conserved or invariant quantity in the system analogous to energy [8].

2.2 Description of Representation

The basic symbol of the bond graph representation, shown in figure 2.1, is the bond. A bond represents the exchange of power between the two systems, subsystems, or elements at each end of the bond. The power flow between the two systems is represented by two variables, an effort and a flow. In general, the boxes representing the boundary of the system are omitted and only the corresponding symbols are used.

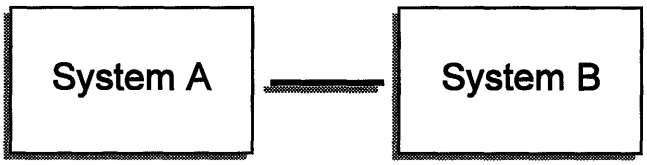


Figure 2.1: A bond representing energetic interaction between two systems.

Because energetic interaction, through a bond, is a function of two variables there are two possibilities when choosing which element to be input and output. Therefore causality is assigned by making a causal stroke at one end of a bond. The system nearest to the causal stroke represents the input into the relationship and the system farthest from the causal stroke represents the output.

In bond graphs, a half-arrow is added to a bond to indicate the direction of positive power flow between the two systems attached to the bond. The combination of causality and power flow can be combined into four possibilities, as shown in figure 2.2. However, not all possibilities may be used with all combinations of systems and is based on the physical model for which the bond graph represents.

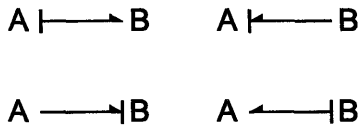


Figure 2.2: Possible causal and power sign assignments.

Although energy cannot be destroyed, it can be dissipated from a system. Energy loss, in the bond graph representation, is represented by a purely dissipative element. This element which represents generalized resistance is depicted with an R.

Figure 2.3: Bond graph symbol for generalized resistance.

The energy dissipated through a generalized resistance element is dependent on the state of the system. However, some elements may add or remove energy to or from a system independent of the state of the system. Thus another bond graph element, the ideal power source, supplies energy to the system independent of the system state. There are two types of ideal sources. An ideal effort source produces an effort independent of the flow from the element and is represented by the symbol S_e. Because an ideal effort source always imposes effort on the rest of the system, causality must be assigned as shown in figure 2.4.

Figure 2.4: Causal assignment for an ideal effort source.

An ideal flow source produces a flow independent of the effort on the element. An ideal flow source, represented by the symbol S_f, must have a causal assignment as shown in figure 2.5 because it always imposes flow on the rest of the system. Ideal sources may either model sources or sinks depending on the direction of power. A sink represents positive power into the element and a source represents positive power out of the element.

Figure 2.5: Causal assignment for an ideal flow source.

To describe the distribution of energy between elements a junction element is used. These elements dictate how other elements, such as generalized resistance and ideal sources, can be assembled. Because the junction element describes the interchange of energy between multiple ports, it is a multi-port element.

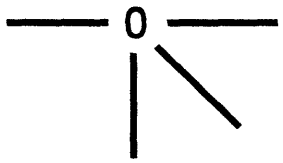


Figure 2.6: Zero Junction.

There are two kinds of junction elements. The zero junction, shown in figure 2.6, has a unique effort associated with all connected bonds. The one junction, shown in figure 2.7, has a unique flow associated with all connected bonds.

Figure 2.7: One Junction

As mentioned previously, energy may be stored in a system. Therefore, there are two elements defined as pure energy-storage elements. The pure generalized capacitor is defined as a phenomenon for which effort is a function of displacement, equation 2.1. Flow is the derivative of displacement, equation 2.2.

$$e = \Psi_C(q) \tag{2.1}$$

$$q = \int f\,dt \tag{2.2}$$

The dual, pure generalized inertance, is defined as a phenomenon for which flow is a function of momentum, equation 2.3. Effort is the derivative of momentum, equation 2.4.

$$f = \Psi_I(p) \tag{2.3}$$

$$p = \int e\,dt \tag{2.4}$$

The generalized capacitor is represented in a bond graph by the symbol C and generalized inertance is represented by the symbol I.

Figure 2.8: Symbols for generalized capacitance and generalized inertance.

There are two asymmetric junction elements. These elements, unlike zero and one junctions, have behavior which is not identical at all ports. The gyrator, represented by the symbol GY, relates efforts on one port to flows on another. The transformer, represented by the symbol TF, relates efforts on one port to efforts on another port.

Because bond graph models describe real physical systems which must obey the laws of thermodynamics, there are some constraints on the possible causality assignments and power directions for the different elements. The gyrator, for instance, can only have the causal assignments shown in figure 2.9 [8].

Figure 2.9: Possible causal assignments for a gyrator.

From a bond graph, systems of equations can be extracted describing the systems behavior. These equations are based on the constitutive laws of the various elements, such as the function of dissipation for a generalized resistance element. These equations are also based on their interconnections through the bonds in the bond graph. For more detailed information on the specific elements of a bond graph and generating the equations for a bond graph see references [5, 6].

2.3 Examples

This section provides a few brief examples to give an idea of the spectrum of systems which may be modelled using bond graphs.

2.3.1 Typical Bond Graph

Equation 2.5 shows the equations derived from the bond graph pictured in figure 2.10.

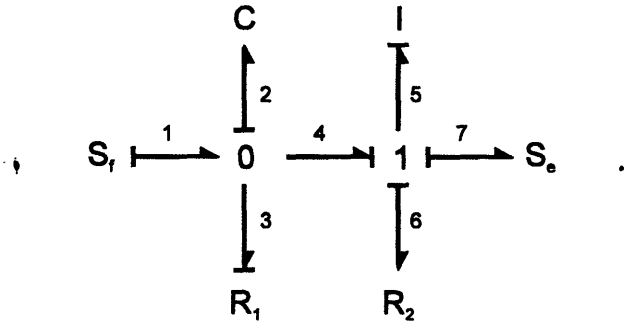


Figure 2.10: A typical bond graph model after naming the bonds and elements in the graph.

$$\begin{bmatrix} e_1 \\ f_2 \\ e_3 \\ e_5 \\ f_6 \\ f_7 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & -1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_1 \\ e_2 \\ f_3 \\ f_5 \\ e_6 \\ e_7 \end{bmatrix} \tag{2.5}$$

2.3.2 Electric Circuit

Figure 2.11 shows an electrical circuit and the accompanying bond graph.

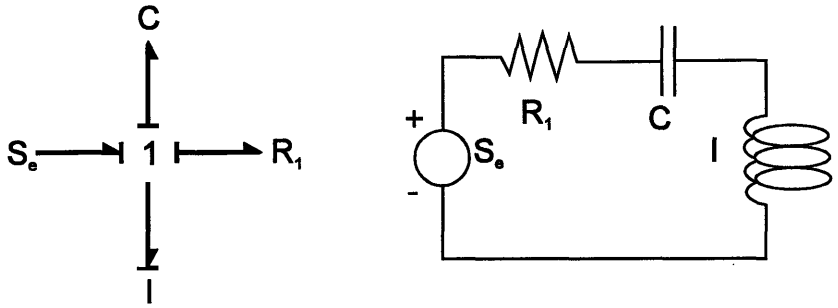


Figure 2.11: Electric circuit and bond graph.

2.3.4 Linear Mechanical System

Figure 2.12 show a simple mechanical system and the accompanying bond graph. In this bond graph, the element symbols are followed by a label to identify its equivalent in the mechanical system's schematic [8].

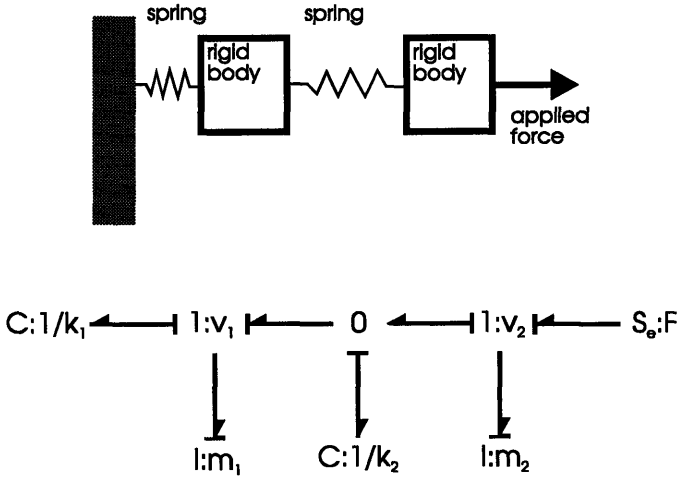


Figure 2.12: Mechanical system schematic and bond graph.

2.4 Summary

Bond graphs form a language for describing energetic interactions in physical systems. Using this language, it is possible to model a broad class of systems. For more information on bond graphs, particularly regarding the formulation of the equations from the models, see [2, 3, 4].

3 Software Overview

3.1 Introduction

3.1.1 Overview

The purpose of this thesis was to develop a rapid prototyping environment which allows a user to perform modelling and design with the bond graph representation. This system, although complete by itself, is only a part of a greater project which will incorporate both analysis and design. The schematic structure of the entire project, including this thesis, is shown in figure 3.1.

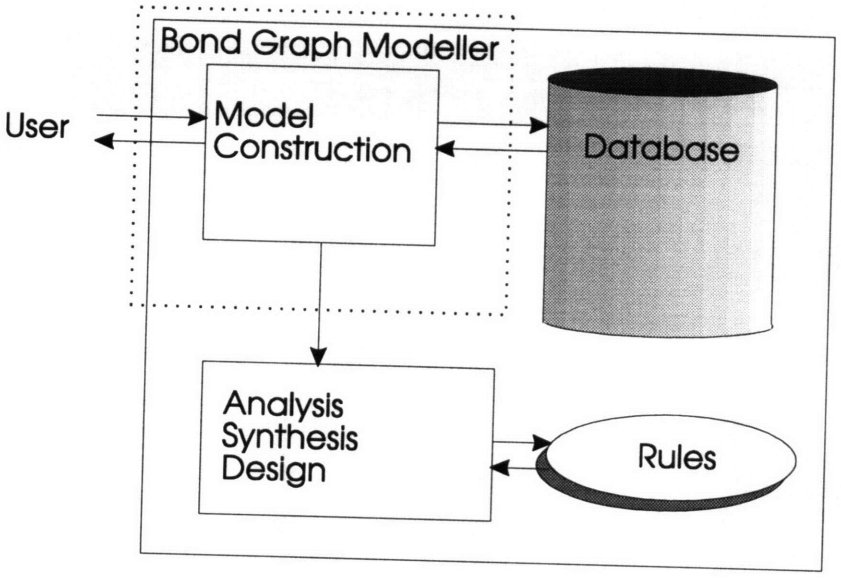


Figure 3.1: The schematic structure of the software under development

The following sections will discuss some of the main features of the bond graph modelling system.

3.1.2 Development Environment

The modeler was developed on a standard 66 Mhz 486-based PC with 20 megabytes of memory. However, the modeler will run on any machine capable of running *Microsoft Windows 3.1* applications. Two megabytes of hard disk space is required to install the application.

Microsoft Windows was used as the system development and run-time platform. *Borland C++* was used as the programming language. *Borland C++* was chosen because it provides *ObjectWindows*, an object oriented class library for creating *Microsoft Windows* programs. The library encapsulates many of the Windows' *Application Programming Interface's* (API) functionality thus shortening the learning and development time to deal with Windows objects.

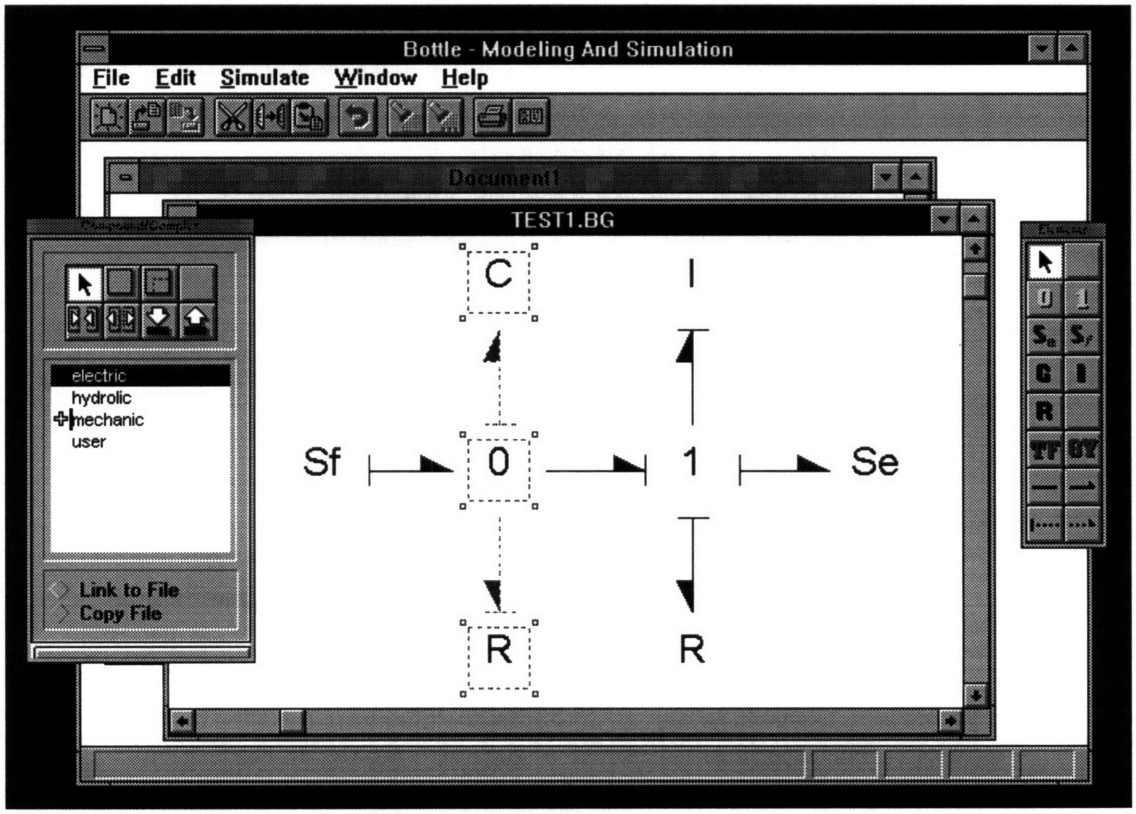


Figure 3.2: Example workspace.

3.2 Global Structure

The modelling system can be divided into the following functionality.

3.2.1 Editor

The editor consists of a set of windows and tools to assist in the rapid construction and modification of bond graph models. Figure 3.2, shows an example workspace with two open windows and two visible palettes.

From the editor the user has access to multiple windows, drawing and construction tools, dialog boxes defining constitutive laws of the elements, file management, print previewing and printing.

3.2.2 Geometric Properties

The modeler allows the various geometric properties of a model to be set and modified. These include element positions, colors, font sizes, etc. A variety of tools are provided to allow the user to rapidly make changes to these properties.

In addition, tools are provided to allow the creation of additional objects such as text, lines, squares, and bitmaps. The user is able to modify the geometric properties of these additional objects, such as font properties, as well.

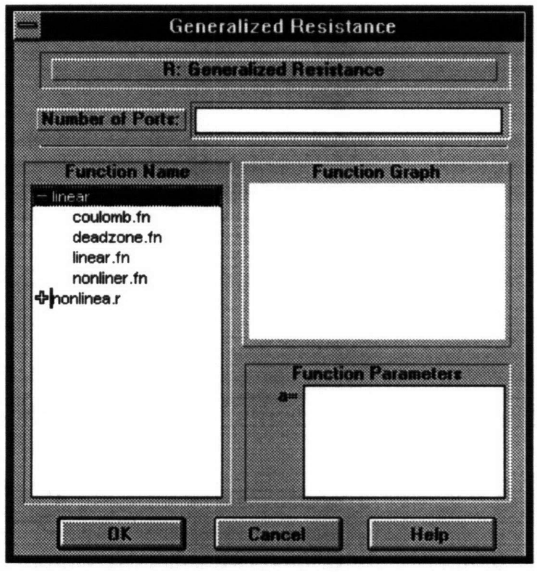


Figure 3.3: Setting the constitutive laws.

3.2.3 Element Properties

Element properties include the constitutive laws of the element. The modeler allows these to be set and modified. These properties may be modified using dialogs similar to the one shown in figure 3.3. The modeler also provides the functionality to easily build a database of functions which may be used when defining an element's behavior.

3.3 General Functions

The modelling system was developed with the following essential features:

3.3.1 Rapid Model Construction

The key feature of this system is that it allows for high efficiency. The user may rapidly develop and modify models to be simulated and analyzed for performance evaluation.

By allowing rapid model construction, this system will allow engineers in academia and industries to be very efficient and focus more on design and conceptual formulation rather than the grammar of a programming language, data structures and variables, equation derivation, etc.

The following features aide in the overall goal of the system to allow rapid model construction.

3.3.2 User Interface

The user interface is most important when trying to realize the main feature and goal of the modeler, to allow rapid model construction. It is important that all of the functionality of the program is easily accessible and the interface is easy to understand and pickup by novice users. The model being constructed should occupy the thoughts of the user, not the interface being worked in.

Microsoft Windows', the platform used, provides a convenient and standard *Graphical User Interface* (GUI). The bond graph modeler provides through this GUI an intuitive interface to produce and edit models in.

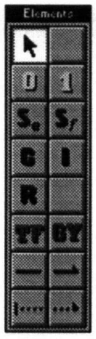


Figure 3.4: Example floating palette.

Floating palettes, as shown in figure 3.4, allow the user quick and easy access to the functionality of the modeler while not getting in the user's way while they are building or modifying a model. Floating palettes can be placed anywhere on the screen and may be hidden at any time. There are four floating palettes, discussed in chapter 5, which group the functionality of the modeler into different subgroups.

By using Windows' Multiple Document Interface, it is possible to edit several models simultaneously. This also provides a convenient method for editing the insides of compound objects without some complicated or tedious procedures.

3.3.3 Compound Objects

One of the features unique to this modeler is the compound object. Users can group a collection of elements into a compound object, thus viewing the functionality of those objects as a whole in a "black box" manner. For instance, a complete model of a car engine would be pretty complex, however this could be grouped into one compound object called "car engine", thus reducing the unnecessary details which can clutter a model and slow the design process. By saving these grouped objects, it is possible to create a library of objects which may be used by others in the construction of their models and thus quicken the entire development process.

The modeler provides the functionality to quickly group and ungroup objects, as well as open a compound object to be modified. Users are also given the option of copying the compound objects definition into their document or linking to an external definition of the object. By linking to an external definition, changes made to the external object will be reflected in the user's document.

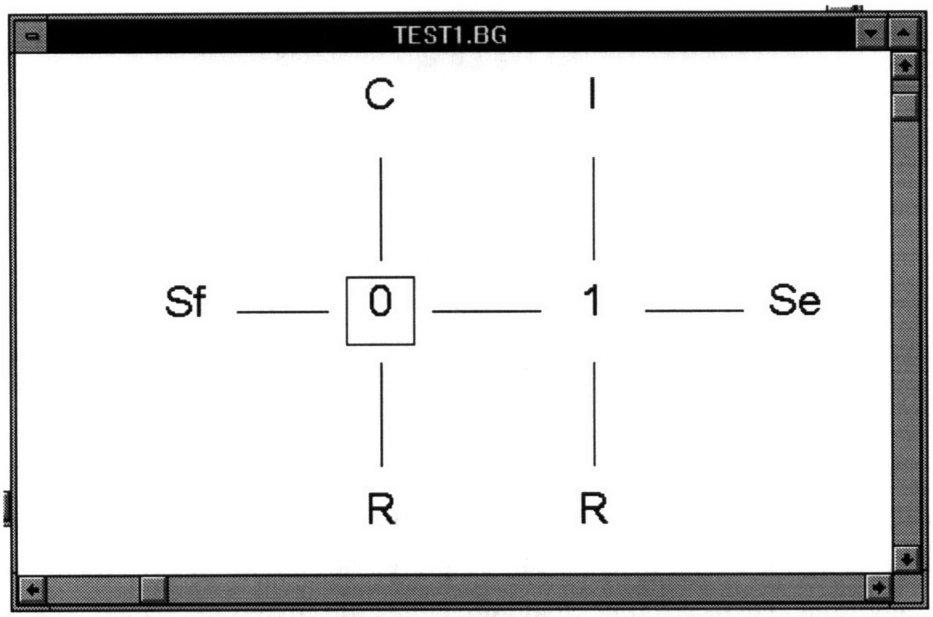


Figure 3.5: Bonds added without causality strokes and power direction arrows.

3.3.4 Causality Strokes and Power Arrows

The modeler allows the user to initially ignore causality and power constraints when adding bonds, as shown in figure 3.5, and thus focus more on the "bigger picture" of the system they are trying to model. By looking past these details it is easier to conceptualize the interactions of the parts of the system.

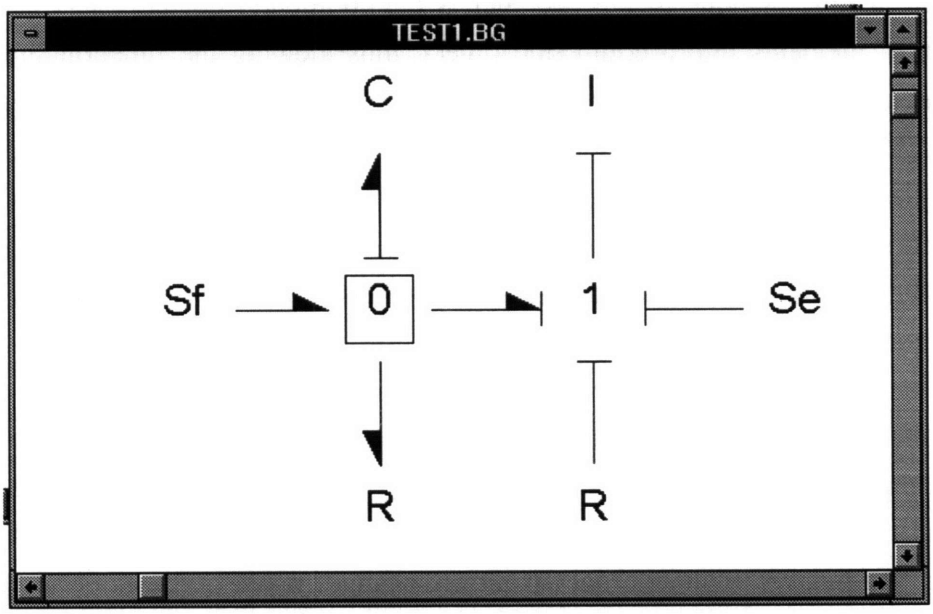


Figure 3.6: Causality and power added after initial bonds were created.

Once the model is laid out, it is then possible to go back and quickly add or modify causality and power direction by clicking on the desired side of the bond. Figure 3.6 shows the previous model after some of the causality and power has been set.

3.3.5 Library of Models and Functions

Through the use of compound objects it is possible to amass a database of reusable models to simplify the design process. As mentioned previously, this can allow the user to design and build systems at a higher level of abstraction. This database can be modified, simply by copying files to a directory, and thus new objects added without recompiling the modelling system.

Similarly, it is possible to build a database of functions to be used when defining the constitutive laws of an element.

3.3.6 Constitutive Laws

As shown in figure 3.3, the constitutive laws governing the functionality of elements may easily be set. In addition, the user is not required to set these properties among creation of the elements. Thus, they are able to initially avoid the details of the system, speeding the design process. Once the elements are in place the user can quickly set the properties through the provided dialog boxes. Making use of the library of functions, described in the previous section, the design process is again simplified by providing a list of standard functions which need not be redefined each time an elements constitutive laws is set.

3.3.7 Other Features

There are additional features which are worthy to mention including the ability to customize the visual properties of a document and to preview and print a document. Users are able to customize, through the attributes palette discussed in chapter 5, the visual properties of most elements. Users are also able to add descriptive labeling, simple drawings, and bitmaps to enhance their documents. Once a model is constructed and labelled the modeler is able to print a document to any supported Windows printer. In addition, users are able to preview the document before it is printed. Appendix A1 provides a sample of the printer output from the modeler.

3.4 Summary

The features of the modeler provide efficient and simple means for the user to construct and modify a model of their system. By providing a simplified intuitive interface, while maintaining extended functionality, users in engineering and academia will be able to focus more on the design and conceptual formulation of their models.

4 Modeler Design

4.1 Introduction

Because, as mentioned earlier, the modelling system will serve as the base for a complete modelling and analysis system, the modeler was designed to be modular and flexible, yet complete.

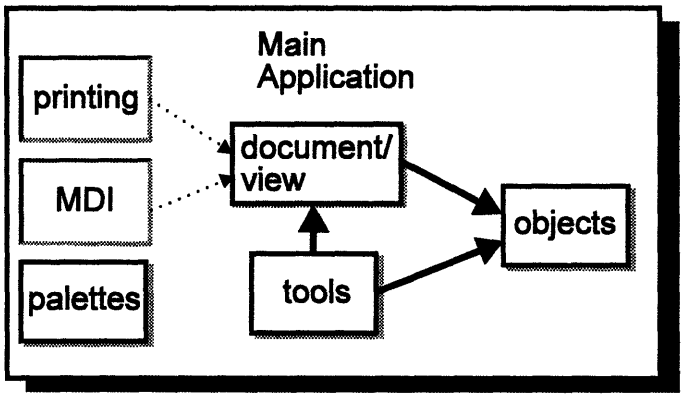


Figure 4.1: Modeler design.

The modeler's design is divided into seven major categories, as shown in figure 4.1. The following sections describe these divisions in more detail. Section 4.2 discusses the Multiple Document Interface (MDI), printing, palettes and main application divisions. The interaction between the remaining categories is covered in section 4.3. Bond graph objects are discussed in section 4.4 and bond graph tools in section 4.5. The document/view structure is covered in section 4.6.

4.2 Main Application, MDI, Palettes and Printing

The main application object handles creation, initialization, and destruction of the application and the other objects. The main window, menu commands, and toolbar are created and controlled by the main application object. In addition, the commands in the

basic elements, shapes, and compound objects palettes are intercepted and dispatched to the corresponding tool by the main application.

The Multiple Document Interface (MDI) objects control creation, destruction, movement and sizing of the child document's window. A view object is indirectly associated with each of the MDI children windows.

The printer object interacts with the view object to print a user's document. Creation and initialization of the structures needed to print and preview a document are handled by the printer object.

The MDI object and printing object require no modifications if additional functionality is added to the program. This is because they interact indirectly with other objects which do the visible work. For instance, if some modification to allow printing only a portion of a bond graph document was added, only the view object needs to be modified. This is because the printer object uses the view to decided how the output should look.

Floating palettes, discussed in chapter 5, provide the user with moveable toolbars which can be placed anywhere on the screen. Some palettes, like the compound objects palette and the attributes palette, have functionality in addition to the simple push buttons used by the shape palette. Therefore, subclasses of the general palette object must be made to handle this additional functionality. However, the simple push buttons in these as well as all push button palettes have their commands intercepted and handled by the main application object.

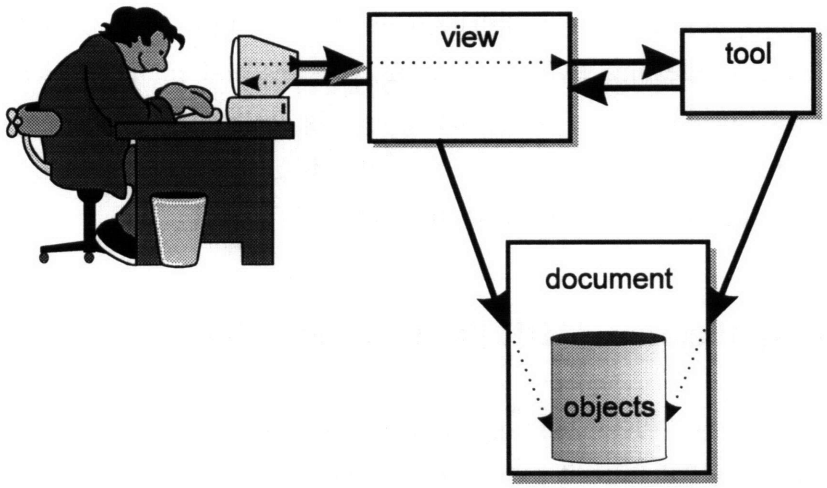


Figure 4.2: Object, tool, document, view interactions.

4.3 Objects, Tools and Document/View Interaction

The four main types which include object, tool, document and view include most of the functionality of the modelling system. Their interactions are shown in figure 4.2. Documents are made up of a collection of objects. Views provide the interface between the user and the document, and as its name suggests, simply provides a view into a document object. Tools are used to add, modify, or delete objects from a document. Tools which are graphical in nature, like a selection tool which allows the user to outline an area, display their work in a view's window.

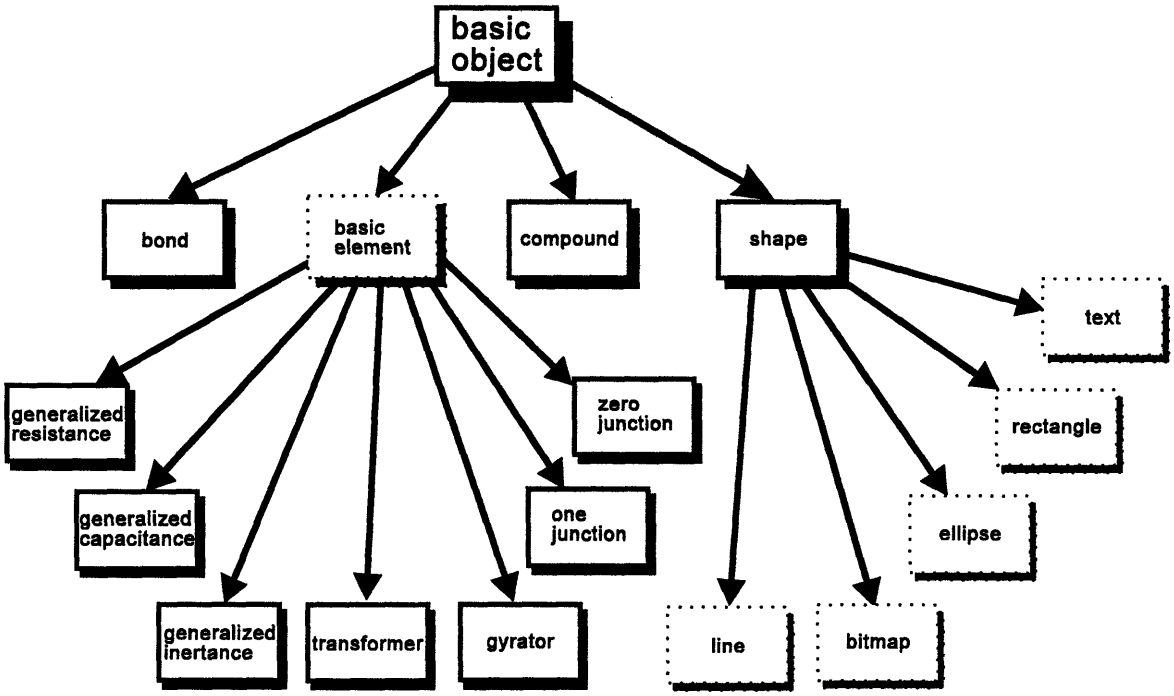


Figure 4.3: Basic object and derivative classes.

4.4 Objects

The basic object class is the smallest element in the model. As mentioned earlier, the document object is made up of objects. The tool objects act on and/or create objects. The view object displays objects either to a window or to a printer object.

The different object classes derived from the basic object are displayed in figure 4.3. The functionality shared by all objects is shown in the code example 4.1.

```
class BasicObject {
public:
      // Input/Output to streams.
      virtual istream& Input(istream&);
      virtual ostream& Output(ostream&);

      // Visual/Display Information
      virtual void MoveTo(int px, int py);
      virtual void MoveBy(int px, int py);

      virtual BOOL XYInObject(int px, int py);
      virtual BOOL ObjectInRect(TRect rect);

      virtual void Draw(TDC*, TRect&, BOOL Selected = FALSE);
      virtual void Paint(TDC*, TRect&);

      virtual void SettingsDialog(TWindow*);

};
```

Example 4.1: Excerpt from the basic object class header.

In the excerpt, the C++ constructor and destructor are left out and only the functions belonging to the basic object are displayed.

The functions Input and Output operate on an input and output stream respectively. These functions provide a means to read documents from and write documents to disk. They also provide a means to read and write the clipboard so that functions such as cut, copy, and paste may be supported. Because each sub-classed object may have different properties it is important to separate out the functionality, such as I/O, which is common to all sub-classed objects.

For some actions, such as selecting a region with the selection tool, it is important to be able to tell if a point is inside and object or if an object is inside a rectangle. The XYInObject and ObjectInRect functions are necessary because different objects may have different shapes. For instance, checking to see if a point is on a line, for the bond object, is much different than checking a point in a rectangle.

Because each object has a very different visual look, the functions Draw and Paint are provided to display an object in a window or to a printer. The SettingsDialog function is provided for those objects, like basic elements and compound objects, for which a user may modify some properties. Not all objects, like text labels and bitmaps, have associated dialog boxes.

4.5 Tools

The tool class either operates on or creates objects by manipulating the view, document and/or object classes. Once a tool is selected various actions by the user in the view object are handled by the tool. Figure 4.4 shows this interaction.

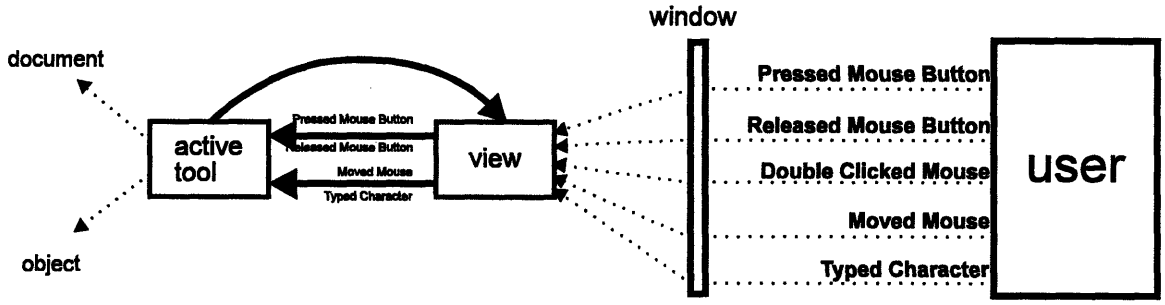


Figure 4.4: Interaction between user, view, and tools.

For example, if the selection tool is active when the user presses the mouse button in the window the tool is notified. The selection tool first checks to see if the mouse is pressed in an existing object by looking through the document object. If it is not then the selection tool deselects all currently selected objects and begins to draw a selection rectangle. As the user moves the mouse the selection tool is notified of each new mouse location and redraws the selection rectangle to match the user's position. When the user releases the mouse button, the selection tool is notified and searches through the document object to find and select all objects who fall within the rectangle defined by the user.

There are a wide variety of tools which are derived from the basic tool object. In addition to tools corresponding to the objects shown in figure 4.3, there is a selection tool. As for objects, sub-classed tools share some common functionality as shown by the code excerpt in example 4.2.

```
class BasicTool {
public:
      // The Tool is notified when it is activated.
      virtual void Selected(TApplication*);
      virtual void DeSelected();

      // Event notification
      virtual BOOL SetCursor(TWindow*, uint);
```

```
        virtual void LeftMouseDown     (TWindow*, uint, TPoint&,
                                        BondGraphDocument&);
        virtual void LeftMouseUp       (TWindow*, uint, TPoint&,
                                        BondGraphDocument&);
        virtual void MouseMove         (TWindow*, uint, TPoint&,
                                        BondGraphDocument&);
        virtual void RightMouseDown    (TWindow*, uint, TPoint&,
                                        BondGraphDocument&);
        virtual void CharTyped         (TWindow*, uint, uint, uint,
                                        BondGraphDocument&);
};
```

Example 4.2: Excerpt from the basic tool class header.

When a tool is activated by selection in a palette or from a menu it is notified by the application, through the function Selected, so it can make any initializations necessary. Similarly, when deselected a tool is notified through the function DeSelected.

As mentioned previously, the tool may react to events caused by the user in the document's view. The function SetCursor is called whenever the cursor needs to be drawn within the window boundaries. This allows the various tools to have cursors which represents the actions the user may engage in. For a text shape tool a caret may be displayed, while a normal pointer would be displayed for the selection tool.

Whenever the mouse button is pressed, released, or the mouse is moved within a window, the active tool is notified of the active document, the mouses position, and any keys on the keyboard which may have been pressed, through the functions LeftMouseDown, RightMouseDown, LeftMouseUp, and MouseMove.

In addition to mouse input, keyboard input is passed to the active tool through the function CharTyped.

4.6 Document/View

The *Document/View* model provided by *Borland C++ ObjectWindows* helps the developer separate the data a window shows from the window itself. Thus two interacting classes, the document class and the view class, are constructed to implement this model.

The document class represents the bond graph document. The document contains a list of the objects currently in the document. The view class provides a virtual window into the document for the user as well as the program.

The document object has functions to add and remove objects from a document. It also handles input, output, and interaction with the Windows' clipboard.

As mentioned earlier, the view class interacts with the tool classes by notifying the currently active tool of events caused by the user. The view class also dictates how a document will be displayed in a window or on a printer.

4.5 Summary

In order to support a modular and flexible design, the main functionality of the modeler was divided into the previously mentioned categories. To add new functionality, new modules may be added or existing ones modified. In addition these modules were further subdivided, like the object class, in order to make addition of new features and modifications to existing ones simpler. Thus, by using an object oriented methodology in designing the modelling application, understanding and modification of the code will be simpler for future programers.

5 Modeler Overview

5.1 Introduction

As mentioned previously, the modeler uses *Windows' Multiple Document Interface* (MDI) to allow users to work in several documents at once. An example workspace with multiple documents open is shown in figure 5.1.

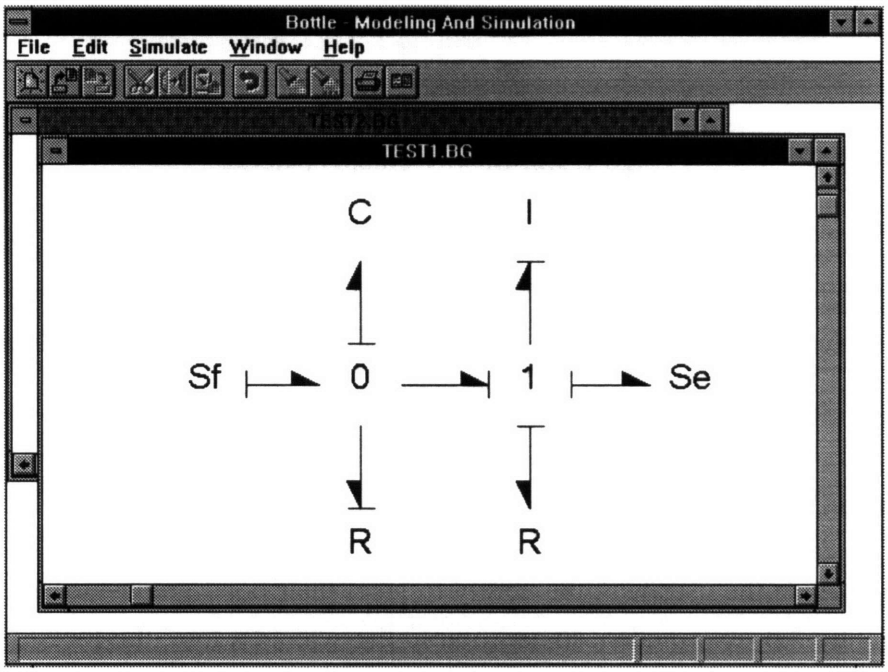


Figure 5.1: Multiple Document Interface (MDI) workspace.

The different functional parts of the program are accessible through floating palettes which can be placed anywhere in the users workspace or hidden from sight. By double clicking on objects in a user's document, it is possible to access the constitutive dialogs which allow the user to modify the physical properties of an element.

The following sections describe the functionality available through each palette and through each dialog.

5.2 Palettes

5.2.1 Basic Elements Palette

The basic elements palette, show in figure 5.2, provides access to the tools to add the basic bond graph elements to the user's document. This palette also provides the tools to set causality and power direction on the bonds in the bond graph.

Figure 5.2: Basic elements palette.

When the last two tools are active, clicking the mouse on a bond causes the causality stroke or power direction arrow to be added to the end of the bond the mouse was clicked closest to. If a causality stroke or power arrow is already on that end of the bond then it is removed.

5.2.2 Shape Palette

In order to provide the user with the ability to label their bond graphs, the shape palette, shown in figure 5.3, is provided. In addition to the selection tool, the tools on the shape palette include the text, line, ellipse, rectangle, and bitmap tools. With these tools the user is not limited to plain bond graphs.

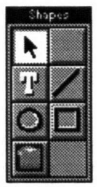


Figure 5.3: Shape palette.

Figure 5.4 shows an example bond graph with and without each of the shape tools used to label it. As in the figure, the line tool can be used for decoration and pointing at elements. The text tool can be used for general labelling. The circle and square tool can be used to outline and group objects for better understanding. The bitmap tool can be used to import graphics created in other applications, such as schematic diagrams and letter heads.

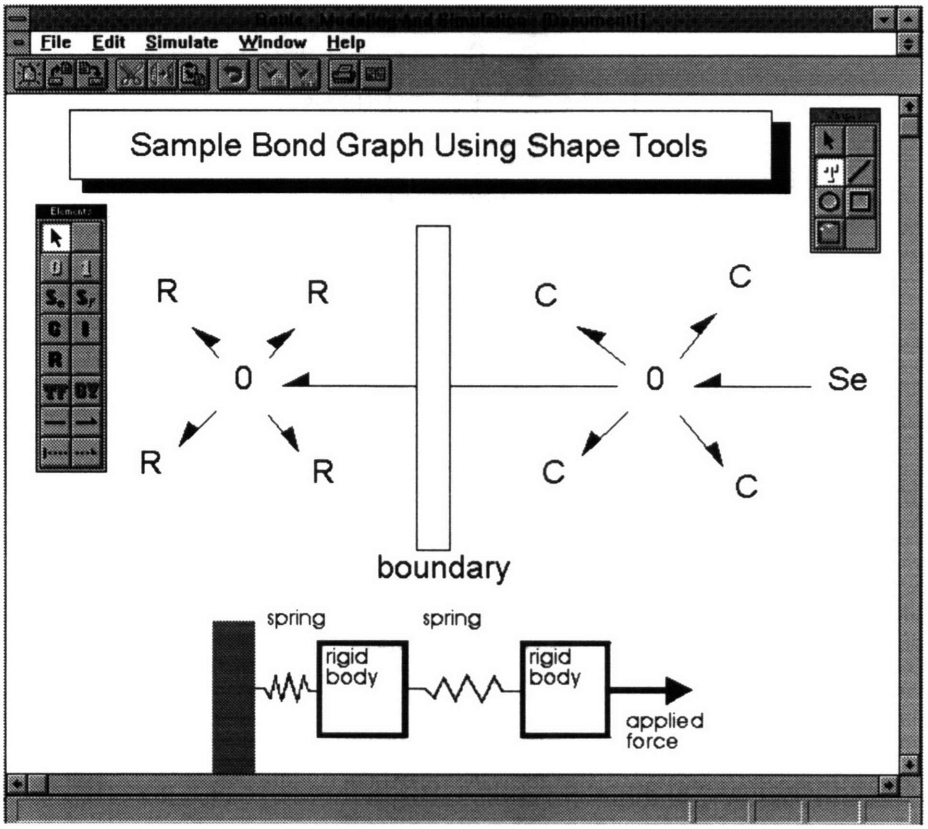


Figure 5.4: Example bond graph using shape objects.

5.2.3 Compound Palette

The compound palette, shown in figure 5.5, provides access to one of the more powerful features of the modelling system. From this palette the user is able to add empty compound objects, predefined compound objects, group existing elements in the bond graph into a compound object, explode an existing compound object, and set which elements in the bond graph are input and/or output elements.

The listbox in the compound palette displays the objects available to be added into the user's document. This list is shown in an outlined form and is based on the files and directories in the models directory. In figure 5.5, the mechanic element in the listbox is a

subcategory of model types. The minus sign before the list element means there are models under this category and can be clicked on to expand or compress the elements in the listbox. To add a new file to the list or a new category or sub-category, the user only has to copy the files into the models directory or make a new subdirectory with the standard *DOS* or *Windows* commands.

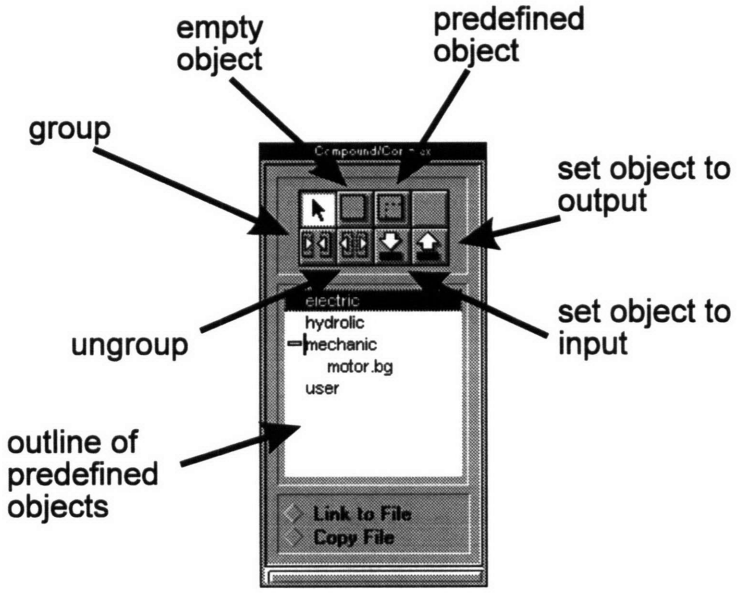


Figure 5.5: Compound palette.

The user can either create an empty object to be filled in later or a predefined object from the list of objects in their document.

When adding predefined objects the user can either copy the object into the file or link to the object. If an object is linked into the user's document then whenever changes are made to the original object's file then the updates will also be visible in the user's document. If an object is not linked but copied, then changes to the objects original file will not be made to the user's document.

5.2.4 Attributes Palette

Through the attributes palette, the user is able to set and modify the visual look of their model. There are five types of sub-palettes accessible through the attributes palette including: grid, alignment, color, display, and font.

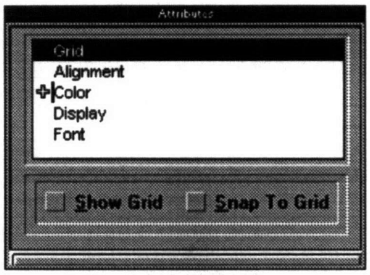


Figure 5.6: Attributes Palette: Grid Settings.

Through the grid sub-palette, shown in figure 5.6, the user can hide or display the grid, shown in figure 5.7, and cause new objects created to be snapped to the grid corners.

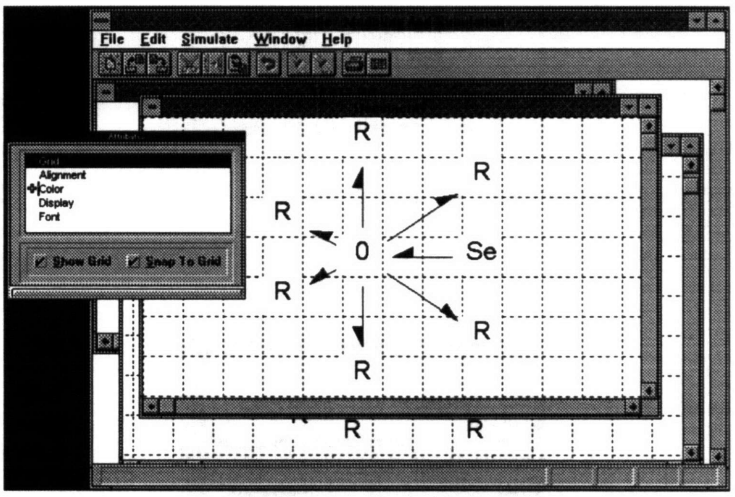


Figure 5.7: Workspace with grid visible.

Objects can be aligned vertically and/or horizontally using the alignment sub-palette, shown in figure 5.8.

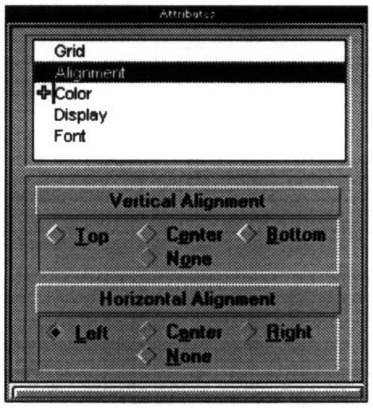


Figure 5.8: Attributes Palette: Alignment Settings.

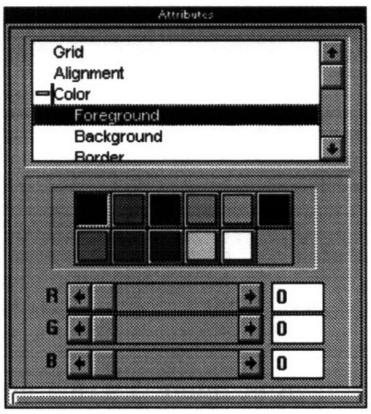


Figure 5.9: Attributes Palette: Color Settings.

Colors for the foreground, background, border, and shadow of an object can be set through the color sub-palette, shown in figure 5.9. The user may either use the colors provided or create their own colors by using the red, green, and blue sliders and/or text fields provided.

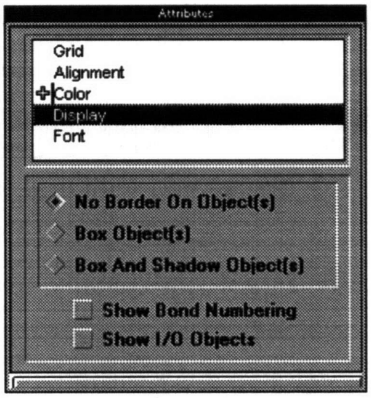


Figure 5.10: Attributes Palette: Display Settings.

In practice, bond graphs are drawn without the boxes defining the border of the elements drawn. However, through the display sub-palette, shown in figure 5.10, it is possible to box and display shadows for objects, including compound objects.

The typeface used in any of the objects may also be modified using the font sub-palette, shown in figure 5.11.

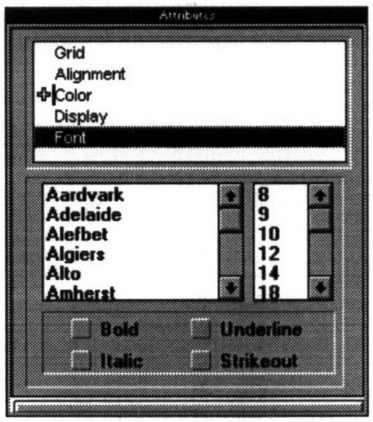


Figure 5.11: Attributes Palette: Font Settings.

5.3 Constitutive Dialogs

As well as adjusting the visual properties of the elements in the bond graph, through the attributes palette, it is possible to modify the physical properties of the elements. The following dialog boxes, which allow access to the physical features of an element, are available by double-clicking on an element or bond with the mouse.

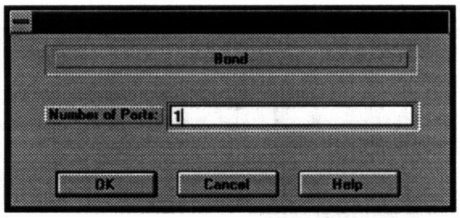


Figure 5.12: Bond dialog.

Through the bond dialog, shown in figure 5.12, it is possible to set the number of ports between the two elements on the end of the bond.

The compound object's dialog, shown in figure 5.13, allows the user to change the label of the object. The user is also able to edit the inside of the compound object by clicking on the edit button. If the object is external and linked in, as shown by the radio buttons in the dialog, then the external file will be loaded into the bond graph editor. If the object is internal then a new window is opened with the contents of that object. When the new window is closed, the changes are made to the internal object.

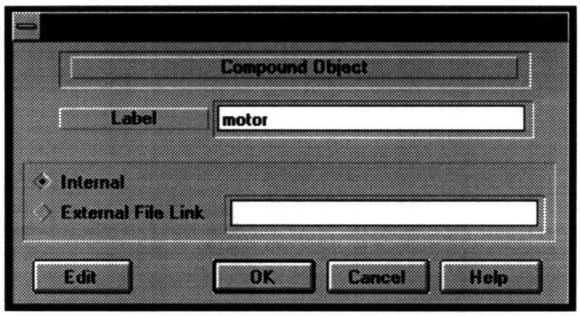


Figure 5.13: Compound object dialog.

Each of the main elements including, resistance, inertance, capacitance, flow source, and effort source, has a dialog similar to the one shown in figure 5.14. Through this dialog it is possible to set the constitutive laws of the element.

In addition to the number of ports on this element, the defining function may be set. An outline of available functions is displayed. Like the compound palette, the functions and categories in the function name list can be modified by the user. To add a new function, its definition simply needs to be added to the function subdirectory. When a function is chosen, the corresponding graph is displayed along with the parameters which may be set or altered by the user.

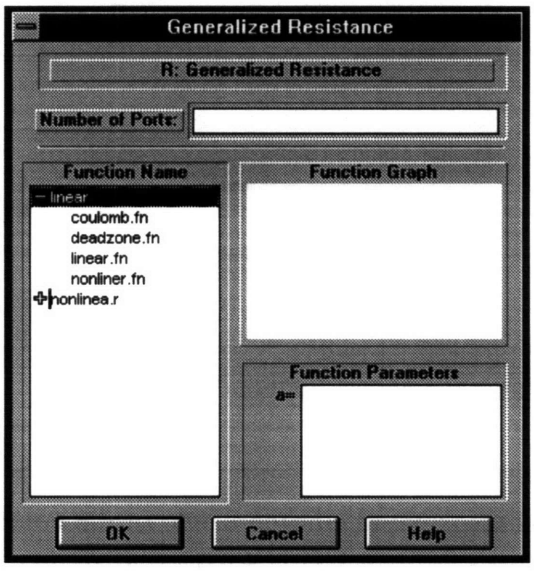


Figure 5.14: Generalized resistance dialog.

5.4 Summary

By separating the functional parts of the modeler into the various palettes, the user is able to easily and quickly have access to the functionality which need at any particular time. Because floating palettes are used, the user is also able to adjust their workspace however they see fit while keeping needed functionality available.

The constitutive dialog for basic elements and the compound object palette allow for easy future modification and addition to the functionality of the program through their list of available functions and objects respectively. Users can add to the database of available models and functions by simply copying a file into the appropriate directory.

6 Conclusions

6.1 Conclusion

This thesis resulted in a complete software package for rapidly modelling physical systems using the bond graph representation. As part of the larger project, to model, simulate and analyze physical systems, this thesis will provide the initial implementation of the modelling system.

The key feature of this system is that it allows for high efficiency. The user may rapidly develop and modify models to be simulated and analyzed for performance evaluation. As well, the user may easily choose from a data base of predefined models.

This system will allow engineers in academia and industries to be very efficient and focus more on design and conceptual formulation rather than the grammar of a programming language, data structures and variables, equation derivation, etc.

6.2 Recommendations

As presented in this thesis, the bond graph modeler is a complete system within itself. However, it was developed as a base from which future versions and systems could be built. There are a wide variety of additions which would increase the usefulness of the software beyond its current capabilities. Possible additions to future versions of the software may include:

- **Help files.** Although the modeler provides an intuitive user interface, the addition of help files would make the software usable to a wider range of users.

- **Database of models.** Currently, the software provides the functionality for making use of a database of models. Once created, it is only a matter of copying

a model to a particular directory in order for it to be available for others to use when developing their models. However, no models currently come with the system and it is necessary for the user to build this library of models from scratch. Now that the software exists, additional models may easily be created for a database and will eventually be available with the software.

- **Integral analysis and simulation.** In order to do analysis with a model created with the modelling tool, the user must export it to be analyzed by a separate software program. Eventually, this software will be incorporated with the modeler so modelling and simulation can be done in one step.

- **Enhanced bond capabilities.** Currently it is not easy to have multiple bonds with opposite causality or power going between the same two systems, as shown in figure 6.1 part a and b. However, it is possible to work around the problem as shown in figure 6.1 part c. Future versions of the software should be able to avoid this problem and necessary work around.

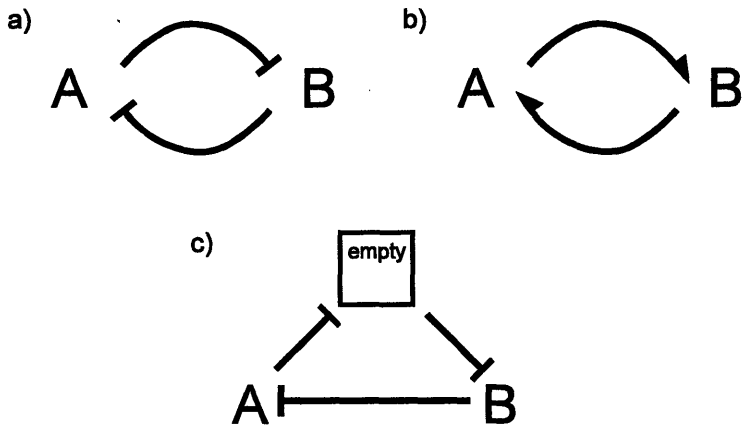


Figure 6.1: Bond problem and work around. a) and b) illustrate the unallowed possibilities. c) illustrates a work around necessary to get the desired description using an empty compound object.

- **Enhanced function capabilities.** The functionality behind the functions which define the constitutional laws of the elements is limited. There are many enhancements and additions which can and should be made here.

- **Real-time constraint warnings.** There are rules dictating which connections, causal strokes, and power arrows are viable in a legal bond graph. A useful

addition to the modeler would be warnings while a bond graph is being constructed, either by a message or simply highlighting the invalid constraints.

- **Automated causality placement.** Because there are rules which are known for assigning causality it is possible for the software to do this while a model is being constructed or after it has initially been constructed. This would reduce the amount of work necessary for the user to complete. It would also free the user's mind to more easily deal with the conceptual formulation of the model.

There are many more possibilities for enhancements to future versions of the modelling software. It is important to remember that the modeler, as presented in this thesis, is meant to serve as a base from which a more powerful modeler may be built and a complete modelling, simulation, and analysis system may be constructed.

Appendix

A.1 Sample Printer Output

The following page contains sample printer output, produced by the bond graph modeler, for the model shown in figure A1.1.

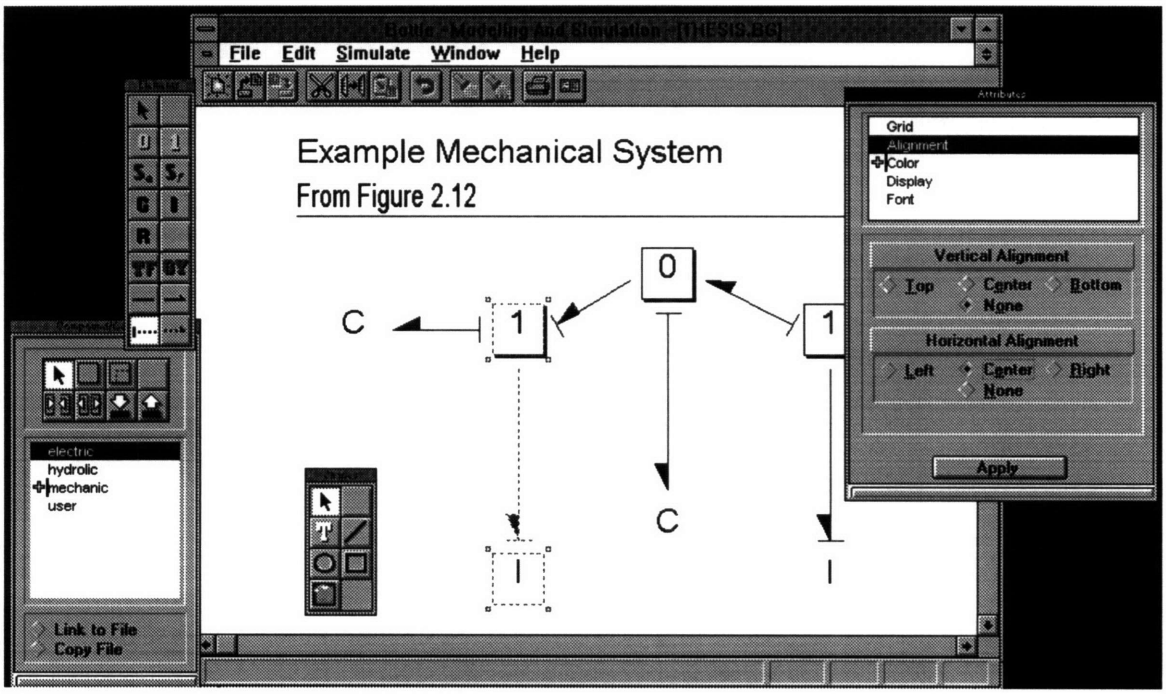
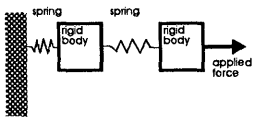
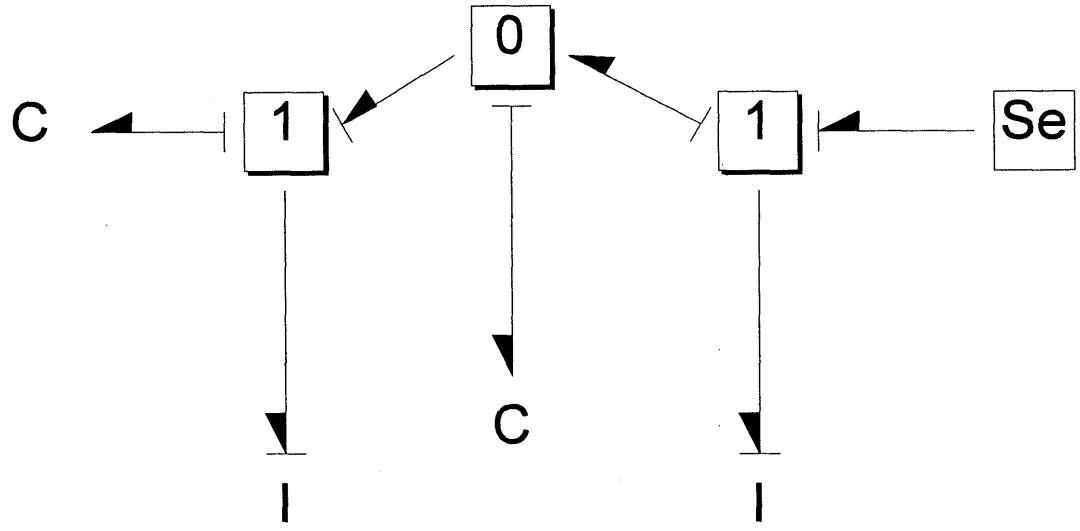


Figure A1.1: Model being created.

Example Mechanical System
From Figure 2.12





A.2 File Overview

The program source code files and contents are as follows:

Filename	Contents
Objects	
bttlobj.h	Header containing definition of basic object class.
bttlobj.cpp	Source for above.
obj_elem.h	Header for basic element objects class.
obj_elem.cpp	Source for above.
obj_comp.h	Header for complex object classes, like compound object.
obj_comp.cpp	Source for above.
obj_shap.h	Header for shape object classes.
obj_shap.cc	Source for above.
Tools	
bttltool.h	Header containing definition of basic tool class.
bttltool.cpp	Source for above.
toolcomp.h	Header for complex tools, like compound tool.
toolcomp.cpp	Source for above.
toolelem.h	Header for basic element tools.
toolelem.cpp	Source for above.
toolslct.h	Header for selection tool.
toolslct.cpp	Source for above.
toolshap.h	Header for shape tools.
toolshap.cpp	Source for above.
bttltlst.h	Header for tool list.
bttltlst.cpp	Source for above.
General	
msoutlin.h	Header for Microsoft Outline VBX control.
koutlin.h	Header for source to manipulate outline tool.
koutlin.cpp	Source for above
bttlbtn.h	Header for bitmapped button.
bttlbtn.cpp	Source for above
Palettes	
bttlpal.h	Header for basic palette class.
bttlpal.cpp	Source for above.
palettes.h	Header for palettes.
palettes.cpp	Source for above.

Filename	Contents
Document/View	
bttldoc.h	Header for document class.
bttldoc.cpp	Source for above.
bttlview.h	Header for view class.
bttlview.cpp	Source for above.
Printing	
apxprev.h	Header for print preview objects.
apxprev.cpp	Source for above.
apxprint.h	Header for pint objects.
apxprint.cpp	Source for above.
Main Application	
bttlapp.h	Header for main application class.
bttlapp.cpp	Source for above and main function.
bttlabtd.h	Header for about dialog class.
bttlabtd.cpp	Source for above.
bttlmdic.h	Header for MDI children class.
bttlmdic.cpp	Source for above.
bttlmdi.h	Header for MDI manager class.
bttlmdi.cpp	Source for above.
Resources	
bttlapp.rh	Header for resource ids.
bttlapp.rc	Resource file for application.
bttlapp.def	Definition file for application.

A.3 Program Compilation

The bond graph modeller program code consists of the files listed in Appendix A.2. It was coded in Borland C++ for Windows v4.5 using Borland ObjectWindows and Resource Workshop. The program has successfully been compiled on two 486 DX2 66 Mhz machines, one with 8 megabytes of ram and the other with 20 megabytes of ram.

It is suggested that dynamic linking be used to reduce the final size of the executable. This reduces the size of the code but adds the need to ship some Windows DLLs with the application. Thus, the following files should accompany the final executable version of the application:

bottle.exe	The bond graph modeller.
bwcc.dll	Borland DLL.
msoutlin.vbx	VBX control for Microsoft Outline.
worklib1.dll	Resource Workshop DLL.
worklib2.dll	Resource Workshop DLL.

The project file bottle.ide contains links to all the source and has all the settings necessary to compile the program. The only change which may need to be made is the setting of the borland file paths.

So, to compile the program:

1) Load the bottle.ide project file.

2) Update the directories in the project settings to reflect the path of Borland C++ on the machine the program is being compiled on.

3) Make the file.

A.4 Source Code

This appendix contains the complete source code listing for the bond graph modeller. The files in this listing include those mentioned in Appendix A.2.

bttlobj.h

```
/* ================================================ */
/* BasicObject Class                                */
/* ------------------------------------------------ */
/*                                                  */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*               Copyright (c) 1995                 */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

/* ------------------------------------------------ */
/* The basic object holds information which is      */
/* common and available to all objects.             */
/*                                                  */
/* Whenever a new object type is added, a unique    */
/* identifier should be added to the list below.    */
/* ------------------------------------------------ */

// Use file only if it's not already included.
#if !defined(__bttlobj_h)
#define __bttlobj_h

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include <owl\owlpch.h>           // Increases size of code, but speeds compilation
#include <classlib\arrays.h>
#pragma hdrstop

/* ================================================ */
/* Defines for Object Types                         */
/* ------------------------------------------------ */
/* Basic Object */
#define BG_BASIC          0

/* Basic Elements */
#define BG_BOND           1     // Bond
#define BG_0JUNCTION      2     // Zero Junction
#define BG_1JUNCTION      3     // One Junction
#define BG_ESOURCE        4     // Effort Source
#define BG_FSOURCE        5     // Flow Source
#define BG_RESISTOR       6     // Generalized Resistance
#define BG_CAPACITOR      7     // Generalized Capacitance
#define BG_INDUCTOR       8     // Generalized Inductance
#define BG_TRANSFORMER    9     // Transformer
#define BG_GYRATOR        10    // Gyrator

/* Complex Objects */
#define BG_COMPOUND       11
#define BG_OLE                13       // Possible Addition
#define BG_FILELINK       14       // Possible Addition

/* Information/Formatting Objects*/
#define BG_SHAPE          12
#define BG_DOCUMENT       15       // Possible Addition

/* Display Modes */
#define DISPLAY_NOBORDER  0
#define DISPLAY_BOX           1
#define DISPLAY_SHADOW        2

/* Line Styles and Thickness */
#define LINE_DEFAULT          255 // Possible Addition

/* For Selections */
#define DrawMiniBox(dc,sx,sy) dc->Rectangle(TPoint((sx),(sy)),TSize(4,4))

/* ================================================ */
/* Class Support Type Definitions                   */
/* ------------------------------------------------ */
#define R    0
#define G    1
#define B    2
typedef char Color[3];

/* ================================================ */
/* Basic Object Class Definition                    */
/* ------------------------------------------------ */
class BasicObject {
public:
```

```
    // Attributes ==================================
    Color    foreground;
    Color background;
    Color shadow;
    Color border;

    char  shadowDepth;
    char  displayMode;

    char  lineThickness;
    char  lineStyle;

    char    *label;
    char     labelFont[13];   // Parameters for CreateFont
    char    *labelFontName;   // Paremeter for CreateFont
    TBitmap *image;

    char *attachedText;   // Possible Addition

    bool ioInput;
    bool ioOutput;

    // ---------------------------------------------
    char typeId; // One of the previously listed types
    int  distinctId; // For linking objects and bonds.

    int  x,y;
    int  w,h;
    BOOL selected;

    // ---------------------------------------------
    // Really a TObjectArray* defined later.
    TArray<BasicObject *> *subobjects;    // Children, etc.
    TArray<BasicObject *> *superobjects;  // Parents, bonds, etc...

    // =============================================
public:
    BasicObject();                              // Basic Constructor
    BasicObject(const BasicObject&);  // Copy Constructor
    ~BasicObject();                  // Destructor

    // Input/Output to streams.
    virtual istream& Input(istream&);
    virtual ostream& Output(ostream&);

    // Visual/Display Information
    virtual void MoveTo(int px, int py);
    virtual void MoveBy(int px=0, int py=0);

    virtual BOOL XYInObject(int px, int py);
    virtual BOOL ObjectInRect(TRect rect);

    virtual void Draw(TDC*, TRect& rect, BOOL Selected = false, float
            Scale = 1.0);
    virtual void Paint(TDC*, TRect&, float);

    virtual void SettingsDialog(TWindow*);
};

/* ================================================ */
/* More Class Support Type Definitions              */
/* ------------------------------------------------ */
typedef TArray<BasicObject *> TObjectArray;
typedef TArrayAsVectorIterator<BasicObject *> TObjectArrayIterator;

/* ================================================ */
/* Class Support Functions                          */
/* ------------------------------------------------ */
BasicObject* NewObject(uint); // Returns a new object with the given identifier.

istream& operator >>(istream&, BasicObject&);  // Read object from a stream.
ostream& operator <<(ostream&, BasicObject&);  // Write object to a stream.

void ColorBlack (Color);
void ColorWhite (Color);
void ColorCopy  (Color, const Color);

// Produces distinct IDs for different objects.
int SetDistinctId(BasicObject&);

#endif
```

bttlobj.cpp

```
/* ================================================ */
/* BasicObject Class                                */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include <windows.h>
#pragma hdrstop

#include "bttlobj.h"

#include "obj_elem.h"
#include "obj_comp.h"
#include "obj_shap.h"

#include <stdlib.h>

/* ================================================ */
/* Class Support Function Definitions               */
/* ------------------------------------------------ */

/* ================================================ */
/* SetDistinctId                                    */
/* ------------------------------------------------ */
/* Gives an object a distinct id if it doesn't have */
/* one yet.                                         */
/*                                                  */
/* input:     BasicObject& object                   */
/* output:    int                                   */
/* ================================================ */
int SetDistinctId (BasicObject& object)
{
  static int differentId = 0;
  if (object.distinctId > 0)
     return (object.distinctId);
  return (object.distinctId = ++differentId);
}

/* ================================================ */
/* ColorBlack                                       */
/* ------------------------------------------------ */
/* Sets the values of color to that for black.      */
/*                                                  */
/* input:     Color color                           */
/* ================================================ */
void ColorBlack (Color color)
{ color[R] = 0; color[G] = 0; color[B] = 0;
}

/* ================================================ */
/* ColorWhite                                       */
/* ------------------------------------------------ */
/* Sets the values of color to that for white.      */
/*                                                  */
/* input:     Color color                           */
/* ================================================ */
void ColorWhite (Color color)
{ color[R] = 255; color[G] = 255; color[B] = 255;
}

/* ================================================ */
/* ColorCopy                                        */
/* ------------------------------------------------ */
/* Copys the color b into a. a = b.                 */
/*                                                  */
/* input:     Color a, b                            */
/* ================================================ */
void ColorCopy (Color a, const Color b)
{ a[R] = b[R]; a[G] = b[G]; a[B] = b[B];
}

/* ================================================ */
/* operator >>                                      */
/* ------------------------------------------------ */
/* reads the next object from the input stream.     */
/*                                                  */
/* input:     istream&     is                       */
/*            BasicObject& object                   */
/* output:    istream&                              */
```

```
/* ================================================= */
istream& operator >>(istream& is, BasicObject& object)
{   return object.Input(is);
}

/* ================================================= */
/* operator <<                                       */
/* ------------------------------------------------- */
/* writes the object onto the output stream.         */
/*                                                   */
/* input:     ostream&     os                        */
/*            BasicObject& object                    */
/* output:    ostream&                               */
/* ================================================= */
ostream& operator <<(ostream& os, BasicObject& object)
{   return object.Output(os);
}

/* ================================================= */
/* NewObject                                         */
/* ------------------------------------------------- */
/* creates a new object of type object_type_id and   */
/* returns it.                                       */
/*                                                   */
/* input:     uint object_type_id                    */
/* output:    BasicObject*                           */
/* ================================================= */
BasicObject* NewObject(uint object_type_id)
{
    BasicObject* bo = NULL;

    switch (object_type_id) {
        case BG_BASIC:       bo = new BasicObject();       break;
        case BG_BOND:        bo = new ObjectBond();        break;
        case BG_0JUNCTION:   bo = new Object0Junction();   break;
        case BG_1JUNCTION:   bo = new Object1Junction();   break;
        case BG_ESOURCE:     bo = new ObjectESource();     break;
        case BG_FSOURCE:     bo = new ObjectFSource();     break;
        case BG_RESISTOR:    bo = new ObjectResistor();    break;
        case BG_CAPACITOR:   bo = new ObjectCapacitor();   break;
        case BG_INDUCTOR:    bo = new ObjectInductor();    break;
        case BG_TRANSFORMER: bo = new ObjectTransformer(); break;
        case BG_GYRATOR:     bo = new ObjectGyrator();     break;
        case BG_COMPOUND:    bo = new ObjectCompound();    break;
        case BG_SHAPE:       bo = new ObjectShape();       break;
    }

    return bo;
}

/* ================================================= */
/* Class Constructors                                */
/* ------------------------------------------------- */

/* ================================================= */
/* BasicObject                                       */
/* ------------------------------------------------- */
/* creates a new basic object.                       */
/*                                                   */
/* output:    BasicObject                            */
/* ================================================= */
BasicObject::BasicObject()
{
    ColorBlack(foreground);
    ColorWhite(background);
    ColorBlack(shadow);
    ColorBlack(border);

    shadowDepth = 5;
    displayMode = DISPLAY_NOBORDER;

    lineThickness = LINE_DEFAULT;
    lineStyle = LINE_DEFAULT;

    label = NULL;
    image = NULL;

    attachedText = NULL;

    typeId = BG_BASIC;
    selected = FALSE;

    subobjects = NULL;
    superobjects = new TObjectArray(1,0,5);

    // Create Font Parameters
    labelFont[0]  = 30;
```

```
    labelFont[1]  = 0;
    labelFont[2]  = 0;
    labelFont[3]  = 0;
    labelFont[4]  = FW_BOLD;
    labelFont[5]  = 0;
    labelFont[6]  = 0;
    labelFont[7]  = 0;
    labelFont[8]  = ANSI_CHARSET;
    labelFont[9]  = OUT_DEFAULT_PRECIS;
    labelFont[10] = CLIP_DEFAULT_PRECIS;
    labelFont[11] = PROOF_QUALITY;
    labelFont[12] = VARIABLE_PITCH;
    labelFontName = (char *) malloc ((strlen("Arial")+1)*sizeof(char));
    strcpy(labelFontName,"Arial");

    distinctId = -1;
    ioInput = ioOutput = FALSE;
}

/* ================================================ */
/* BasicObject                                      */
/* ------------------------------------------------ */
/* Copy constructor. Creates a new basic object     */
/* which is a copy of object.                       */
/*                                                  */
/* input:  const BasicObject& object                */
/* output:       BasicObject                        */
/* ================================================ */
BasicObject::BasicObject(const BasicObject& object)
{
    ColorCopy(foreground,object.foreground);
    ColorCopy(background,object.background);
    ColorCopy(shadow,object.shadow);
    ColorCopy(border,object.border);

    shadowDepth = object.shadowDepth;
    displayMode = object.displayMode;

    lineThickness = object.lineThickness;
    lineStyle = object.lineStyle;

    free (label);
    label = (char *) calloc (strlen(object.label)+1,sizeof(char));
    strcpy(label, object.label);

    // Need To Change
    image = object.image;

    free (attachedText);
    attachedText = (char *) calloc (strlen(object.attachedText)+1,sizeof(char));
    strcpy(attachedText, object.attachedText);

    typeId = object.typeId;
    selected = object.selected;

    // Copy Parents and Children
    delete subobjects;
    if (object.subobjects == NULL)
      subobjects = NULL;
    else {
      subobjects = new TObjectArray(object.subobjects->GetItemsInContainer(),0,10);
      TObjectArrayIterator ObjectIterator(*object.subobjects);
      for (;ObjectIterator.Current();ObjectIterator++)
         subobjects->Add(ObjectIterator.Current());
    }

    delete superobjects;
    if (object.superobjects == NULL)
      superobjects = NULL;
    else {
      superobjects = new TObjectArray(object.superobjects->GetItemsInContainer(),0,10);
      TObjectArrayIterator ObjectIterator2(*object.superobjects);
      for (;ObjectIterator2.Current();ObjectIterator2++)
         superobjects->Add(ObjectIterator2.Current());
    }

    // Create Font Parameters
    for (int i=0;i<13;i++)
      labelFont[i] = object.labelFont[i];
    free(labelFontName);
    labelFontName = (char *) malloc ((strlen(object.labelFontName)+1)*sizeof(char));
    strcpy(labelFontName,object.labelFontName);

    distinctId = object.distinctId;
}

/* ================================================ */
```

```
/* ~BasicObject                                          */
/* ----------------------------------------------------- */
/* frees memory used by the BasicObject                  */
/*                                                       */
/* ===================================================== */
BasicObject::~BasicObject()
{
    free(label);
    free(labelFontName);
    free(attachedText);
    free(superobjects);
    free(subobjects);
}

/* ===================================================== */
/* Input                                                 */
/* ----------------------------------------------------- */
/* Reads the object from the input stream is.            */
/*                                                       */
/* Input Format:                                         */
/*   ,x,y,w,h,foreground,background,shadow,border,       */
/*   shadowDepth,displayMode,lineThickness,lineStyle,    */
/*   distinctId)                                         */
/*                                                       */
/* input:  istream& is                                   */
/* output: istream&                                      */
/* ===================================================== */
istream& BasicObject::Input(istream& is)
{
    char c;
    int r,g,b;

    is >> c >> x >> c >> y;
    is >> c >> w >> c >> h;

    is >> c >> r >> c >> g >> c >> b;
    foreground[R] = r; foreground[G] = g; foreground[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    background[R] = r; background[G] = g; background[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    shadow[R] = r;     shadow[G] = g;     shadow[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    border[R] = r;     border[G] = g;     border[B] = b;

    is >> c >> r >> c >> g;
    shadowDepth = r;
    displayMode = g;

    is >> c >> r >> c >> g;
    lineThickness = r;
    lineStyle = g;

    is >> c >> distinctId;
    is >> c;

    return is;
}

/* ===================================================== */
/* Output                                                */
/* ----------------------------------------------------- */
/* Writes the object to the Output stream os.            */
/*                                                       */
/* Output Format:                                        */
/*   (OBJECT_TYPE_ID,                                    */
/*   ,x,y,w,h,foreground,background,shadow,border,       */
/*   shadowDepth,displayMode,lineThickness,lineStyle,    */
/*   distinctId)                                         */
/*                                                       */
/* input:  ostream& os                                   */
/* output: ostream&                                      */
/* ===================================================== */
ostream& BasicObject::Output(ostream& os)
{
    char c = ',';

    os << '(' << (int) typeId;
    os << c << x << c << y;
    os << c << w << c << h;
    os << c << (int) foreground[R] << c << (int) foreground[G] << c << (int) foreground[B];
    os << c << (int) background[R] << c << (int) background[G] << c << (int) background[B];
    os << c << (int) shadow[R]     << c << (int) shadow[G]     << c << (int) shadow[B];
    os << c << (int) border[R]     << c << (int) border[G]     << c << (int) border[B];
    os << c << (int) shadowDepth   << c << (int) displayMode;
```

```
    os << c << (int) lineThickness << c << (int) lineStyle;
    os << c << SetDistinctId(*this);
    os << ')';

    return os;
}

/* ================================================ */
/* MoveTo                                           */
/* ------------------------------------------------ */
/* Moves the object to px, py.                      */
/*                                                  */
/* input:  int px, py                               */
/* ================================================ */
void BasicObject::MoveTo(int px, int py)
{   x = px;  y = py;
}

/* ================================================ */
/* MoveBy                                           */
/* ------------------------------------------------ */
/* Relative move. Move object over by px, py pixels. */
/*                                                  */
/* input:  int px, py                               */
/* ================================================ */
void BasicObject::MoveBy(int px, int py)
{   x += px; y += py;
}

/* ================================================ */
/* XYInObject                                       */
/* ------------------------------------------------ */
/* Tells whether the point px, py is in the object.  */
/*                                                  */
/* input:  int px, py                               */
/* output: BOOL                                     */
/* ================================================ */
BOOL BasicObject::XYInObject(int px, int py)
{
    if (x <= px && px <= x+w &&
         y <= py && py <= y+h)
        return (TRUE);
    return (FALSE);
}

/* ================================================ */
/* ObjectInRect                                     */
/* ------------------------------------------------ */
/* Tells whether the object is in the rectangle rect.*/
/*                                                  */
/* input:  TRect rect                               */
/* output: BOOL                                     */
/* ================================================ */
BOOL BasicObject::ObjectInRect(TRect rect)
{
    TRect objRect(x,y,x+w,y+h);
    return (rect.Contains(objRect));
}

/* ================================================ */
/* Draw                                             */
/* ------------------------------------------------ */
/* Draws the object to the screen dc.               */
/*                                                  */
/* input: TDC*   dc                                 */
/*        TRect&                                    */
/*        BOOL   drawSelected                       */
/*        float                                     */
/* ================================================ */
void DrawIOArrows(TDC* dc, int x, int y, int w)
{
  TPoint points[3];
  dc->SelectObject(TBrush(TColor::Black));
  for (int cx=x; cx<x+w; cx+=8) {
     points[0].x = cx;   points[0].y = y;
     points[1].x = cx+2; points[1].y = y - 4;
     points[2].x = cx+4; points[2].y = y;
     dc->Polygon(points,3);
  }
  dc->RestoreBrush();
}

void BasicObject::Draw(TDC* dc, TRect&, BOOL drawSelected, float )
{
    // Draw Shadow
    if (displayMode == DISPLAY_SHADOW) {
      dc->SelectObject(TBrush(TColor(shadow[R],shadow[G],shadow[B])));
```

```
      dc->Rectangle(TRect(TPoint(x+shadowDepth,y+shadowDepth),TSize(w,h)));
    }

    // Draw Input / Output Lines
    if (ioInput)
      DrawIOArrows(dc,x,y+h+4+2,w);

    if (ioOutput)
      DrawIOArrows(dc,x,y-2,w);

    // Draw Border and Background
    if (drawSelected)
          dc->SelectObject(TPen((TColor::LtRed), 0, PS_DOT));
    else if (displayMode == DISPLAY_NOBORDER)
          dc->SelectObject(TPen(TColor::Black, 0, PS_NULL));
        else
          dc->SelectObject(TPen(TColor(border[R],border[G],border[B]), 0, PS_SOLID));
    dc->SelectObject(TBrush(TColor(background[R],background[G],background[B])));

    dc->Rectangle(TRect(TPoint(x,y),TSize(w,h)));

    // Draw Label
    if (label) {
      HFONT hfont = CreateFont(labelFont[0],labelFont[1],labelFont[2],labelFont[3],labelFont[4],

    labelFont[5],labelFont[6],labelFont[7],labelFont[8],labelFont[9],
                                           labelFont[10],labelFont[11],labelFont[12],labelFontName);
      TFont *theFont = new TFont(hfont);

      dc->SetTextColor(TColor(foreground[R],foreground[G],foreground[B]));
      dc->SelectObject(*theFont);
      dc->SetBkMode(TRANSPARENT);
      dc->DrawText(label,-1,TRect(TPoint(x,y),TSize(w,h)),DT_CENTER | DT_VCENTER);
      dc->RestoreFont();

      delete theFont;
      DeleteObject(hfont);
    }

    // if selected draw mini boxes
    if (drawSelected) {
        dc->SelectObject(TPen(TColor::LtRed, 0, PS_SOLID));
        DrawMiniBox(dc,x-5,y-5);
        DrawMiniBox(dc,x+w+1,y-5);
        DrawMiniBox(dc,x+w+1,y+h+1);
        DrawMiniBox(dc,x-5,y+h+1);
    }

    dc->RestorePen();
}

/* ================================================= */
/* Paint                                             */
/* ------------------------------------------------- */
/* Paints the object to the printer dc.              */
/*                                                   */
/* input: TDC*   dc                                  */
/*        TRect&                                     */
/*        float                                      */
/* ================================================= */
void BasicObject::Paint(TDC* dc, TRect& r, float s)
{ Draw(dc,r,FALSE,s);
}

#include "msoutlin.h"
#include "koutlin.h"
#include <string.h>
#include <stdio.h>

class ObjectDialog : public TDialog {
public:
  BasicObject* object;
  KOutline* outline;

  ObjectDialog(TWindow*, int resId, BasicObject* object);

  void SetupWindow();
  void UpdateListBox(char *filespec, int level, int &which);
};

ObjectDialog::ObjectDialog(TWindow* thewin, int resId, BasicObject* o)
    : TDialog(thewin,resId)
{
  object = o;
}

void ObjectDialog::SetupWindow()
```

```
{
  TDialog::SetupWindow();

  outline = new KOutline(this,IDC_FUNCTION_LIST);
  outline->Create();

  // Initialize Outline Object
  int which = 1;

  UpdateListBox(".\\FUNCTION", 1, which);
  outline->SetPropListIndex(1);
}

void ObjectDialog::UpdateListBox (char *filespec, int level, int& which)
{
    int  i,j,len;
    char tmpBuf[80], tmpBuf2[30];
    TListBox tmpBox((TWindow*)this,0,0,0,10,10);
    tmpBox.Attr.Style += ~WS_VISIBLE;
    tmpBox.Create();

    // Get Directories
    sprintf(tmpBuf,"%s\\*.*",filespec);
    tmpBox.DirectoryList(DDL_DIRECTORY+DDL_EXCLUSIVE,tmpBuf);

    j = tmpBox.GetCount();
    for (i=0;i<j;i++) {
      len = tmpBox.GetString(tmpBuf,i);
      if (len-1 > 0) {
      tmpBuf[len-1] = 0;
      if (strcmp("[..",tmpBuf)!=0) {
         outline->AddItem(&tmpBuf[1],level);
         which++;

         sprintf(tmpBuf2,"%s\\%s",filespec,&tmpBuf[1]);
         UpdateListBox(tmpBuf2,level+1,which);
      }
      }
    }

    // Get Files
    tmpBox.ClearList();
    sprintf(tmpBuf,"%s\\*.*",filespec);
    tmpBox.DirectoryList(DDL_READWRITE,tmpBuf);

    j = tmpBox.GetCount();
    for (i=0;i<j;i++) {
      len = tmpBox.GetString(tmpBuf,i);
      outline->AddItem(tmpBuf,level);
      which++;
    }
}

/* ================================================ */
/* SettingsDialog                                   */
/* ------------------------------------------------ */
/* Opens a dialog box and labels it with the object  */
/* type. However, this will not be used when each    */
/* object has its own dialog defined.               */
/*                                                  */
/* input: TWindow* thewin                           */
/* ================================================ */
void BasicObject::SettingsDialog(TWindow* thewin)
{
    ObjectDialog Dialog(thewin,IDD_BASICDIALOG,this);
    switch (typeId) {
        case BG_BASIC:        Dialog.SetCaption("BasicObject");           break;
        case BG_BOND:         Dialog.SetCaption("BondObject");                break;
        case BG_ESOURCE:      Dialog.SetCaption("Effort Source");             break;
        case BG_FSOURCE:      Dialog.SetCaption("Flow Source");           break;
        case BG_RESISTOR:     Dialog.SetCaption("Generalized Resistance");    break;
        case BG_CAPACITOR:    Dialog.SetCaption("Generalized Capacitance"); break;
        case BG_INDUCTOR:     Dialog.SetCaption("Generalized Inertance");     break;
        case BG_GYRATOR:      Dialog.SetCaption("Gyrator");                                   break;
        case BG_TRANSFORMER: Dialog.SetCaption("Transformer");                        break;
    }
    Dialog.Execute();
}
```

obj_elem.h

```
/* ================================================ */
/* Basic Element Object Classes                     */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*              Copyright (c) 1995                  */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

#if !defined(__obj_elem_h)  // Use file only if it's not already included.
#define __obj_elem_h

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include "bttlobj.h"

/* ================================================ */
/* Basic Element Class Definitions                  */
/* ------------------------------------------------ */

/* Zero Junction Object Class                       */
/* ------------------------------------------------ */
class Object0Junction : public BasicObject{
public:
    Object0Junction();
    Object0Junction(const Object0Junction& object) : BasicObject(object) {};

    void SettingsDialog(TWindow*);
};

/* One Junction Object Class                        */
/* ------------------------------------------------ */
class Object1Junction : public BasicObject{
public:
    Object1Junction();
    Object1Junction(const Object1Junction& object) : BasicObject(object) {} ;

    //void SettingsDialog(TWindow*);
};

/* Effort Source Object Class                       */
/* ------------------------------------------------ */
class ObjectESource : public BasicObject{
public:
    ObjectESource();
    ObjectESource(const ObjectESource& object) : BasicObject(object) {} ;

    //void SettingsDialog(TWindow*);
};

/* Flow Source Object Class                         */
/* ------------------------------------------------ */
class ObjectFSource : public BasicObject{
public:
    ObjectFSource();
    ObjectFSource(const ObjectFSource& object) : BasicObject(object) {} ;

    //void SettingsDialog(TWindow*);
};

/* Generalized Resistance Object Class              */
/* ------------------------------------------------ */
class ObjectResistor : public BasicObject{
public:
    ObjectResistor();
    ObjectResistor(const ObjectResistor& object) : BasicObject(object) {};

    //void SettingsDialog(TWindow*);
};

/* Generalized Capacitance Object Class             */
/* ------------------------------------------------ */
class ObjectCapacitor : public BasicObject{
public:
    ObjectCapacitor();
    ObjectCapacitor(const ObjectCapacitor& object) : BasicObject(object) {};

    //void SettingsDialog(TWindow*);
};
```

```
/* Generalized Inductance Object Class                */
/* -------------------------------------------------- */
class ObjectInductor : public BasicObject{
public:
    ObjectInductor();
    ObjectInductor(const ObjectInductor& object) : BasicObject(object) {};

    //void SettingsDialog(TWindow*);
};

/* Transformer Object Class                           */
/* -------------------------------------------------- */
class ObjectTransformer : public BasicObject{
public:
    ObjectTransformer();
    ObjectTransformer(const ObjectTransformer& object) : BasicObject(object) {};

    //void SettingsDialog(TWindow*);
};

/* Gyrator Object Class                               */
/* -------------------------------------------------- */
class ObjectGyrator : public BasicObject{
public:
    ObjectGyrator();
    ObjectGyrator(const ObjectGyrator& object) : BasicObject(object) {};

    //void SettingsDialog(TWindow*);
};

#endif
```

obj_elem.cpp

```
/* ================================================== */
/* Basic Element Object Classes                       */
/* -------------------------------------------------- */
/*                                                    */
/* -------------------------------------------------- */
/*   Code by Richard K. Branton - MIT co 94/95 EECS   */
/*                Copyright (c) 1995                  */
/*       Massachusetts Institute of Technology        */
/*         Mechanical Engineering Department          */
/* ================================================== */

/* ================================================== */
/* Include Files                                      */
/* -------------------------------------------------- */
#include "obj_elem.h"
#include "bttlapp.rh"

/* ================================================== */
/* Class Support Function Definitions                 */
/* -------------------------------------------------- */

/* ================================================== */
/* Zero Junction Object Class                         */
/* -------------------------------------------------- */

/* ================================================== */
/* Object0Junction                                    */
/* -------------------------------------------------- */
/* creates a new zero junction object.                */
/*                                                    */
/* ================================================== */
Object0Junction::Object0Junction()
    : BasicObject()
{
    typeId = BG_0JUNCTION;

    label = (char *) calloc (strlen("0"),sizeof(char));
    strcpy(label,"0");
}

/* ================================================== */
/* SettingsDialog                                     */
/* -------------------------------------------------- */
/* Opens a dialog box to allow the user to set        */
/* options for the Zero Junction.                     */
/*                                                    */
/* input: TWindow* thewin                             */
/* ================================================== */
void Object0Junction::SettingsDialog(TWindow* thewin)
{
    // Make a dialog specific to the Zero Junction
```

```
    TDialog Dialog(thewin,IDD_BASICDIALOG);
    Dialog.SetCaption("Zero Junction");
    Dialog.Execute();
}

/* ================================================ */
/* One Junction Object Class                        */
/* ------------------------------------------------ */

/* ================================================ */
/* Object1Junction                                  */
/* ------------------------------------------------ */
/* creates a new one junction object.               */
/*                                                  */
/* ================================================ */
Object1Junction::Object1Junction()
    : BasicObject()
{
    typeId = BG_1JUNCTION;

    label = (char *) calloc (strlen("1"),sizeof(char));
    strcpy(label,"1");
}

/* ================================================ */
/* Effort Source Object Class                       */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectESource                                    */
/* ------------------------------------------------ */
/* creates a new effort source object.              */
/*                                                  */
/* ================================================ */
ObjectESource::ObjectESource()
    : BasicObject()
{
    typeId = BG_ESOURCE;

    label = (char *) calloc (strlen("Se"),sizeof(char));
    strcpy(label,"Se");
}

/* ================================================ */
/* Flow Source Object Class                         */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectFSource                                    */
/* ------------------------------------------------ */
/* creates a new flow source object.                */
/*                                                  */
/* ================================================ */
ObjectFSource::ObjectFSource()
    : BasicObject()
{
    typeId = BG_FSOURCE;

    label = (char *) calloc (strlen("Sf"),sizeof(char));
    strcpy(label,"Sf");
}

/* ================================================ */
/* Generalized Resistance Object Class              */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectResistor                                   */
/* ------------------------------------------------ */
/* creates a new generalized resistance object.     */
/*                                                  */
/* ================================================ */
ObjectResistor::ObjectResistor()
    : BasicObject()
{
    typeId = BG_RESISTOR;

    label = (char *) calloc (strlen("R"),sizeof(char));
    strcpy(label,"R");
}

/* ================================================ */
/* Generalized Capacitance Object Class             */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectCapacitor                                  */
```

```
/* ------------------------------------------------ */
/* creates a new generalized capacitance object.    */
/*                                                  */
/* ================================================ */
ObjectCapacitor::ObjectCapacitor()
    : BasicObject()
{
    typeId = BG_CAPACITOR;

    label = (char *) calloc (strlen("C"),sizeof(char));
    strcpy(label,"C");
}

/* ================================================ */
/* Generalized Inductance Object Class              */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectInductor                                   */
/* ------------------------------------------------ */
/* creates a new generalized inductance object.     */
/*                                                  */
/* ================================================ */
ObjectInductor::ObjectInductor()
    : BasicObject()
{
    typeId = BG_INDUCTOR;

    label = (char *) calloc (strlen("I"),sizeof(char));
    strcpy(label,"I");
}

/* ================================================ */
/* Transformer Object Class                         */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectTransformer                                */
/* ------------------------------------------------ */
/* creates a new transformer object.                */
/*                                                  */
/* ================================================ */
ObjectTransformer::ObjectTransformer()
    : BasicObject()
{
    typeId = BG_TRANSFORMER;

    label = (char *) calloc (strlen("TF"),sizeof(char));
    strcpy(label,"TF");
}

/* ================================================ */
/* Gyrator Object Class                             */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectGyrator                                    */
/* ------------------------------------------------ */
/* creates a new gyrator object.                    */
/*                                                  */
/* ================================================ */
ObjectGyrator::ObjectGyrator()
    : BasicObject()
{
    typeId = BG_GYRATOR;

    label = (char *) calloc (strlen("GY"),sizeof(char));
    strcpy(label,"GY");
}
```

obj_comp.h

```
/* ================================================ */
/* Complex Element Object Classes                   */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
```

```
/*         Mechanical Engineering Department          */
/* ================================================== */

// Use file only if it's not already included.
#if !defined(__obj_comp_h)
#define __obj_comp_h

/* ================================================== */
/* Include Files                                      */
/* -------------------------------------------------- */
#include "bttlobj.h"
#include "bttldoc.h"

/* ================================================== */
/* Defines for Complex Objects                        */
/* -------------------------------------------------- */

/* ================================================== */
/* Complex Element Class Definitions                  */
/* -------------------------------------------------- */

/* Document Information Object Class                  */
/* -------------------------------------------------- */
class ObjectDocumentInfo : public BasicObject {
public:
    // Grid Information
    char  gridFrequency;
    Color gridColor;
    int   gridShow;
    int   gridSnap;
    BasicObject* inputObject;
   BasicObject* outputObject;
public:
    ObjectDocumentInfo();
};

/* Compound Object Class                              */
/* -------------------------------------------------- */
class ObjectCompound : public BasicObject {
public:
    // Could be either an external file or an internal file.
    int fileType; // 0 = empty, 1 = internal, 2 = external
    int inputNode;
    int outputNode;
    char *filename;
    char *children;
    int childOpen;
public:
    ObjectCompound();
    ObjectCompound(BondGraphDocument&);
    void Draw(TDC*,TRect&,BOOL,float);

    // Input/Output to streams.
    istream& Input(istream&);
    ostream& Output(ostream&);
    void FixLinks(TObjectArray* theArray);
    void SettingsDialog(TWindow* thewin);
   bool ExtractToArray(TObjectArray* theArray);
};

/* OLE Object Class                                   */
/* -------------------------------------------------- */
class ObjectOLE : public BasicObject {
public:
    ObjectOLE();
};

/* File Link Object Class(Like Compound Object)       */
/* -------------------------------------------------- */
class ObjectFileLink : public BasicObject {
public:
    ObjectFileLink();
};

/* Bond Object Class                                  */
/* -------------------------------------------------- */
class ObjectBond : public BasicObject {
public:
    // FixLinks information
    int fromId;
    int toId;

    // Bond attributes
    int Causality; // 0 = From, 1 = To, 2 = None
    int Power;       // 0 = From, 1 = To, 2 = None
    int BondType;  // 0 = Line, 1 = Polyline, 2 = Curve
   int number;
```

```
    // Visual information
    int x1,y1,x2,y2;

public:
    ObjectBond(BasicObject* = NULL, BasicObject* = NULL, int C = 2,int P = 2,int bt = 0);
    ObjectBond(const ObjectBond&);

    // Input/Output to streams.
    istream& Input(istream&);
    ostream& Output(ostream&);

    // Visual/Display Information
    void Draw(TDC*, TRect& rect, BOOL Selected = false, float Scale = 1.0);
    void Paint(TDC*, TRect& rect, float scale = 1.0);

    BOOL XYInObject(int px, int py);
    BOOL ObjectInRect(TRect rect);

    void SettingsDialog(TWindow*);

//  void FixLinks(BondGraphDocument&);
    void FixLinks(TObjectArray* theArray);
};

#endif
```

obj_comp.cpp

```
/* ================================================ */
/* Complex Element Object Classes                   */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include "obj_comp.h"
#include "bttlapp.rh"

#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <strstrea.h>

/* ================================================ */
/* Class Support Function Definitions               */
/* ------------------------------------------------ */

/* ================================================ */
/* Document Information Object Class                */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectDocumentInfo                               */
/* ------------------------------------------------ */
/* creates a new document info object.              */
/*                                                  */
/* ================================================ */
ObjectDocumentInfo::ObjectDocumentInfo()
    : BasicObject()
{
    typeId = BG_DOCUMENT;

    gridFrequency = 50;
    ColorBlack(gridColor);
}

/* ================================================ */
/* Compound Object Class                            */
/* ------------------------------------------------ */
void GetStringFromStream(istream& is, char **dibFilename);

/* ================================================ */
/* ObjectCompound                                   */
/* ------------------------------------------------ */
/* creates a new compound object.                   */
/*                                                  */
```

```
/* ================================================ */
ObjectCompound::ObjectCompound()
    : BasicObject()
{
    typeId = BG_COMPOUND;
    displayMode = DISPLAY_BOX;
    label = (char *) malloc (strlen("untitled")*sizeof(char)+1);
    strcpy(label,"untitled");
    childOpen = FALSE;
    fileType = 1;
    filename = 0;
    children = 0;
    subobjects = new TObjectArray(4,0,4);
}

bool ObjectCompound::ExtractToArray(TObjectArray* theArray)
{
    // if more than one object and input/output not set then exit
    BasicObject *inObject = NULL, *outObject = NULL;

    int numObjects = subobjects->GetItemsInContainer();
    for (int j=0;j<numObjects;j++) {
      if ((*subobjects)[j]->ioInput == TRUE)
          inObject = (*subobjects)[j];
      if ((*subobjects)[j]->ioOutput == TRUE)
          outObject = (*subobjects)[j];
    }

    if (inObject && outObject) {
      inObject->ioInput = FALSE;
      outObject->ioOutput = FALSE;

      // extract.
      numObjects = subobjects->GetItemsInContainer();
      for (j=numObjects-1;j>=0;j--) {
         theArray->Add((*subobjects)[j]);
         (*subobjects)[j]->selected = TRUE;
         subobjects->Detach(j);
      }

      // move input bonds to input object and output bonds to output object
      numObjects = superobjects->GetItemsInContainer();
      for (j=numObjects-1;j>=0;j--) {
          if ((*superobjects)[j]->typeId == BG_BOND) {
             if (((ObjectBond*)(*superobjects)[j])->Power == 0 && (*(*superobjects)[j]->subobjects)[0]
== this) {
               // coming in (From)
               (*(*superobjects)[j]->subobjects)[0] = inObject;
               inObject->superobjects->Add((*superobjects)[j]);
             } else if (((ObjectBond*)(*superobjects)[j])->Power == 1 && (*(*superobjects)[j]-
>subobjects)[1] == this) {
               // coming in (To)
               (*(*superobjects)[j]->subobjects)[1] = inObject;
               inObject->superobjects->Add((*superobjects)[j]);
             } else // going out or not marked
             if ((*(*superobjects)[j]->subobjects)[0] == this) {
               (*(*superobjects)[j]->subobjects)[0] = outObject;
               outObject->superobjects->Add((*superobjects)[j]);
             } else
             if ((*(*superobjects)[j]->subobjects)[1] == this) {
               (*(*superobjects)[j]->subobjects)[1] = outObject;
               outObject->superobjects->Add((*superobjects)[j]);
             }
          }
      }
     return TRUE;
    }

    return FALSE;
}

ObjectCompound::ObjectCompound(BondGraphDocument& doc)
//   : BasicObject()
{
    int i = 1;
    int minx = 32000, miny = 32000;
    int maxx = -32000, maxy = -32000;

    childOpen = FALSE;
    fileType = 1; // internal
    label = (char *) malloc (strlen("untitled")*sizeof(char)+1);
    strcpy(label,"untitled");

    // Give All Selected Objects a Distinct Id
    BasicObject* objectPtr, *object;
    doc.Start();
    while (objectPtr = doc.Current()) {
```

```
      if (objectPtr->selected)
         objectPtr->distinctId = i++;
      doc.Next();
    }

    // Put Objects Into Compound Object
    subobjects = new TObjectArray(4,0,4);
    int numObjects = doc.ObjectArray->GetItemsInContainer();
    for (i=numObjects-1;i>=0;i--) {
      if ((*doc.ObjectArray)[i]->selected)
         if ((*doc.ObjectArray)[i]->distinctId > 0 &&
              (*doc.ObjectArray)[i]->typeId != BG_BOND) {
              object = (*doc.ObjectArray)[i];
              object->selected = FALSE;

              if (object->typeId != BG_SHAPE) {
                 if (object->x < minx) minx = object->x;
                 if (object->y < miny) miny = object->y;
                 if (object->x+object->w > maxx) maxx = object->x+object->w;
                 if (object->y+object->h > maxy) maxy = object->y+object->h;
              }

              doc.ObjectArray->Detach(i);
              subobjects->Add(object);
            }
    }

    // Deselect remaining objects and set distinctIds
    doc.Start();
    while (objectPtr = doc.Current()) {
      objectPtr->selected = FALSE;
      objectPtr->distinctId = -1;
      doc.Next();
    }

    // Put Bonds Into Compound Object or Link Bond to Object
    numObjects = doc.ObjectArray->GetItemsInContainer();
    for (i=numObjects-1;i>=0;i--) {
      if ((*doc.ObjectArray)[i]->typeId == BG_BOND) {
         // check if from or to or both in compound
         object = (*doc.ObjectArray)[i];
         int x = ((*object->subobjects)[0]->distinctId > 0);
         int y = ((*object->subobjects)[1]->distinctId > 0);
         if (x==1 && y==1) {
            doc.ObjectArray->Detach(i);
            subobjects->Add(object);
         } else if (x==1 && y==0) {
            (*object->subobjects)[0] = this; // also need to remove it from the parents of the
original
         } else if (x==0 && y==1) {
            (*object->subobjects)[1] = this;
         }
      }
    }

    // Reset distinct ids of subobjects
    numObjects = subobjects->GetItemsInContainer();
    for (i=0;i<numObjects;i++)
     (*subobjects)[i]->distinctId = -1;

    // Set Object Specifics
    typeId = BG_COMPOUND;
    displayMode = DISPLAY_BOX;
    w = 40;
    h = 40;
    x = minx + (maxx - minx - w) / 2;
    y = miny + (maxy - miny - h) / 2;
    selected = TRUE;
    filename = 0;
    children = 0;
}

void ObjectCompound::Draw(TDC* dc, TRect&, BOOL drawSelected, float )
{
    // Draw Shadow
    if (displayMode == DISPLAY_SHADOW) {
      dc->SelectObject(TBrush(TColor(shadow[R],shadow[G],shadow[B])));
      dc->Rectangle(TRect(TPoint(x+shadowDepth,y+shadowDepth),TSize(w,h)));
    }

    // Draw Border and Background
    if (drawSelected)
          dc->SelectObject(TPen((TColor::LtRed), 0, PS_DOT));
    else if (displayMode == DISPLAY_NOBORDER)
          dc->SelectObject(TPen(TColor::Black, 0, PS_NULL));
        else
          dc->SelectObject(TPen(TColor(border[R],border[G],border[B]), 0, PS_SOLID));
```

```
    dc->SelectObject(TBrush(TColor(background[R],background[G],background[B])));

    dc->Rectangle(TRect(TPoint(x,y),TSize(w,h)));

    // Draw Label
    if (label) {
      HFONT hfont = CreateFont(14,labelFont[1],labelFont[2],labelFont[3],labelFont[4],

    labelFont[5],labelFont[6],labelFont[7],labelFont[8],labelFont[9],
                                          labelFont[10],labelFont[11],labelFont[12],labelFontName);
      TFont *theFont = new TFont(hfont);

      dc->SetTextColor(TColor(foreground[R],foreground[G],foreground[B]));
      dc->SelectObject(*theFont);
      dc->SetBkMode(TRANSPARENT);
      dc->DrawText(label,-1,TRect(TPoint(x,y),TSize(w,h)),DT_CENTER | DT_VCENTER);
      dc->RestoreFont();
//    dc->RestoreTextBrush();

      delete theFont;
      DeleteObject(hfont);
    }

    // if selected draw mini boxes
    if (drawSelected) {
        dc->SelectObject(TPen(TColor::LtRed, 0, PS_SOLID));
        DrawMiniBox(dc,x-5,y-5);
        DrawMiniBox(dc,x+w+1,y-5);
        DrawMiniBox(dc,x+w+1,y+h+1);
        DrawMiniBox(dc,x-5,y+h+1);
    }

    dc->RestorePen();
}

/* ================================================ */
/* Input                                            */
/* ------------------------------------------------ */
/* Reads the object from the input stream is.       */
/*                                                  */
/* Input Format:                                    */
/*   ,fileType,(filename),(label)                   */
/*   ,(child1,child2,...,childn)                    */
/*   ,x,y,w,h,foreground,background,shadow,border   */
/*   ,shadowDepth,displayMode,lineThickness,lineStyle*/
/*   ,distinctId)                                   */
/*                                                  */
/* input:  istream& is                              */
/* output: istream&                                 */
/* ================================================ */
istream& ObjectCompound::Input(istream& is)
{
    char c;
    int r,g,b;
    FILE* fp = fopen("test.txt","w");
    // Specific Stuff
    is >> c >> fileType;

    fprintf(fp,"fileType = %d\n",fileType); fflush(fp);
    if (fileType == 2) { // external
      is >> c; GetStringFromStream(is,&filename);
    }

    fprintf(fp,"filename = '%s'\n",filename); fflush(fp);
    is >> c; GetStringFromStream(is,&label);

    fprintf(fp,"label = '%s'\n",label); fflush(fp);
    if (fileType == 1) {
      is >> c; GetStringFromStream(is,&children);
    }

    fprintf(fp,"children = '%s'\n",children); fflush(fp);
    // General Stuff
    is >> c >> x >> c >> y;
    is >> c >> w >> c >> h;

    is >> c >> r >> c >> g >> c >> b;
    foreground[R] = r; foreground[G] = g; foreground[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    background[R] = r; background[G] = g; background[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    shadow[R] = r;     shadow[G] = g;     shadow[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    border[R] = r;     border[G] = g;     border[B] = b;
```

```
    is >> c >> r >> c >> g;
    shadowDepth = r;
    displayMode = g;

    is >> c >> r >> c >> g;
    lineThickness = r;
    lineStyle = g;

    is >> c >> distinctId;
    is >> c;

    fprintf(fp,"distinctId = %d\n",distinctId); fflush(fp);
    fclose(fp);
    return is;
}

void ObjectCompound::FixLinks(TObjectArray* theArray)
{
    // Count num of commas
    int j,num,numObjects;
    int i, count = 1;

    if (children && fileType == 1) {
      for (i=0;i<strlen(children);i++)
         if (children[i] == ',')
            count++;

      istrstream* is = new istrstream(children);

      // remove each subobject from
      char c;

      //BasicObject* object;
      for (i=0;i<count;i++) {
         *is >> num >> c;
         numObjects = theArray->GetItemsInContainer();
         for (j=numObjects-1;j>=0;j--) {
            if ((*theArray)[j]->distinctId == num) {
              // remove from theArray and place in subobjects
              subobjects->Add((*theArray)[j]);
              (*theArray)[j]->distinctId = -1;
              theArray->Detach(j);
              j=-1;
            }
         }
      }

      delete is;
      free (children);
      children = NULL;
    }
}

/* ================================================ */
/* Output                                           */
/* ------------------------------------------------ */
/* Writes the object to the Output stream os.       */
/*                                                  */
/* Output Format:                                   */
/*   (OBJECT_TYPE_ID                                */
/*   ,fileType,(filename),(label)                   */
/*   ,(child1,child2,...,childn)                    */
/*   ,x,y,w,h,foreground,background,shadow,border   */
/*   ,shadowDepth,displayMode,lineThickness,lineStyle*/
/*   ,distinctId)                                   */
/*                                                  */
/* input:  ostream& os                              */
/* output: ostream&                                 */
/* ================================================ */
ostream& ObjectCompound::Output(ostream& os)
{
    char c = ',';

    os << '(' << (int) typeId;
    os << c << fileType;
    if (fileType == 2) { os << c << '(' << filename << ')'; }

    os << c << '(' << label << ')';

    int i,numObjects;
    if (fileType == 1) {
      os << c << '(';
      numObjects = subobjects->GetItemsInContainer();
      if (numObjects > 0) {
         for (i=0;i<numObjects-1;i++) {
            os << SetDistinctId(*(*subobjects)[i]) << ',';
```

```
        }
        os << SetDistinctId(*(*subobjects)[i]);
      }
      os << ')';
    }

    os << c << x << c << y;
    os << c << w << c << h;
    os << c << (int) foreground[R] << c << (int) foreground[G] << c << (int) foreground[B];
    os << c << (int) background[R] << c << (int) background[G] << c << (int) background[B];
    os << c << (int) shadow[R]     << c << (int) shadow[G]     << c << (int) shadow[B];
    os << c << (int) border[R]     << c << (int) border[G]     << c << (int) border[B];
    os << c << (int) shadowDepth   << c << (int) displayMode;
    os << c << (int) lineThickness << c << (int) lineStyle;
    os << c << SetDistinctId(*this);
    os << ')' << '\n';

    if (fileType == 1)
      WriteToStream((ostrstream*)&os,subobjects,TRUE,FALSE);

    return os;
}

// ========================================================================
class CompoundDialog : public TDialog {
public:
  TWindow*          window;
  ObjectCompound* object;
  TRadioButton*   button;
  TRadioButton*   button2;
  TEdit*          editLabel;
  TEdit*          editFilename;

  CompoundDialog(TWindow* w, int r, ObjectCompound* o) : TDialog(w,r) {object = o; window = w;}
  ~CompoundDialog();

  void CmCompoundEdit();
  void SetupWindow();
  void CmOk();

  DECLARE_RESPONSE_TABLE(CompoundDialog);
};

DEFINE_RESPONSE_TABLE1(CompoundDialog,TDialog)
    EV_COMMAND(IDC_COMPOUND_EDITFILE,CmCompoundEdit),
    EV_COMMAND(IDOK, CmOk),
END_RESPONSE_TABLE;

void CompoundDialog::CmCompoundEdit()
{
  if (button->GetCheck() == BF_CHECKED) {
      // open document
      CloseWindow(IDOK);
      window->GetApplication()->GetDocManager()->CreateAnyDoc(object->filename, dtAutoOpen);

  } else {
      CloseWindow(IDOK);

      BondGraphDocument* newDoc = (BondGraphDocument*) window->GetApplication()->GetDocManager()-
>CreateAnyDoc(0, dtNewDoc);

      if (newDoc->Open(0)) {
         newDoc->isSubDocument = TRUE;
         int numObjects = object->subobjects->GetItemsInContainer();
         for (int i=numObjects-1;i>=0;i--) {
           newDoc->AddObject((*object->subobjects)[i]);
           object->subobjects->Detach(i);
         }
         object->childOpen = TRUE;
         newDoc->parentObject = object;
      }
  }
}

CompoundDialog::~CompoundDialog()
{
  delete button;
  delete button2;
  delete editLabel;
  delete editFilename;
}

void CompoundDialog::SetupWindow()
{
    TDialog::SetupWindow();

    // Create And Check File Type
```

```
    button = new TRadioButton(this,IDC_COMPOUND_FILELINK); button->Create();
    button2 = new TRadioButton(this,IDC_COMPOUND_FILEIN); button2->Create();

    if (object->fileType == 2) { // external
      button->Check();
      button2->Uncheck();
    } else { // internal
      button->Uncheck();
      button2->Check();
    }

    editLabel = new TEdit(this,IDC_COMPOUND_LABEL,100);    editLabel->Create();
    editFilename = new TEdit(this,IDC_COMPOUND_FILE,100); editFilename->Create();

    char buffer[100];
    if (object->label) {sprintf(buffer,"%s",object->label); editLabel->Transfer(buffer,tdSetData); }
    if (object->filename) {sprintf(buffer,"%s",object->filename); editFilename-
>Transfer(buffer,tdSetData); }
}

void CompoundDialog::CmOk()
{
    // Copy label
    char buffer[100];
    editLabel->Transfer(buffer,tdGetData);

    free(object->label);
    object->label = (char *) malloc ((strlen(buffer)+1)*sizeof(char));
    strcpy(object->label,buffer);

    if (button->GetCheck() == BF_CHECKED) {
      // change filename
      editFilename->Transfer(buffer,tdGetData);

      free(object->filename);
      object->filename = (char *) malloc ((strlen(buffer)+1)*sizeof(char));
      strcpy(object->filename,buffer);
    }

    TDialog::CmOk();
}

// ========================================================================

void ObjectCompound::SettingsDialog(TWindow* thewin)
{
    CompoundDialog dialog(thewin,IDD_COMPOUNDDIALOG,this);
    dialog.Execute();
   thewin->Invalidate();
}

/* ================================================ */
/* OLE Object Class                                 */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectOLE                                        */
/* ------------------------------------------------ */
/* creates a new OLE object.                        */
/*                                                  */
/* ================================================ */
ObjectOLE::ObjectOLE()
    : BasicObject()
{
    typeId = BG_OLE;
}

/* ================================================ */
/* File Link Object Class                           */
/* ------------------------------------------------ */

/* ================================================ */
/* ObjectFileLink                                   */
/* ------------------------------------------------ */
/* creates a new file link object.                  */
/*                                                  */
/* ================================================ */
ObjectFileLink::ObjectFileLink()
    : BasicObject()
{
    typeId = BG_FILELINK;
}

/* ================================================ */
/* Bond Object Class                                */
/* ------------------------------------------------ */
```

```
/* ------------------------------------------------ */
/* Bond Object Support Functions                    */
/* ------------------------------------------------ */

#define round(a) ((a)+ 0.5)
#define THRESHOLD 0.000001

bool IntersectLineAndCircle(float m, int d, int& x1, int& y1, int& x2, int& y2)
{
  float radical = (1.0+m*m);
  if (radical <= THRESHOLD)
     return FALSE;
  float root = d/sqrt(radical);
  x1 = round(root); y1 = round(m * root);
  x2 = -x1;         y2 = round(-m * root);
  return TRUE;
}

bool FindPointOnLine(int x1, int y1, int h, int k, int d,
                             int& x3, int& y3, int& x4, int& y4)
{
  char result = TRUE;
  if (x1 == h || fabs(0.0 + h - x1) < THRESHOLD ) { // vertical line
     x3 = x4 = h;
     y3 = k - d;
     y4 = k + d;
  } else if (y1 == k) { // horizontal line
     y3 = y4 = k;
     x3 = h - d;
     x4 = h + d;
  } else { // other lines
     float m = (0.0 + k - y1) / (0.0 + h - x1);
     result = IntersectLineAndCircle(m,d,x3,y3,x4,y4);
     x3 += h; y3 += k;
     x4 += h; y4 += k;
  }
  return result;
}

bool FindPointOnSegment(int x1, int y1, int h, int k, int d,
                                  int& x3, int& y3)
{
  char result = TRUE;
  float tv1, tv2;
  int x4, y4;

  result = FindPointOnLine(x1,y1,h,k,d,x3,y3,x4,y4);
  if (result) {
     tv1 = (x1-x3); tv1 *= tv1;
     tv2 = (y1-y3); tv2 *= tv2;
     float dto3 = sqrt(tv1 + tv2);

     tv1 = (x1-x4); tv1 *= tv1;
     tv2 = (y1-y4); tv2 *= tv2;
     float dto4 = sqrt(tv1 + tv2);

     if (dto3 > dto4) { // swap them
        x3 = x4; y3 = y4;
     }
  }

  return result;
}

bool GetTrianglePoints(int x1, int y1, int x2, int y2, int d1, int d2,
                               int& x3, int&y3, int& x4, int& y4, int& x5, int& y5,
                               int ArrowVsLine)
{
    if (FindPointOnSegment(x1,y1,x2,y2,d1,x3,y3)) {
      if (y2 != y1) {
         float invM = -1.0 * (x2 - x1) / (y2 - y1);
         if (IntersectLineAndCircle(invM,d2,x4,y4,x5,y5)) {
            if (ArrowVsLine) {
              x4 += x3; y4 += y3;
              x5 += x3; y5 += y3;
            } else {
              x4 += x2; y4 += y2;
              x5 += x2; y5 += y2;
            }
            return TRUE;
         }
      } else {
         if (ArrowVsLine)
            if (x1 < x2)
                x3 = x2 - d1;
            else
                x3 = x2 + d1;
```

```
          else x3 = x2;

          y3 = y2;
          x4 = x3; y4 = y2 - d2;
          x5 = x3; y5 = y2 + d2;
          return TRUE;
       }
     }
  return FALSE;
}

#define HANDLEWD 2

/* ================================================ */
/* PtOnLine                                         */
/* ------------------------------------------------ */
/* Tells whether a point is on a line, or near it.  */
/*                                                  */
/* input:  int inner                                */
/*         int px, py, x1, y1, x2, y2               */
/* output: BOOL                                     */
/* ================================================ */
BOOL PtOnLine (int inner, int px, int py, int x1, int y1, int x2, int y2)
{
  double t, tx, ty, deltax, deltay;
  deltax = x2 - x1;
  deltay = y2 - y1;
  if (fabs(deltax) > 0 && fabs(deltay) < fabs(deltax)) {
     t = (double) (px - x1) / ( x2 - x1);
     if (t >= 0 && t <= 1) {
        ty = y1 + (y2 - y1) * t;
        if (ty - HANDLEWD <= py && py <= ty + HANDLEWD)
          return TRUE;
     }
     if (inner==0)
        return PtOnLine(1,px,py,x2,y2,x1,y1);
     else
        return FALSE;
  } else if (fabs(deltay) > 0) {
     t = (double) (py - y1) / (y2 - y1);
     if (t >= 0 && t <= 1) {
        tx = x1 + (x2 - x1) * t;
        if (tx - HANDLEWD <= px && px <= tx + HANDLEWD)
          return TRUE;
     }
     if (inner==0)
        return PtOnLine(1,px,py,x2,y2,x1,y1);
     else
        return FALSE;
  }

  if (inner==0)
     return PtOnLine(1,px,py,x2,y2,x1,y1);
  else
     return FALSE;
}

/* ================================================ */
/* GetBondLine                                      */
/* ------------------------------------------------ */
/* Calculates the position of the line connecting   */
/* FromO to toO.                                    */
/*                                                  */
/* input:  BasicObject *FromO, *toO                 */
/*         int &x1, &y1, &x2, &y2                   */
/* ================================================ */
void GetBondLine(BasicObject* FromO, BasicObject* ToO, int& x1, int& y1, int& x2, int& y2)
{
    int x[2], y[2], f[2], w[2], h[2], tx, ty, tw, th;

    // Initialize Variables
    f[0] = 1;                          f[1] = 0;
    x[0] = FromO->x + FromO->w / 2;    x[1] = ToO->x + ToO->w / 2;
    y[0] = FromO->y + FromO->h / 2;    y[1] = ToO->y + ToO->h / 2;
    w[0] = FromO->w / 2;                w[1] = ToO->w / 2;
    h[0] = FromO->h / 2;                h[1] = ToO->h / 2;

    // Make sure lines go left to right. Swap if necessary
    if (x[0] > x[1]) {
      tx = x[0]; x[0] = x[1]; x[1] = tx;
      ty = y[0]; y[0] = y[1]; y[1] = ty;
      tw = w[0]; w[0] = w[1]; w[1] = tw;
      th = h[0]; h[0] = h[1]; h[1] = th;
      f[0] = 0;  f[1] = 1;
    }

    if (x[0] == x[1]) {  // Vertical Line
```

```
      if (y[0] < y[1]) {
         y[0] += h[0];
         y[1] -= h[1];
      } else {
         y[0] -= h[0];
         y[1] += h[1];
      }

    } else {                  // Nonvertical Line
      // Calculate Slope Of Line
      float m = (y[1] - y[0]) / (x[1] - x[0]);
      float am = fabs(m);
      if (am < 1) {           // Slope is small
         x[0] += w[0];
         x[1] -= w[1];

      } else if (am > 1) {  // Slope is large
         if (y[0] < y[1]) {       // is the start higher than the finish
            y[0] += h[0];
            y[1] -= h[1];
         } else {                  // is the start lower than the finish
            y[0] -= h[0];
            y[1] += h[1];
         }

      } else {                     // Slope is perfect diagonal
         if (y[0] < y[1]) {       // is the start higher than the finish
            x[0] += w[0];
            y[0] += h[0];
            x[1] -= w[0];
            y[1] -= h[0];
         } else {                  // is the start lower than the finish
            x[0] += w[0];
            y[0] -= h[0];
            x[1] -= w[0];
            y[1] += h[0];
         }
      }
    }

    // If numbers were swapped, swap them back.
    if (f[0] == 1) { x1 = x[0]; y1 = y[0]; x2 = x[1]; y2 = y[1]; }
      else { x2 = x[0]; y2 = y[0]; x1 = x[1]; y1 = y[1]; }


    if (FromO->displayMode == DISPLAY_NOBORDER) {
        x1 = FromO->x + FromO->w / 2;
        y1 = FromO->y + FromO->h / 2;

    }

    if (ToO->displayMode == DISPLAY_NOBORDER) {
        x2 = ToO->x + ToO->w / 2;
        y2 = ToO->y + ToO->h / 2;
    }

    // Move Away From Calculated Point To Allow Space For Power and Causality
    // Strokes.

    if (FromO->displayMode == DISPLAY_NOBORDER)
      FindPointOnSegment(x2,y2,x1,y1,30,x1,y1);
    else FindPointOnSegment(x2,y2,x1,y1,10,x1,y1);

    if (ToO->displayMode == DISPLAY_NOBORDER)
      FindPointOnSegment(x1,y1,x2,y2,30,x2,y2);
    else FindPointOnSegment(x1,y1,x2,y2,10,x2,y2);
}

/* ================================================ */
/* ObjectBond                                       */
/* ------------------------------------------------ */
/* Creates a bond object.                           */
/*                                                  */
/* input: BasicObject *From, *Toct                  */
/*        int Caus, Pow, bt                         */
/* ================================================ */
ObjectBond::ObjectBond(BasicObject* From, BasicObject* To, int Caus, int Pow, int bt)
    : BasicObject()
{
    typeId = BG_BOND;

    // Link to objects
    subobjects = new TObjectArray(2,0,2);
    if (From != NULL && To != NULL) {
      subobjects->Add(From);
      subobjects->Add(To);
      From->superobjects->Add(this);
```

```
      To->superobjects->Add(this);
    }

    // Bond Attributes
    Causality = Caus;
    Power = Pow;
    BondType = bt;

    // FixLinks Information
    fromId = toId = -1;
}

/* ================================================ */
/* ObjectBond                                       */
/* ------------------------------------------------ */
/* Copy constructor. creates a copy of object.      */
/*                                                  */
/* input: const ObjectBond& object                  */
/* ================================================ */
ObjectBond::ObjectBond(const ObjectBond& object)
{
    // Bond Specific Information
    fromId = object.fromId;
    toId = object.toId;

    Causality = object.Causality;
    Power = object.Power;
    BondType = object.BondType;

    x2 = object.x2;
    y2 = object.y2;

    // Copy Links
    (*subobjects)[0] = (*object.subobjects)[0];
    (*subobjects)[1] = (*object.subobjects)[1];

    // Copy Parents
    delete superobjects;
    if (object.superobjects == NULL)
      superobjects = NULL;
    else {
      superobjects = new TObjectArray(object.superobjects->GetItemsInContainer(),0,10);
      TObjectArrayIterator ObjectIterator2(*object.superobjects);
      for (;ObjectIterator2.Current();ObjectIterator2++)
         superobjects->Add(ObjectIterator2.Current());
    }

    // General Information
    ColorCopy(foreground,object.foreground);
    ColorCopy(background,object.background);
    ColorCopy(shadow,object.shadow);
    ColorCopy(border,object.border);

    shadowDepth = object.shadowDepth;
    displayMode = object.displayMode;

    lineThickness = object.lineThickness;
    lineStyle = object.lineStyle;

    free (label);
    label = (char *) calloc (strlen(object.label)+1,sizeof(char));
    strcpy(label, object.label);

    // Need To Change
    image = object.image;

    free (attachedText);
    attachedText = (char *) calloc (strlen(object.attachedText)+1,sizeof(char));
    strcpy(attachedText, object.attachedText);

    typeId = object.typeId;
    selected = object.selected;
    distinctId = object.distinctId;
}

/* ================================================ */
/* Input                                            */
/* ------------------------------------------------ */
/* Reads the bond from the input stream is.         */
/*                                                  */
/* Input Format:                                    */
/*   ,fromID,toID,border,lineThickness,lineStyle,   */
/*   Causality,Power,BondType,distinctId)           */
/*                                                  */
/* input:  istream& is                              */
/* output: istream&                                 */
/* ================================================ */
```

```
istream& ObjectBond::Input(istream& is)
{
    char c;

    is >> c >> (int) fromId        >> c >> (int) toId;
    is >> c >> (int) border[R]     >> c >> (int) border[G] >> c >> (int) border[B];
    is >> c >> (int) lineThickness >> c >> (int) lineStyle;
    is >> c >> (int) Causality     >> c >> (int) Power     >> c >> (int) BondType;
    is >> c >> distinctId;
    is >> c;

    return is;
}

/* ================================================ */
/* Output                                           */
/* ------------------------------------------------ */
/* Writes the object to the Output stream os.       */
/*                                                  */
/* Output Format:                                   */
/*   (OBJECT_TYPE_ID,                               */
/*   ,fromId,toId,border,lineThickness,lineStyle,   */
/*   Causality,Power,BondType,distinctId)           */
/*                                                  */
/* input:  ostream& os                              */
/* output: ostream&                                 */
/* ================================================ */
ostream& ObjectBond::Output(ostream& os)
{
    char c = ',';

    os << '(' << (int) typeId;
    os << c << (int) SetDistinctId(*(*subobjects)[0])      << c << (int)
SetDistinctId(*(*subobjects)[1]);
    os << c << (int) border[R]     << c << (int) border[G] << c << (int) border[B];
    os << c << (int) lineThickness << c << (int) lineStyle;
    os << c << (int) Causality     << c << (int) Power     << c << (int) BondType;
    os << c << (int) SetDistinctId(*this);
    os << ')';

    return os;
}

/* ================================================ */
/* XYInObject                                       */
/* ------------------------------------------------ */
/* Tells whether the point px, py is in the object. */
/*                                                  */
/* input:  int px, py                               */
/* output: BOOL                                     */
/* ================================================ */
BOOL ObjectBond::XYInObject(int px, int py)
{
    GetBondLine((*subobjects)[0], (*subobjects)[1], x1, y1, x2, y2);
    x = x1; if (x2 < x1) x = x2; w = abs(x2 - x1);
    y = y1; if (y2 < y1) y = y2; h = abs(y2 - y1);
    return PtOnLine(0,px,py,x1,y1,x2,y2);
}

/* ================================================ */
/* ObjectInRect                                     */
/* ------------------------------------------------ */
/* Tells whether the object is in the rectangle rect.*/
/*                                                  */
/* input:  TRect rect                               */
/* output: BOOL                                     */
/* ================================================ */
BOOL ObjectBond::ObjectInRect(TRect rect)
{
    GetBondLine((*subobjects)[0], (*subobjects)[1], x1, y1, x2, y2);
    x = x1; if (x2 < x1) x = x2; w = abs(x2 - x1);
    y = y1; if (y2 < y1) y = y2; h = abs(y2 - y1);
    TRect objRect(TPoint(x,y),TSize(w,h));
    return (rect.Contains(objRect));
}

/* ================================================ */
/* Draw                                             */
/* ------------------------------------------------ */
/* Draws the object to the screen dc.               */
/*                                                  */
/* input: TDC*   dc                                 */
/*        TRect&                                    */
/*        BOOL   drawSelected                       */
/*        float                                     */
/* ================================================ */
void ObjectBond::Draw(TDC* dc, TRect&, BOOL drawSelected, float)
```

```
{
    GetBondLine((*subobjects)[0], (*subobjects)[1], x1, y1, x2, y2);
    if (drawSelected) {
      dc->SelectObject(TPen(TColor::LtRed,0,PS_DOT));
      dc->SelectObject(TBrush(TColor::LtRed));
    } else {
      dc->SelectObject(TPen(TColor(foreground[R],foreground[G],foreground[B])));
      dc->SelectObject(TBrush(TColor(foreground[R],foreground[G],foreground[B])));
    }

    // Draw Bond
    dc->MoveTo(x1,y1);
    dc->LineTo(x2,y2);

    TPoint points[3];
    int x3,y3,x4,y4,x5,y5;

    // Draw Causality
    if (Causality == 1) {
      if (GetTrianglePoints(x1,y1,x2,y2,28,10,x3,y3,x4,y4,x5,y5,FALSE))
         { dc->MoveTo(x4,y4); dc->LineTo(x5,y5); }
    } else if (Causality == 0) {
      if (GetTrianglePoints(x2,y2,x1,y1,28,10,x3,y3,x4,y4,x5,y5,FALSE))
         { dc->MoveTo(x4,y4); dc->LineTo(x5,y5); }
    }

    // Draw Arrow
    if (Power == 1) {
      if (GetTrianglePoints(x1,y1,x2,y2,20,10,x3,y3,x4,y4,x5,y5,TRUE)) {
         points[0].x = x2; points[0].y = y2;
         points[1].x = x3; points[1].y = y3;
         if (y4 < y5) { points[2].x = x4; points[2].y = y4; }
            else      { points[2].x = x5; points[2].y = y5; }
         dc->Polygon(points,3);
      }
    } else if (Power == 0) {
      if (GetTrianglePoints(x2,y2,x1,y1,20,10,x3,y3,x4,y4,x5,y5,TRUE)) {
         points[0].x = x1; points[0].y = y1;
         points[1].x = x3; points[1].y = y3;
         if (y4 < y5) { points[2].x = x4; points[2].y = y4; }
            else      { points[2].x = x5; points[2].y = y5; }
         dc->Polygon(points,3);
      }
    }

    //Restore Old dc Settings
    dc->RestorePen();
    dc->RestoreBrush();
}

/* ================================================ */
/* Paint                                            */
/* ------------------------------------------------ */
/* Paints the object to the printer dc.             */
/*                                                  */
/* input: TDC*   dc                                 */
/*        TRect&                                    */
/*        float                                     */
/* ================================================ */
void ObjectBond::Paint(TDC* dc, TRect& r, float s)
{   Draw(dc,r,FALSE,s);
}

/* ================================================ */
/* SettingsDialog                                   */
/* ------------------------------------------------ */
/* Opens a dialog box to allow the user to set      */
/* options for the Bond.                            */
/*                                                  */
/* input: TWindow* thewin                           */
/* ================================================ */
void ObjectBond::SettingsDialog(TWindow* thewin)
{
    TDialog Dialog(thewin,IDD_BONDDIALOG);
    Dialog.Execute();
}

/* ================================================ */
/* FixLinks                                         */
/* ------------------------------------------------ */
/* Is called when a file is opened, or data is      */
/* pasted into the document. It is necessary to link */
/* the bonds which had links between them.          */
/*                                                  */
/* input: BondGraphDocument& doc                    */
/* ================================================ */
//void ObjectBond::FixLinks(BondGraphDocument& doc)
```

```
void ObjectBond::FixLinks(TObjectArray* theArray)
{
    if (fromId != -1 && toId != -1) {
        // Find and attach to 'From' and 'To' objects
        BasicObject* objectPtr;
        int numObjects = theArray->GetItemsInContainer();   //
        for (int i=0;i<numObjects;i++) {                    //
          objectPtr = (*theArray)[i];                        //
            if (objectPtr->distinctId == fromId)
                (*subobjects)[0] = objectPtr;
            else if (objectPtr->distinctId == toId)
                (*subobjects)[1] = objectPtr;
        }
    }
    fromId = -1;
    toId = -1;
}
```

obj_shap.h

```
/* ================================================ */
/* Shape Object Classes                             */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

#if !defined(__obj_shap_h)    // Use file only if it's not already included.
#define __obj_shap_h

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include "bttlobj.h"
#include "bttldoc.h"

/* ================================================ */
/* Defines for Shape Objects                        */
/* ------------------------------------------------ */
#define SH_LINE      0
#define SH_SQUARE    1
#define SH_RECTANGLE 2
#define SH_CIRCLE    3
#define SH_ELLIPSE   4
#define SH_TEXT      5
#define SH_BITMAP    6

/* ================================================ */
/* Shape Class Definition                           */
/* ------------------------------------------------ */
class ObjectShape : public BasicObject {
public:
    int shapeType;
    TDib* Dib;
    char* dibFilename;
    int x1,y1,x2,y2;

public:
    ObjectShape(uint which=SH_RECTANGLE);

    // Input/Output to streams.
    virtual istream& Input(istream&);
    virtual ostream& Output(ostream&);

    void Draw(TDC*, TRect& rect, BOOL Selected = false, float Scale = 1.0);
    void Paint(TDC*, TRect&, float);

    BOOL ObjectInRect(TRect rect);
    BOOL XYInObject(int px, int py);
};

#endif
```

obj_shap.cpp

```
/* ================================================ */
/* Shape Object Classes                             */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
```

```
/*   Code by Richard K. Branton - MIT co 94/95 EECS  */
/*               Copyright (c) 1995                  */
/*       Massachusetts Institute of Technology       */
/*         Mechanical Engineering Department         */
/* ============================================== */

/* ============================================== */
/* Include Files                                  */
/* ---------------------------------------------- */
#include "obj_shap.h"
#include "bttlapp.rh"

/* ============================================== */
/* Class Support Function Definitions             */
/* ---------------------------------------------- */
BOOL PtOnLine (int inner, int px, int py, int x1, int y1, int x2, int y2);

/* ============================================== */
/* Shape Object Class                             */
/* ---------------------------------------------- */

/* ============================================== */
/* ObjectShape                                    */
/* ---------------------------------------------- */
/* creates a shape object.                        */
/*                                                */
/* input: uint shapeT                             */
/* ============================================== */
ObjectShape::ObjectShape(uint shapeT)
    : BasicObject()
{
    typeId = BG_SHAPE;
    shapeType = shapeT;
    Dib = 0;
    dibFilename = 0;
}

/* ============================================== */
/* Draw                                           */
/* ---------------------------------------------- */
/* Draws the object to the screen dc.             */
/*                                                */
/* input: TDC*   dc                               */
/*        TRect&                                  */
/*        BOOL   drawSelected                     */
/*        float                                   */
/* ============================================== */
void ObjectShape::Draw(TDC* dc, TRect&, BOOL drawSelected, float )
{
    // Draw Shadow
    // dc->SelectObject(TBrush(TColor(shadow[R],shadow[G],shadow[B])));
    // dc->Rectangle(TRect(TPoint(x+shadowDepth,y+shadowDepth),TSize(w,h)));

    // Draw Border and Background
    if (drawSelected)
          dc->SelectObject(TPen((TColor::LtRed), 0, PS_DOT));
    else dc->SelectObject(TPen(TColor(foreground[R],foreground[G],foreground[B])));
    dc->SelectObject(TBrush(TColor(background[R],background[G],background[B])));

    switch (shapeType) {
        case SH_LINE:
            dc->MoveTo(x,y); dc->LineTo(w,h);
            break;

        case SH_SQUARE:
            dc->Rectangle(TRect(TPoint(x,y),TSize(w,w)));
            break;

        case SH_RECTANGLE:
            dc->Rectangle(TRect(TPoint(x,y),TSize(w,h)));
            break;

        case SH_CIRCLE:
            dc->Ellipse(TRect(TPoint(x,y),TSize(w,w)));
            break;

        case SH_ELLIPSE:
            dc->Ellipse(TRect(TPoint(x,y),TSize(w,h)));
            break;

        case SH_TEXT:
            if (label) {
                HFONT hfont =
CreateFont(labelFont[0],labelFont[1],labelFont[2],labelFont[3],labelFont[4],

    labelFont[5],labelFont[6],labelFont[7],labelFont[8],labelFont[9],
```

```
    labelFont[10],labelFont[11],labelFont[12],labelFontName);
                TFont *theFont = new TFont(hfont);

                dc->SetTextColor(TColor(foreground[R],foreground[G],foreground[B]));
                dc->SelectObject(*theFont);
                dc->SetBkMode(TRANSPARENT);
                dc->DrawText(label,-1,TRect(TPoint(x,y),TSize(w,h)),DT_LEFT | DT_NOCLIP);
                dc->RestoreFont();

                delete theFont;
                DeleteObject(hfont);
            }
            break;

        case SH_BITMAP:
            if (Dib)
              SetDIBitsToDevice(*dc,x,y,Dib->Width(),Dib->Height(),0,0,0,Dib->NumScans(),
                                Dib->GetBits(),Dib->GetInfo(),Dib->Usage());
            else {
              dc->Rectangle(TRect(TPoint(x,y),TSize(w,h)));
              dc->MoveTo(x,y);   dc->LineTo(x+w,y+h);
              dc->MoveTo(x,y+h); dc->LineTo(x+w,y);
            }
            break;
    };

    // if selected draw mini boxes
    if (drawSelected) {
        dc->SelectObject(TPen(TColor::LtRed, 0, PS_SOLID));
        DrawMiniBox(dc,x-5,y-5);
        DrawMiniBox(dc,x+w+1,y-5);
        DrawMiniBox(dc,x+w+1,y+h+1);
        DrawMiniBox(dc,x-5,y+h+1);
    }

    dc->RestorePen();
}

/* ================================================ */
/* Paint                                            */
/* ------------------------------------------------ */
/* Paints the object to the printer dc.             */
/*                                                  */
/* input: TDC*   dc                                 */
/*        TRect&                                    */
/*        float                                     */
/* ================================================ */
void ObjectShape::Paint(TDC* dc, TRect& r, float s)
{  Draw(dc, r, FALSE, s);
}

void GetStringFromStream(istream& is, char **dibFilename)
{
  char c, buf[100];
  int i;

  is >> c; // Read '('
  for (i=0;c != ')';i++) {
     is >> c;
     buf[i] = c;
  }
  buf[i-1] = 0;

  free(*dibFilename);
  *dibFilename = (char *) malloc (sizeof(char)*strlen(buf)+1);
  strcpy(*dibFilename,buf);
}

/* ================================================ */
/* Input                                            */
/* ------------------------------------------------ */
/* Reads the object from the input stream is.       */
/*                                                  */
/* Input Format:                                    */
/*   shapetype, (dibFilename/label)                 */
/*   ,x,y,w,h,foreground,background,shadow,border,  */
/*   shadowDepth,displayMode,lineThickness,lineStyle,*/
/*   distinctId)                                    */
/*                                                  */
/* input:  istream& is                              */
/* output: istream&                                 */
/* ================================================ */
istream& ObjectShape::Input(istream& is)
{
    char c;
    int r,g,b;
```

```
    is >> c >> shapeType;

    if (shapeType == SH_BITMAP) {
      is >> c;
      GetStringFromStream(is,&dibFilename);
      try {Dib = new TDib(dibFilename); }
      catch(TGdiBase::TXGdi) {
         Dib = 0;
      }
    }

    if (shapeType == SH_TEXT) {
      is >> c;
      GetStringFromStream(is,&label);
    }

    is >> c >> x >> c >> y;
    is >> c >> w >> c >> h;

    is >> c >> r >> c >> g >> c >> b;
    foreground[R] = r; foreground[G] = g; foreground[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    background[R] = r; background[G] = g; background[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    shadow[R] = r;     shadow[G] = g;     shadow[B] = b;

    is >> c >> r >> c >> g >> c >> b;
    border[R] = r;     border[G] = g;     border[B] = b;

    is >> c >> r >> c >> g;
    shadowDepth = r;
    displayMode = g;

    is >> c >> r >> c >> g;
    lineThickness = r;
    lineStyle = g;

    is >> c >> distinctId;
    is >> c;

    return is;
}

/* ================================================ */
/* Output                                           */
/* ------------------------------------------------ */
/* Writes the object to the Output stream os.       */
/*                                                  */
/* Output Format:                                   */
/*   (OBJECT_TYPE_ID,                               */
/*   shapetype, (dibFilename/label)                 */
/*   x,y,w,h,foreground,background,shadow,border,   */
/*   shadowDepth,displayMode,lineThickness,lineStyle,*/
/*   distinctId)                                    */
/*                                                  */
/* input:  ostream& os                              */
/* output: ostream&                                 */
/* ================================================ */
ostream& ObjectShape::Output(ostream& os)
{
    char c = ',';

    os << '(' << (int) typeId;
    os << c << shapeType;
    if (shapeType == SH_BITMAP)
      os << c << "(" << dibFilename << ")";
    if (shapeType == SH_TEXT)
      os << c << "(" << label << ")";
    os << c << x << c << y;
    os << c << w << c << h;
    os << c << (int) foreground[R] << c << (int) foreground[G] << c << (int) foreground[B];
    os << c << (int) background[R] << c << (int) background[G] << c << (int) background[B];
    os << c << (int) shadow[R]     << c << (int) shadow[G]     << c << (int) shadow[B];
    os << c << (int) border[R]     << c << (int) border[G]     << c << (int) border[B];
    os << c << (int) shadowDepth   << c << (int) displayMode;
    os << c << (int) lineThickness << c << (int) lineStyle;
    os << c << SetDistinctId(*this);
    os << ')';

    return os;
}

/* ================================================ */
/* XYInObject                                       */
```

```
/* ------------------------------------------------ */
/* Tells whether the point px, py is in the object.  */
/*                                                  */
/* input:  int px, py                               */
/* output: BOOL                                     */
/* ================================================ */
BOOL ObjectShape::XYInObject(int px, int py)
{
    if (shapeType == SH_LINE) {
      return (PtOnLine (FALSE,px,py,x,y,w,h));
    } else {
      if (x <= px && px <= x+w &&
            y <= py && py <= y+h)
      return (TRUE);
    }
    return (FALSE);
}

/* ================================================ */
/* ObjectInRect                                     */
/* ------------------------------------------------ */
/* Tells whether the object is in the rectangle rect.*/
/*                                                  */
/* input:  TRect rect                               */
/* output: BOOL                                     */
/* ================================================ */
BOOL ObjectShape::ObjectInRect(TRect rect)
{
    if (shapeType == SH_LINE) {
      TRect objRect(x,y,w,h);
      return (rect.Contains(objRect));
    }
    TRect objRect(x,y,x+w,y+h);
    return (rect.Contains(objRect));
}
```

bttltool.h

```
/* ================================================ */
/* Basic Tool Class                                 */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS  */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

#if !defined(__bttltool_h)  // Use file only if it's not already included.
#define __bttltool_h

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include <owl\owlpch.h>
#include "bttlobj.h"
#include "bttldoc.h"

/* ================================================ */
/* Basic Tool Class Definitions                     */
/* ------------------------------------------------ */
class BasicTool {
public:
    int toolTypeId;
    int Id;

public:
    BasicTool(int ToolId);    // Constructor
    ~BasicTool();                 // Destructor

    // The Tool is notified when it is picked or
    // another is picked.
    virtual void Selected(TApplication*);
    virtual void DeSelected();

    // Change cursor when in window.
    virtual BOOL SetCursor(TWindow*, uint);

    // Event notification
    virtual void LeftMouseDown  (TWindow*, uint, TPoint&, BondGraphDocument&);
    virtual void LeftMouseUp    (TWindow*, uint modKeys, TPoint& point, BondGraphDocument& doc);
    virtual void MouseMove      (TWindow*, uint modKeys, TPoint& point,BondGraphDocument& doc);
    virtual void RightMouseDown (TWindow*, uint, TPoint&, BondGraphDocument&);
    virtual void CharTyped      (TWindow*, uint, uint, uint, BondGraphDocument&);
```

```
};

#endif
```

bttltool.cpp

```
/* ================================================ */
/* Basic Element Tool Class                         */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include "bttltool.h"
#include <windows.h>

/* ================================================ */
/* Class Support Function Definitions               */
/* ------------------------------------------------ */

/* ================================================ */
/* BasicTool                                        */
/* ------------------------------------------------ */
/* Create a new BasicTool.                          */
/*                                                  */
/* input: int toolId                                */
/* ================================================ */
BasicTool::BasicTool(int toolId)
{   Id = toolId;
}

/* ================================================ */
/* ~BasicTool                                       */
/* ------------------------------------------------ */
/* Frees memory used by BasicTool                   */
/*                                                  */
/* ================================================ */
BasicTool::~BasicTool()
{}

/* ================================================ */
/* Virtual Functions Which Have To Be Defined In    */
/* Subclasses.                                      */
/* ------------------------------------------------ */
/* Event Handlers                                   */
/* ================================================ */
void BasicTool::LeftMouseDown(TWindow*, uint, TPoint&, BondGraphDocument&) {}
void BasicTool::LeftMouseUp(TWindow*, uint, TPoint&, BondGraphDocument&) {}
void BasicTool::MouseMove(TWindow*, uint, TPoint&,BondGraphDocument&) {}
void BasicTool::CharTyped(TWindow*, uint, uint, uint, BondGraphDocument&) {}
void BasicTool::Selected(TApplication*) {}
void BasicTool::DeSelected() {}

/* ================================================ */
/* RightMouseDown                                   */
/* ------------------------------------------------ */
/* Open a settings dialog for the object.           */
/*                                                  */
/* ================================================ */
void BasicTool::RightMouseDown(TWindow* thewin, uint, TPoint& point,BondGraphDocument& doc)
{
    thewin->ReleaseCapture();
    // Check if Mouse is in an object
    BasicObject* topMostObject = NULL;
    BasicObject* curObject = NULL;

    doc.Start();
    while (curObject = doc.Current()) {
        if (curObject->XYInObject(point.x,point.y))
            topMostObject = curObject;
        doc.Next();
    }

    if (topMostObject) {
        topMostObject->SettingsDialog(thewin);
    }
    MessageBeep(-1);
}
```

```
/* ================================================= */
/* SetCursor                                         */
/* ------------------------------------------------- */
/*                                                   */
/* input: TWindow*, uint                             */
/* ================================================= */
BOOL BasicTool::SetCursor(TWindow*,uint)
{   return FALSE;
}
```

toolcomp.h

```
#if !defined(__toolcomp_h)
#define __toolelem_h

#include <owl\owlpch.h>
#include "bttlobj.h"
#include "bttldoc.h"
#include "bttltool.h"
#include "obj_comp.h"
#include "msoutlin.h"
#include "koutlin.h"

class CompoundTool : public BasicTool {
public:
    // Variables for object placement
    int ox, oy, Placing;
    TDC* BasicDC;

    TMemoryDC* OrigBack;      // For Dragging Object Image
    TBitmap*   OrigBackBmp;

    TMemoryDC* BasicCursor;   // Variables to Create Cursor
    TBitmap*   BasicCursorBmp;

    //TVbxOutline* outline;
    KOutline* outline;
    TRadioButton* compoundLink;

public:
    CompoundTool(int, /*TVbx*/KOutline* control = NULL, TRadioButton* compoundLink = NULL);
    ~CompoundTool();

    // Event Handlers
    void LeftMouseDown(TWindow*, uint, TPoint&, BondGraphDocument&);
    void MouseMove(TWindow*, uint modKeys, TPoint& point,BondGraphDocument& doc);
    void LeftMouseUp(TWindow*, uint modKeys, TPoint& point, BondGraphDocument& doc);
};

#endif
```

toolcomp.cpp

```
/* ================================================= */
/* Compound Tool Class                               */
/* ------------------------------------------------- */
/*                                                   */
/* ------------------------------------------------- */
/*   Code by Richard K. Branton - MIT co 94/95 EECS  */
/*                Copyright (c) 1995                 */
/*       Massachusetts Institute of Technology       */
/*         Mechanical Engineering Department         */
/* ================================================= */

/* ================================================= */
/* Include Files                                     */
/* ------------------------------------------------- */
#include "toolcomp.h"

#include "obj_elem.h"
#include "obj_comp.h"

#include "bttlview.h"

#include <windows.h>
#include <stdio.h>

/* ================================================= */
/* Class Support Function Definitions                */
/* ------------------------------------------------- */
void CreateFileNameAndPath (TVbxOutline* outline, char *filename, char *filepath);
```

```
/* ================================================ */
/* Basic Element Tool Class                         */
/* ------------------------------------------------ */

/* ================================================ */
/* BasicElemTool                                    */
/* ------------------------------------------------ */
/* creates a new Basic Element Tool of type typeId.  */
/*                                                  */
/* input: int   toolId, typeId                      */
/*        char *label                               */
/* ================================================ */
CompoundTool::CompoundTool(int theToolId, /*TVbx*/KOutline* control, TRadioButton* controlLink)
    : BasicTool(theToolId)
{
    BasicDC = 0;
    Placing = FALSE;
    outline = control;
    compoundLink = controlLink;
}

/* ================================================ */
/* ~BasicElemTool                                   */
/* ------------------------------------------------ */
/* frees variables used by the Basic Element.       */
/*                                                  */
/* ================================================ */
CompoundTool::~CompoundTool()
{
}

/* ================================================ */
/* LeftMouseDown                                    */
/* ------------------------------------------------ */
/* called when the mouse is pressed in the user     */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void CompoundTool::LeftMouseDown(TWindow* thewin, uint, TPoint& point, BondGraphDocument&)
{
    // If Not Already Placing An Object, Start Placing It.
    if (!Placing) {
        Placing = TRUE;
        thewin->SetCapture();

        // Initialize Variables Used
        BasicDC = new TClientDC(*thewin);

        // Create Cursor For Dragging Object In Window
        BasicCursorBmp = new TBitmap(40,40);
        BasicCursor = new TMemoryDC(*BasicDC);
        BasicCursor->SelectObject(*BasicCursorBmp);

        OrigBackBmp = new TBitmap(40,40);
        OrigBack = new TMemoryDC(*BasicDC);
        OrigBack->SelectObject(*OrigBackBmp);

        // Draw Border And Background For Cursor
        BasicCursor->SelectObject(TBrush(TColor::White));
        BasicCursor->Rectangle(TPoint(0,0),TSize(40,40));

        // Save Original Background Then Place Cursor
        ox = point.x;
        oy = point.y;

        // Save Back / Draw Cursor
        OrigBack->BitBlt(0,0,40,40,*BasicDC,ox-20,oy-20,SRCCOPY);
        BasicDC->BitBlt(ox-20,oy-20,40,40,*BasicCursor,0,0,SRCCOPY);
    }
}

/* ================================================ */
/* MouseMove                                        */
/* ------------------------------------------------ */
/* called when the mouse is moved in the user       */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
```

```
void CompoundTool::MouseMove(TWindow*, uint, TPoint& point,BondGraphDocument&)
{
    // If currently placing an object.
    if (Placing) {
        // Restore Original Background
        BasicDC->BitBlt(ox-20,oy-20,40,40,*OrigBack,0,0,SRCCOPY);

        ox = point.x;
        oy = point.y;

        // Save Back / Draw Cursor
        OrigBack->BitBlt(0,0,40,40,*BasicDC,ox-20,oy-20,SRCCOPY);
        BasicDC->BitBlt(ox-20,oy-20,40,40,*BasicCursor,0,0,SRCCOPY);
    }
}

/* ================================================ */
/* LeftMouseUp                                      */
/* ------------------------------------------------ */
/* called when the mouse is released in the user    */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
BOOL /*BondGraphDocument::*/ ReadFromifStream(ifstream* is, TObjectArray* theArray);
void CompoundTool::LeftMouseUp(TWindow* thewin, uint, TPoint& point, BondGraphDocument& thedoc)
{
    // If currently placing an object stop placing it.
    if (Placing) {
        Placing = FALSE;
        thewin->ReleaseCapture();

        BasicDC->BitBlt(ox-20,oy-20,40,40,*OrigBack,0,0,SRCCOPY);

        // Get Final Position And Place Object
        ox = point.x;
        oy = point.y;

        // Free Memory Used For Placement
        delete BasicDC;
        delete BasicCursor;
        delete BasicCursorBmp;
        delete OrigBack;
        delete OrigBackBmp;

        // Add Object To Document
        BasicObject* objectPtr = NULL;
        if (outline == NULL || compoundLink == NULL) { // Add Empty Compound Object
          objectPtr = new ObjectCompound();
          ((ObjectCompound*) objectPtr)->fileType = 1; // empty
          objectPtr->x = ox-20;
          objectPtr->y = oy-20;
          objectPtr->w = 40;
          objectPtr->h = 40;
        } else { // Add Compound Object From File
            char tmpS[101];
            string stringBuffer;
            int n;

            outline->GetPropText(stringBuffer);
            strcpy(tmpS,stringBuffer.c_str());

            if (strcmp(".bg",&(tmpS[strlen(tmpS)-3]))==0) {
                objectPtr = new ObjectCompound();
                ((ObjectCompound*) objectPtr)->fileType = 2; // external

                objectPtr->x = ox-20;
                objectPtr->y = oy-20;
                objectPtr->w = 40;
                objectPtr->h = 40;

                //CreateFileNameAndPath (outline, tmpS, buffer);
                outline->GetPropListIndex(n);
                outline->GetFullPathName(stringBuffer,n);
                stringBuffer = "models\\" + stringBuffer;

                ((ObjectCompound*)objectPtr)->filename = (char *) malloc
((strlen(stringBuffer.c_str())+1)*sizeof(char));
                strcpy(((ObjectCompound*)objectPtr)->filename,stringBuffer.c_str());

                objectPtr->label = (char *) malloc (sizeof(char)*(strlen(tmpS)+1));
                strcpy(objectPtr->label,tmpS);
                objectPtr->label[strlen(objectPtr->label)-3] = 0;
```

```
                if (compoundLink->GetCheck() != BF_CHECKED) {
                    // Copy file in
                    ((ObjectCompound*) objectPtr)->fileType = 1; // internal

                    // read from disk
                    ifstream* is = new ifstream(((ObjectCompound*)objectPtr)->filename,ios::out);

                    if (is) {
                      if (ReadFromifStream(is,objectPtr->subobjects)) {
                         MessageBeep(-1);
                      }
                    }

                    delete is;
                }
            }
        }

        if (objectPtr) {
          thedoc.AddObject(objectPtr);

          // Redraw Area With New Object
          thewin->InvalidateRect(TRect(TPoint(ox-20,oy-20),TSize(50,50)));
        }
    }
}
```

toolelem.h

```
/* ================================================ */
/* Basic Element Tool Class                         */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*               Copyright (c) 1995                 */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

#if !defined(__toolelem_h)  // Use file only if it's not already included.
#define __toolelem_h

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include <owl\owlpch.h>
#include "bttlobj.h"
#include "bttldoc.h"
#include "bttltool.h"

/* ================================================ */
/* Basic Element Tool Definitions                   */
/* ------------------------------------------------ */
class BasicElemTool : public BasicTool {
public:
    char ElementType;
    int  toolId;

    // Attributes
    char *label;

    // Variables for object placement
    int ox, oy, Placing;
    TDC* BasicDC;

    TMemoryDC* OrigBack;      // For Dragging Object Image
    TBitmap*   OrigBackBmp;

    // Display Variables
    TCursor*   TheCursor;

    TMemoryDC* BasicCursor;   // Variables to Create Cursor
    TBitmap*   BasicCursorBmp;

public:
    BasicElemTool(int toolId, int typeId, char* label = NULL);  // Constructor
    ~BasicElemTool();                                                                    //
Destructor

    // Selected/Deselected Functions
    void Selected(TApplication*);
    void DeSelected();

    // Cursor Manipulation
```

```
    BOOL SetCursor(TWindow*,uint);

    // Event Handlers
    void LeftMouseDown(TWindow*, uint, TPoint&, BondGraphDocument&);
    void MouseMove(TWindow*, uint modKeys, TPoint& point,BondGraphDocument& doc);
    void LeftMouseUp(TWindow*, uint modKeys, TPoint& point, BondGraphDocument& doc);

    // Add New Object To Document
    virtual void AddObject(int,int,BondGraphDocument&);
};

/* ================================================ */
/* Bond Element Tool Definition                     */
/* ------------------------------------------------ */
class BondTool : public BasicTool {
public:
    // Variables for Object Placement
    int sx,sy,dx,dy,Drawing,lastinobj;
    BasicObject *FromO, *ToO;
    TDC* BondDC;
    int bondType; // 0 = regular, 1 = power, 2 = causality, 3 = set power, 4 = set causality

public:
    BondTool(int toolId,int bT=0);     // Constructor

    // Event Handlers
    void LeftMouseDown(TWindow*, uint, TPoint&, BondGraphDocument&);
    void MouseMove(TWindow*, uint modKeys, TPoint& point,BondGraphDocument& doc);
    void LeftMouseUp(TWindow*, uint modKeys, TPoint& point, BondGraphDocument& doc);
};

#endif
```

toolelem.cpp

```
/* ================================================ */
/* Basic Element Tool Class                         */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include "toolelem.h"

#include "obj_elem.h"
#include "obj_comp.h"

#include "bttlview.h"

#include <windows.h>
//#include <stdlib.h>
//#include <stdio.h>

/* ================================================ */
/* Class Support Function Definitions               */
/* ------------------------------------------------ */
void SnapToGrid(POINT& pt, BondGraphDocument* doc);

/* ================================================ */
/* SetLabel                                         */
/* ------------------------------------------------ */
/* allocates memory and copies labelstring to label. */
/*                                                  */
/* input: char *labelstring, **label                */
/* ================================================ */
void SetLabel(char *labelstring, char **label)
{
    (*label) = (char *) malloc ((strlen(labelstring)+1)*sizeof(char));
    strcpy((*label),labelstring);
}

/* ================================================ */
/* Basic Element Tool Class                         */
/* ------------------------------------------------ */

/* ================================================ */
/* BasicElemTool                                    */
/* ------------------------------------------------ */
```

```
/* creates a new Basic Element Tool of type typeId.  */
/*                                                   */
/* input: int   toolId, typeId                       */
/*        char *label                                */
/* ================================================= */
BasicElemTool::BasicElemTool(int theToolId, int theTypeId, char *theLabel)
    : BasicTool(theToolId)
{
    // Identifiers
    ElementType = theTypeId;
    toolId = theToolId;

    // Label Creation
    switch (ElementType) {
        case BG_0JUNCTION:   SetLabel("0", &label);    break;
        case BG_1JUNCTION:   SetLabel("1", &label);    break;
        case BG_ESOURCE:     SetLabel("Se", &label);   break;
        case BG_FSOURCE:     SetLabel("Sf", &label);   break;
        case BG_RESISTOR:    SetLabel("R", &label);    break;
        case BG_CAPACITOR:   SetLabel("C", &label);    break;
        case BG_INDUCTOR:    SetLabel("I", &label);    break;
        case BG_TRANSFORMER: SetLabel("TF", &label);   break;
        case BG_GYRATOR:     SetLabel("GY", &label);   break;
    };

    BasicDC = 0;
    Placing = FALSE;

    TheCursor = 0;
}

/* ================================================= */
/* ~BasicElemTool                                    */
/* ------------------------------------------------- */
/* frees variables used by the Basic Element.        */
/*                                                   */
/* ================================================= */
BasicElemTool::~BasicElemTool()
{
    delete TheCursor;
    if (label)
        free(label);
}

/* ================================================= */
/* Selected                                          */
/* ------------------------------------------------- */
/* called when a tool is first selected in the       */
/* palette.                                          */
/*                                                   */
/* input: TApplication*                              */
/* ================================================= */
void BasicElemTool::Selected(TApplication* theapp)
{
    if (!TheCursor)
        TheCursor = new TCursor(theapp->GetInstance(),CURSOR_PLACE);
}

/* ================================================= */
/* DeSelected                                        */
/* ------------------------------------------------- */
/* called when a tool is deselected and another tool */
/* is selected in the palette.                       */
/*                                                   */
/* ================================================= */
void BasicElemTool::DeSelected()
{
}

/* ================================================= */
/* SetCursor                                         */
/* ------------------------------------------------- */
/* called when the mouse is moved in the user window */
/* to allow the tool to change the cursor.           */
/*                                                   */
/* input:  TWindow*, uint hitTest                    */
/* output: BOOL                                      */
/* ================================================= */
BOOL BasicElemTool::SetCursor(TWindow*, uint hitTest)
{
    if (hitTest == HTCLIENT)
        ::SetCursor(*TheCursor);
    return TRUE;
}

/* ================================================= */
/* LeftMouseDown                                     */
```

```
/* ------------------------------------------------ */
/* called when the mouse is pressed in the user     */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void BasicElemTool::LeftMouseDown(TWindow* thewin, uint, TPoint& point, BondGraphDocument& doc)
{
    // If Not Already Placing An Object, Start Placing It.
    if (!Placing) {
        Placing = TRUE;
        thewin->SetCapture();

        // Initialize Variables Used
        BasicDC = new TClientDC(*thewin);

        // Create Cursor For Dragging Object In Window
        BasicCursorBmp = new TBitmap(40,40);
        BasicCursor = new TMemoryDC(*BasicDC);
        BasicCursor->SelectObject(*BasicCursorBmp);

        OrigBackBmp = new TBitmap(40,40);
        OrigBack = new TMemoryDC(*BasicDC);
        OrigBack->SelectObject(*OrigBackBmp);

        // Draw Border And Background For Cursor
        BasicCursor->SelectObject(TBrush(TColor::White));
        BasicCursor->Rectangle(TPoint(0,0),TSize(40,40));

        // Draw Label For Cursor
        if (label) {
            HFONT hfont =
CreateFont(30,0,0,0,FW_BOLD,0,0,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,
                                                    PROOF_QUALITY,VARIABLE_PITCH,"Arial");
            TFont *theFont = new TFont(hfont);

            BasicCursor->SelectObject(*theFont);
            BasicCursor->SetBkMode(TRANSPARENT);
            BasicCursor->DrawText(label,-1,TRect(TPoint(0,0),TSize(40,40)),DT_CENTER | DT_VCENTER);

            delete theFont;
            DeleteObject(hfont);
        }

        SnapToGrid(point, &doc);

        // Save Original Background Then Place Cursor
        ox = point.x;
        oy = point.y;

        // Simulate Grid Temporarily
        // int rx = ox % 20;  if (rx < 10) ox -= rx; else   ox += rx;
        // int ry = oy % 20;  if (ry < 10) oy -= ry; else   oy += ry;

        // Save Back / Draw Cursor
        OrigBack->BitBlt(0,0,40,40,*BasicDC,ox-20,oy-20,SRCCOPY);
        BasicDC->BitBlt(ox-20,oy-20,40,40,*BasicCursor,0,0,SRCCOPY);
    }
}

/* ================================================ */
/* MouseMove                                        */
/* ------------------------------------------------ */
/* called when the mouse is moved in the user       */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void BasicElemTool::MouseMove(TWindow*, uint, TPoint& point,BondGraphDocument& doc)
{
    // If currently placing an object.
    if (Placing) {
        // Restore Original Background
        BasicDC->BitBlt(ox-20,oy-20,40,40,*OrigBack,0,0,SRCCOPY);

        SnapToGrid(point, &doc);
        ox = point.x;
        oy = point.y;

        // Simulate Grid Temporarily
        //int rx = ox % 20;   if (rx < 10) ox -= rx; else   ox += rx;
```

```
        //int ry = oy % 20;   if (ry < 10) oy -= ry; else   oy += ry;

        // Save Back / Draw Cursor
        OrigBack->BitBlt(0,0,40,40,*BasicDC,ox-20,oy-20,SRCCOPY);
        BasicDC->BitBlt(ox-20,oy-20,40,40,*BasicCursor,0,0,SRCCOPY);
    }
}

/* ================================================ */
/* LeftMouseUp                                      */
/* ------------------------------------------------ */
/* called when the mouse is released in the user    */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void BasicElemTool::LeftMouseUp(TWindow* thewin, uint, TPoint& point, BondGraphDocument& thedoc)
{
    // If currently placing an object stop placing it.
    if (Placing) {
        Placing = FALSE;
        thewin->ReleaseCapture();

        BasicDC->BitBlt(ox-20,oy-20,40,40,*OrigBack,0,0,SRCCOPY);

        // Get Final Position And Place Object
        SnapToGrid(point, &thedoc);
        ox = point.x;
        oy = point.y;

        // Simulate Grid Temporarily
        //int rx = ox % 20;   if (rx < 10) ox -= rx; else   ox += rx;
        //int ry = oy % 20;   if (ry < 10) oy -= ry; else   oy += ry;

        // Free Memory Used For Placement
        delete BasicDC;
        delete BasicCursor;
        delete BasicCursorBmp;
        delete OrigBack;
        delete OrigBackBmp;

        // Add Object To Document
        AddObject(ox-20,oy-20,thedoc);

        // Redraw Area With New Object
        thewin->InvalidateRect(TRect(TPoint(ox-20,oy-20),TSize(50,50)));
    }
}

/* ================================================  */
/* AddObject                                         */
/* ------------------------------------------------  */
/* called to add a new object to the BondGraphDocument*/
/* window.                                           */
/*                                                   */
/* input:  int x, y                                  */
/*         BondGraphDocument&                        */
/* ================================================  */
void BasicElemTool::AddObject(int x, int y, BondGraphDocument& theDoc)
{
    BasicObject* objectPtr;

    switch (ElementType) {
        case BG_0JUNCTION:   objectPtr = new Object0Junction();   break;
        case BG_1JUNCTION:   objectPtr = new Object1Junction();   break;
        case BG_ESOURCE:     objectPtr = new ObjectESource();     break;
        case BG_FSOURCE:     objectPtr = new ObjectFSource();     break;
        case BG_RESISTOR:    objectPtr = new ObjectResistor();   break;
        case BG_CAPACITOR:   objectPtr = new ObjectCapacitor();   break;
        case BG_INDUCTOR:    objectPtr = new ObjectInductor();    break;
        case BG_TRANSFORMER: objectPtr = new ObjectTransformer(); break;
        case BG_GYRATOR:     objectPtr = new ObjectGyrator();     break;
    };

    objectPtr->x = x;
    objectPtr->y = y;
    objectPtr->w = 40;
    objectPtr->h = 40;
    objectPtr->shadowDepth = 2;

    theDoc.AddObject(objectPtr);
}

/* ================================================ */
```

```
/* Bond Element Tool Definition                          */
/* ----------------------------------------------------- */

/* ===================================================== */
/* BondTool                                              */
/* ----------------------------------------------------- */
/* creates a new Basic Element Tool of type typeId.      */
/*                                                       */
/* input: int   toolId                                   */
/* ===================================================== */
BondTool::BondTool(int toolId, int bT)
    : BasicTool(toolId)
{
    Drawing = FALSE;

    BondDC = NULL;
    FromO = NULL;
    ToO = NULL;
    bondType = bT;
}

/* ===================================================== */
/* LeftMouseDown                                         */
/* ----------------------------------------------------- */
/* called when the mouse is pressed in the user          */
/* window.                                               */
/*                                                       */
/* input:  TWindow* thewin                               */
/*         uint                                          */
/*         TPoint&                                       */
/*         BondGraphDocument&                            */
/* ===================================================== */
#include <math.h>
float distance (int x1, int y1, int x2, int y2)
{
  return sqrt((0.0 + x2-x1)*(0.0 + x2-x1)+(0.0 + y2-y1)*(0.0 + y2-y1));
}

void BondTool::LeftMouseDown(TWindow* thewin, uint, TPoint& point, BondGraphDocument& doc)
{
    // If Not Adding a Bond start adding it now.
    if (!Drawing) {
        sx = dx = point.x;
        sy = dy = point.y;

        // Check if Mouse is in an object
        BasicObject* topMostObject = NULL;
        BasicObject* curObject = NULL;

        // Find out which object if any, the mouse was pressed in
        doc.Start();
        while (curObject = doc.Current()) {
            if (curObject->XYInObject(sx,sy))
                topMostObject = curObject;
            doc.Next();
        }

        if (bondType == 3 || bondType == 4) {
            if (topMostObject && topMostObject->typeId == BG_BOND) {
              // Find out if clicked closer to from or to object

              MessageBeep(-1);
              if (bondType == 3) {// Set Power
                 if (distance(point.x,point.y,((ObjectBond*)topMostObject)-
>x1,((ObjectBond*)topMostObject)->y1)<
                       distance(point.x,point.y,((ObjectBond*)topMostObject)-
>x2,((ObjectBond*)topMostObject)->y2)) {
                    if (((ObjectBond*)topMostObject)->Power == 0)
                      ((ObjectBond*)topMostObject)->Power = 2;
                    else ((ObjectBond*)topMostObject)->Power = 0;
                 } else {
                    if (((ObjectBond*)topMostObject)->Power == 1)
                        ((ObjectBond*)topMostObject)->Power = 2;
                    else ((ObjectBond*)topMostObject)->Power = 1;
                 }

              } else if (bondType == 4) { // Set Causality
                 if (distance(point.x,point.y,((ObjectBond*)topMostObject)-
>x1,((ObjectBond*)topMostObject)->y1)<
                       distance(point.x,point.y,((ObjectBond*)topMostObject)-
>x2,((ObjectBond*)topMostObject)->y2)) {
                    if (((ObjectBond*)topMostObject)->Causality == 0)
                      ((ObjectBond*)topMostObject)->Causality = 2;
                    else ((ObjectBond*)topMostObject)->Causality = 0;
                 } else {
                    if (((ObjectBond*)topMostObject)->Causality == 1)
                        ((ObjectBond*)topMostObject)->Causality = 2;
```

```
                    else ((ObjectBond*)topMostObject)->Causality = 1;
                }
            }

            thewin->Invalidate(); //TRect(((ObjectBond*)topMostObject)-
>x1,((ObjectBond*)topMostObject)->y1,
                                                //  ((ObjectBond*)topMostObject)-
>x2,((ObjectBond*)topMostObject)->y2));
            }
        } else
        if (topMostObject && topMostObject->typeId != BG_BOND) {
            Drawing = TRUE;
            thewin->SetCapture();
            FromO = topMostObject;
            FromO->distinctId = 1;

            sx = FromO->x + FromO->w/2;
            sy = FromO->y + FromO->h/2;

            // Setup Drawing Stuff And Draw Line
            BondDC = new TClientDC(*thewin);
            BondDC->SetROP2(R2_NOT);

            lastinobj = FALSE;
            BondDC->SelectObject(TBrush(TColor::Black));
            BondDC->Ellipse(sx-5,sy-5,sx+5,sy+5);
            BondDC->MoveTo(sx,sy);
            BondDC->LineTo(dx,dy);
            BondDC->Rectangle(dx-5,dy-5,dx+5,dy+5);
        }
    }
}

/* ================================================ */
/* MouseMove                                        */
/* ------------------------------------------------ */
/* called when the mouse is moved in the user       */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void BondTool::MouseMove(TWindow*, uint, TPoint& point,BondGraphDocument& doc)
{
    // If Currently Drawing
    if (Drawing) {
        // Erase Object
        BondDC->SelectObject(TBrush(TColor::Black));
        BondDC->Ellipse(sx-5,sy-5,sx+5,sy+5);
        BondDC->MoveTo(sx,sy);
        BondDC->LineTo(dx,dy);
        if (lastinobj)
            BondDC->Ellipse(dx-5,dy-5,dx+5,dy+5);
        else BondDC->Rectangle(dx-5,dy-5,dx+5,dy+5);

        // Get New Position
        dx = point.x;
        dy = point.y;

        // Check if Mouse is in an object
        // If So Draw an Ellipse Instead of a Box
        BasicObject* topMostObject = NULL;
        BasicObject* curObject = NULL;

        doc.Start();
        while (curObject = doc.Current()) {
            if (curObject->XYInObject(dx,dy)) {
                topMostObject = curObject;
                break;
            }
            doc.Next();
        }
        lastinobj = (topMostObject != NULL) &&
                        (topMostObject->distinctId != 1) &&
                        (topMostObject->typeId != BG_BOND);

        // Draw New Object
        BondDC->Ellipse(sx-5,sy-5,sx+5,sy+5);
        BondDC->MoveTo(sx,sy);
        BondDC->LineTo(dx,dy);
        if (lastinobj)
              BondDC->Ellipse(dx-5,dy-5,dx+5,dy+5);
        else BondDC->Rectangle(dx-5,dy-5,dx+5,dy+5);
    }
}
```

```
/* ================================================ */
/* LeftMouseUp                                      */
/* ------------------------------------------------ */
/* called when the mouse is released in the user    */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void BondTool::LeftMouseUp(TWindow* thewin, uint, TPoint& point, BondGraphDocument& doc)
{
    // If Currently Drawing
    if (Drawing) {
        Drawing = FALSE;
        thewin->ReleaseCapture();
        FromO->distinctId = -1;

        // Erase Object
        BondDC->SelectObject(TBrush(TColor::Black));

        BondDC->Ellipse(sx-5,sy-5,sx+5,sy+5);
        BondDC->MoveTo(sx,sy);
        BondDC->LineTo(dx,dy);
        if (lastinobj)
            BondDC->Ellipse(dx-5,dy-5,dx+5,dy+5);
        else BondDC->Rectangle(dx-5,dy-5,dx+5,dy+5);

        // Place Final Object
        dx = point.x;
        dy = point.y;

        // Check if Mouse is in an object
        BasicObject* topMostObject = NULL;
        BasicObject* curObject = NULL;

        doc.Start();
        while (curObject = doc.Current()) {
            if (curObject->XYInObject(dx,dy))
                topMostObject = curObject;
            doc.Next();
        }

        if (topMostObject) {
            ToO = topMostObject;
            if (FromO != ToO && ToO->typeId != BG_BOND && ToO->typeId != BG_SHAPE) {
                // Later Need To Fix Ordering To Keep Bonds Above All Others In Array
                ObjectBond* objectPtr = new ObjectBond(FromO,ToO);
                doc.AddObject(objectPtr);

                if (bondType == 1) { // Set Power
                  objectPtr->Power = 1; // 'To' Object
                } else if (bondType == 2) { // Set Causality
                  objectPtr->Causality = 0; // 'From' Object
                }

                // Only Redraw Area Between Two Objects
                int minx = FromO->x; if (ToO->x < minx) minx = ToO->x;
                int maxx = FromO->x + FromO->w; if (ToO->x + ToO->w > maxx) maxx = ToO->x + ToO->w;
                int miny = FromO->y; if (ToO->y < miny) miny = ToO->y;
                int maxy = FromO->y + FromO->h; if (ToO->y + ToO->h > maxy) maxy = ToO->y + ToO->h;

                thewin->InvalidateRect(TRect(minx,miny,maxx,maxy));
            }
        }

        delete BondDC;
        ToO = FromO = NULL;
        BondDC = NULL;
    }
}
```

toolslct.h

```
/* ================================================ */
/* Selection Tool Class                             */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*              Copyright (c) 1995                  */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
```

```
/* ================================================ */

#if !defined(__toolslct_h)  // Use file only if it's not already included.
#define __toolslct_h

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include <owl\owlpch.h>
#include "bttlobj.h"
#include "bttldoc.h"
#include "bttltool.h"

/* ================================================ */
/* Selection Tool Defines                           */
/* ------------------------------------------------ */
#define NOTHING  0
#define SELECTING1
#define MOVING       2

/* ================================================ */
/* Selection Tool Definitions                       */
/* ------------------------------------------------ */
class SelectionTool : public BasicTool {
public:
    int  SelectMode;

    // Selection/Move Bounds
    int  sx,sy,dx,dy;
    int  ox,oy,minx,miny,maxx,maxy;

    TRect oRect, nRect;
    TCursor* TheCursor;
    TDC* SelectDC;

    // Keep Track Of Document and Window For Deselection
    BondGraphDocument* docptr;
    TWindow* winptr;

    // For moving single object vs group of objects
    BasicObject* theObject;

public:
    SelectionTool(int id);    // Constructor
    ~SelectionTool();         // Destructor

    // The Tool is notified when it is picked or
    // another is picked.
    void Selected(TApplication*);
    void DeSelected();

    // Change cursor when in window.
    BOOL SetCursor(TWindow*,uint);

    // Event notification
    void LeftMouseDown(TWindow*, uint, TPoint&, BondGraphDocument&);
        // Start Selection
    void MouseMove(TWindow*, uint modKeys, TPoint& point,BondGraphDocument& doc);
                        // Update Selection
    void LeftMouseUp(TWindow*, uint modKeys, TPoint& point, BondGraphDocument& doc);  // End Selection
};

#endif
```

toolslct.cpp

```
/* ================================================ */
/* Basic Element Tool Class                         */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*               Copyright (c) 1995                 */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include "toolslct.h"

#include "bttlobj.h"
#include "obj_elem.h"
#include "obj_comp.h"
```

```
#include "obj_shap.h"
#include "bttlview.h"
#include <windows.h>
#include <stdlib.h>
//#include <string.h>
#include <stdio.h>

// Need to fix selection in a scrolled window.

/* ================================================ */
/* Class Support Function Definitions               */
/* ------------------------------------------------ */
void SnapToGrid(POINT& pt, BondGraphDocument* doc);

/* ================================================ */
/* DrawRectangle                                    */
/* ------------------------------------------------ */
/*                                                  */
/* input: TDC* theDC, int x1,y1,x2,y2               */
/* ================================================ */
void DrawRectangle(TDC* theDC, int x1, int y1, int x2, int y2)
{
    theDC->MoveTo(x1,y1); theDC->LineTo(x2,y1); theDC->LineTo(x2,y2);
    theDC->LineTo(x1,y2); theDC->LineTo(x1,y1);
}

/* ================================================ */
/* Basic Element Tool Class                         */
/* ------------------------------------------------ */

/* ================================================ */
/* SelectionTool                                    */
/* ------------------------------------------------ */
/*                                                  */
/* input: int id                                    */
/* ================================================ */
SelectionTool::SelectionTool(int id)
    : BasicTool(id)
{
    SelectMode = NOTHING;
    TheCursor = NULL;
}

/* ================================================ */
/* ~SelectionTool                                   */
/* ------------------------------------------------ */
/*                                                  */
/* ================================================ */
SelectionTool::~SelectionTool()
{   delete TheCursor;
}

/* ================================================ */
/* Selected                                         */
/* ------------------------------------------------ */
/*                                                  */
/* ================================================ */
void SelectionTool::Selected(TApplication* theapp)
{
    if (!TheCursor)
        TheCursor = new TCursor(theapp->GetInstance(),CURSOR_PLACE);
    docptr = 0;
}

/* ================================================ */
/* DeSelected                                       */
/* ------------------------------------------------ */
/*                                                  */
/* ================================================ */
void SelectionTool::DeSelected()
{
    // If Objects have been selected, unselect them and
    // Redraw window.
    if (docptr && winptr) {
        BasicObject* curObject;
        docptr->Start();
        while (curObject = docptr->Current()) {
            curObject->selected = FALSE;
            docptr->Next();
        }
        winptr->Invalidate();
    }
    docptr = NULL;
    winptr = NULL;
}

/* ================================================ */
```

```
/* SetCursor                                          */
/* -------------------------------------------------- */
/*                                                    */
/* input: TWindow*, uint hitTest                      */
/* ================================================== */
BOOL SelectionTool::SetCursor(TWindow*, uint hitTest)
{
    if (SelectMode == NOTHING)
        return FALSE;

    if (hitTest == HTCLIENT)
        ::SetCursor(*TheCursor);

    return TRUE;
}

/* ================================================== */
/* LeftMouseDown                                      */
/* -------------------------------------------------- */
/* called when the mouse is pressed in the user       */
/* window.                                            */
/*                                                    */
/* input:  TWindow* thewin                            */
/*         uint                                       */
/*         TPoint&                                    */
/*         BondGraphDocument&                         */
/* ================================================== */
void SelectionTool::LeftMouseDown(TWindow* thewin, uint modKeys, TPoint& point, BondGraphDocument&
doc)
{
    // If Not Selecting, Then Start
    if (SelectMode == NOTHING) {
        // If Mouse is not pressed inside an object, then start drawing a
        // selection rectangle.
        //
        // If Mouse is pressed inside an object, we could either be about
        // to select or move the object or maybe both.

        minx = miny = 20000;
        maxx = maxy = 0;

        sx = dx = point.x;
        sy = dy = point.y;

        // Check if Mouse is in an object
        BasicObject* topMostObject = NULL;
        BasicObject* curObject = NULL;

        doc.Start();
        while (curObject = doc.Current()) {
            if (curObject->XYInObject(sx,sy))
                topMostObject = curObject;

            if (curObject->selected) {
                if (curObject->x < minx) minx = curObject->x;
                if (curObject->y < miny) miny = curObject->y;
                if (curObject->x + curObject->w > maxx)
                    maxx = curObject->x + curObject->w;
                if (curObject->y + curObject->h > maxy)
                    maxy = curObject->y + curObject->h;
            }
            doc.Next();
        }

        // Setup Selection DC
        SelectDC = new TClientDC(*thewin);

        if (topMostObject == NULL || modKeys == MK_CONTROL) {
            // Start drawing selection bounds.
            SelectMode = SELECTING;

            SelectDC->SetROP2(R2_NOT);
            DrawRectangle(SelectDC,sx,sy,dx,dy);
        } else {
            // If mouse in unselected object, select it and start moving otherwise
            // just start moving.
            SelectMode = MOVING;

            if (topMostObject->selected == FALSE) {
                // Deselect old objects
                theObject = topMostObject;

                minx = topMostObject->x;
                maxx = minx + topMostObject->w;
                miny = topMostObject->y;
                maxy = miny + topMostObject->h;
            }
```

```
            maxx -= minx; // Get Selection Width and Height
            maxy -= miny;

            ox = sx - point.x + minx;
            oy = sy - point.y + miny;

            SnapToGrid(point,&doc);
            minx = point.x - minx;
            miny = point.y - miny;

            SelectDC->SetROP2(R2_NOT);
            DrawRectangle(SelectDC,sx-minx,sy-miny,sx-minx+maxx,sy-miny+maxy);

            if (TheCursor)
                ::SetCursor(*TheCursor);
        }

        // Grab Mouse
        thewin->SetCapture();
    }
}

/* ================================================ */
/* MouseMove                                        */
/* ------------------------------------------------ */
/* called when the mouse is moved in the user       */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void SelectionTool::MouseMove(TWindow*, uint, TPoint& point,BondGraphDocument& doc)
{
    if (SelectMode == MOVING) {
        // Erase Old Bounds and Redraw New Bounds
        DrawRectangle(SelectDC,sx-minx,sy-miny,sx-minx+maxx,sy-miny+maxy);
        SnapToGrid(point,&doc);
        sx = point.x;
        sy = point.y;
        DrawRectangle(SelectDC,sx-minx,sy-miny,sx-minx+maxx,sy-miny+maxy);

    } else if (SelectMode == SELECTING) {
        // Erase and Redraw Bounds
        DrawRectangle(SelectDC,sx,sy,dx,dy);
        dx = point.x;
        dy = point.y;
        DrawRectangle(SelectDC,sx,sy,dx,dy);
    }
}

/* ================================================ */
/* LeftMouseUp                                      */
/* ------------------------------------------------ */
/* called when the mouse is released in the user    */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void SelectionTool::LeftMouseUp(TWindow* win, uint modKeys, TPoint& point, BondGraphDocument& doc)
{
  if (SelectMode != NOTHING) {
    win->ReleaseCapture();
    BOOL found = FALSE; // Only Redraw If Something Selected

    // Get offset to tell difference between moving and selecting
    int nx = (point.x - minx) - ox;
    int ny = (point.y - miny) - oy;

    // Erase original rectangles
    if (SelectMode == MOVING)
        DrawRectangle(SelectDC,sx-minx,sy-miny,sx-minx+maxx,sy-miny+maxy);
    else DrawRectangle(SelectDC,sx,sy,dx,dy);

    // If selecting or moving but only a small distance
    if (SelectMode == SELECTING ||
        (SelectMode == MOVING && ((abs(nx) < 3 && abs(ny) < 3) || modKeys == MK_CONTROL))) {

        dx = point.x;
        dy = point.y;

        // ====================
```

```
    if (abs(sx - dx) <= 5 && abs(sy - dy) <= 5) {
        // Select Single Object
        BasicObject* topMostObject = NULL;
        BasicObject* curObject;
        doc.Start();
        while (curObject = doc.Current()) {
            if (curObject->XYInObject(sx,sy)==TRUE)
                topMostObject = curObject;

            if (modKeys != MK_CONTROL)
                if (curObject->selected) {
                    found = TRUE;
                    curObject->selected = FALSE;
                }

            doc.Next();
        }

        if (topMostObject) {
            if (modKeys == MK_CONTROL && topMostObject->selected == TRUE) {
                topMostObject->selected = FALSE;
                found = TRUE;
            } else {
                if (topMostObject->selected == FALSE)
                    found = TRUE;
                topMostObject->selected = TRUE;
            }
        }
    } else {
        TRect selRect(sx,sy,dx,dy);
        selRect.Normalize();
        // Select Objects in Selection Rectangle
        doc.Start();
        while (doc.Current()) {
            if (doc.Current()) {
                if (doc.Current()->ObjectInRect(selRect)==TRUE) {
                    if (doc.Current()->selected == FALSE)
                        found = true;
                    doc.Current()->selected = TRUE;
                } else if (modKeys != MK_CONTROL) {
                    if (doc.Current()->selected == TRUE)
                        found = true;
                    doc.Current()->selected = FALSE;
                }
            }
            doc.Next();
        }
    }

} else {
    BasicObject* curObject = NULL;
    if (theObject)
      if (theObject->selected == FALSE) {
        doc.Start();
        while (curObject=doc.Current()) {
            curObject->selected = FALSE;
            doc.Next();
        }
        theObject->selected = TRUE;
    }

    doc.Start();
    while (curObject=doc.Current()) {
        if (curObject->selected)
            curObject->MoveBy(nx,ny);
        doc.Next();
    }

    found = TRUE;
    // Notify Doc about movement.
}

SelectMode = NOTHING;
//if (found)
win->Invalidate();
delete SelectDC;
SelectDC = 0;

BasicObject* curObject;
doc.Start();
doc.inSelected = FALSE;
while (curObject = doc.Current()) {
  if (curObject->selected) {
      doc.inSelected = TRUE;
      docptr = &doc;
      winptr = win;
```

```
            return;
          }
          doc.Next();
        }
      }
    }
```

toolshap.h

```
/* ================================================ */
/* Shape Tool Class                                 */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

#if !defined(__toolshap_h)  // Use file only if it's not already included.
#define __toolshap_h

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include <owl\owlpch.h>
#include "bttlobj.h"
#include "obj_shap.h"
#include "bttldoc.h"
#include "bttltool.h"

/* ================================================ */
/* Shape Tool Definition                            */
/* ------------------------------------------------ */
class ShapeTool : public BasicTool {
public:
    char ShapeType;
    int  toolId;

    // Variables for object placement
    int sx,sy, dx,dy, Placing;
    TDC* BasicDC;
    BondGraphDocument* whichDoc;
    TWindow* whichWin;

    int Typing;
    char text[50];

    int   cursorPos;

    // Display Variables
    TCursor*   TheCursor;
    TMemoryDC* BasicCursor;   // Variables to Create Cursor
    TBitmap*   BasicCursorBmp;

    int origW, origH;
    TBitmap*   origBmp;
    TMemoryDC* origDC;

    void PlaceLabel(int,int,char*,int shownew = TRUE);

public:
    ShapeTool(int toolId, int typeId);// Constructor
    ~ShapeTool();                                  // Destructor

    // Selected/Deselected Functions
    void Selected(TApplication*);
    void DeSelected();

    // Cursor Manipulation
    BOOL SetCursor(TWindow*,uint);

    // Event Handlers
    void LeftMouseDown(TWindow*, uint, TPoint&, BondGraphDocument&);
    void MouseMove(TWindow*, uint modKeys, TPoint& point,BondGraphDocument& doc);     // Follow Mouse
Snapping To Grid
    void LeftMouseUp(TWindow*, uint modKeys, TPoint& point, BondGraphDocument& doc);
    void CharTyped      (TWindow*, uint, uint, uint, BondGraphDocument&);
};

#endif
```

toolshap.cpp

```
/* ================================================ */
/* Shape Tool Class                                 */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*               Copyright (c) 1995                 */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include "toolshap.h"

#include "obj_elem.h"
#include "obj_shap.h"

#include "bttlview.h"

#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/* ================================================ */
/* Class Support Function Definitions               */
/* ------------------------------------------------ */
void DrawRectangle(TDC* theDC, int x1, int y1, int x2, int y2);
void SetLabel(char *labelstring, char **label);
void SnapToGrid(POINT& pt, BondGraphDocument* doc);

/* ================================================ */
/* DrawLabel                                        */
/* ------------------------------------------------ */
/*                                                  */
/* input: TDC* theDC, int x1,y1,x2,y2               */
/* ================================================ */
void DrawLabel(TDC* theDC, int x1, int y1, char *text)
{
  HFONT hfont = CreateFont(30,0,0,0,FW_BOLD,0,0,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,
                                                  PROOF_QUALITY,VARIABLE_PITCH,"Arial");
  TFont *theFont = new TFont(hfont);
  theDC->SelectObject(*theFont);
  theDC->SetBkMode(TRANSPARENT);
  theDC->DrawText(text,-1,TRect(TPoint(x1,y1),TSize(40,40)),DT_LEFT | DT_NOCLIP);
  delete theFont;
  DeleteObject(hfont);
}

/* ================================================ */
/* LengthLabel                                      */
/* ------------------------------------------------ */
/*                                                  */
/* input: TDC* theDC, char *text                    */
/* ================================================ */
TSize LengthLabel(TDC* theDC, char *text)
{
  char ss[100];
  sprintf(ss,"%s  ",text);
  HFONT hfont = CreateFont(30,0,0,0,FW_BOLD,0,0,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,
                                                  PROOF_QUALITY,VARIABLE_PITCH,"Arial");
  TFont *theFont = new TFont(hfont);
  theDC->SelectObject(*theFont);
  theDC->SetBkMode(TRANSPARENT);
  TSize sz(theDC->GetTextExtent(ss,strlen(ss)));
  delete theFont;
  DeleteObject(hfont);
  return sz;
}

/* ================================================ */
/* Shape Tool Class                                 */
/* ------------------------------------------------ */

/* ================================================ */
/* ShapeTool                                        */
/* ------------------------------------------------ */
/* creates a new Shape Tool of type typeId.         */
/*                                                  */
/* input: int   toolId, typeId                      */
/* ================================================ */
ShapeTool::ShapeTool(int theToolId, int theTypeId)
    : BasicTool(theToolId)
{
```

```
    // Identifiers
    ShapeType = theTypeId;
    toolId = theToolId;

    BasicDC = 0;
    Placing = FALSE;
    Typing = FALSE;

    TheCursor = 0;
}

/* ================================================ */
/* ~ShapeTool                                       */
/* ------------------------------------------------ */
/* frees variables used by the Shape Tool.          */
/*                                                  */
/* ================================================ */
ShapeTool::~ShapeTool()
{
    delete TheCursor;
    delete origBmp;
    delete origDC;
}

/* ================================================ */
/* Selected                                         */
/* ------------------------------------------------ */
/* called when a tool is first selected in the      */
/* palette.                                         */
/*                                                  */
/* input: TApplication*                             */
/* ================================================ */
void ShapeTool::Selected(TApplication* theapp)
{
//if (!TheCursor)
//  TheCursor = new TCursor(theapp->GetInstance(),CURSOR_PLACE);
  origBmp = NULL;
  Typing = FALSE;
}

/* ================================================ */
/* DeSelected                                       */
/* ------------------------------------------------ */
/* called when a tool is deselected and another tool */
/* is selected in the palette.                      */
/*                                                  */
/* ================================================ */
void ShapeTool::DeSelected()
{
  if (Typing) {
    if (strlen(text) > 1) {
      BasicObject* objectPtr;
      objectPtr = new ObjectShape(SH_TEXT);
      objectPtr->x = sx;
      objectPtr->y = sy;
      objectPtr->w = 40;
      objectPtr->h = 40;
      text[strlen(text)-1] = 0;
      ((ObjectShape*)objectPtr)->label = (char *) malloc (strlen(text)*sizeof(char) + 1);
      strcpy(((ObjectShape*)objectPtr)->label,text);
      whichDoc->AddObject(objectPtr);
    }
    delete origDC;
    delete BasicDC;
    whichWin->Invalidate();
  }
}

/* ================================================ */
/* SetCursor                                        */
/* ------------------------------------------------ */
/* called when the mouse is moved in the user window */
/* to allow the tool to change the cursor.          */
/*                                                  */
/* input:  TWindow*, uint hitTest                   */
/* output: BOOL                                     */
/* ================================================ */
BOOL ShapeTool::SetCursor(TWindow*, uint hitTest)
{
//  if (hitTest == HTCLIENT)
//      ::SetCursor(*TheCursor);
//  return TRUE;
  return FALSE;
}

/* ================================================ */
/* CharTyped                                        */
```

```
/* ------------------------------------------------ */
/* called when the mouse is pressed in the user     */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint key, repeatCount, flags             */
/*         BondGraphDocument&                       */
/* ================================================ */
void ShapeTool::CharTyped (TWindow* thewin, uint key, uint repeatCount, uint flags,
BondGraphDocument&)
{
    if (Typing) {
      int len = strlen(text);
      if (key == 8) {
         if (len > 1) {
            text[len-2] = '_';
            text[len-1] = 0;
         }
      } else if (key >= ' ') {
         text[len-1] = key;
         text[len] = '_';
         text[len+1] = 0;
      }
      PlaceLabel(sx,sy,text);
    }
}

void ShapeTool::PlaceLabel(int tx, int ty, char *text, int shownew)
{
    // place old and free memory
    if (origBmp != NULL) {
        BasicDC->BitBlt(tx,ty,origW-10,origH,*origDC,0,0,SRCCOPY);
    }

    // get new
    if (shownew) {
      TSize sz(LengthLabel(BasicDC, text));
      origW = sz.cx + 10; origH = sz.cy;

      delete origBmp;
      origBmp = new TBitmap(origW,origH);
      origDC->SelectObject(*origBmp);
      origDC->BitBlt(0,0,origW,origH,*BasicDC,tx,ty,SRCCOPY);

      // place new
      DrawLabel(BasicDC, tx, ty, text);
    }
}

/* ================================================ */
/* LeftMouseDown                                    */
/* ------------------------------------------------ */
/* called when the mouse is pressed in the user     */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void ShapeTool::LeftMouseDown(TWindow* thewin, uint, TPoint& point, BondGraphDocument& theDoc)
{
    // If Not Already Placing An Object, Start Placing It.
    if (ShapeType == SH_TEXT) {
        if (Typing) {
            if (strlen(text) > 1) {
              BasicObject* objectPtr;
              objectPtr = new ObjectShape(SH_TEXT);
              objectPtr->x = sx;
              objectPtr->y = sy;
              objectPtr->w = 40;
              objectPtr->h = 40;
              text[strlen(text)-1] = 0;
              ((ObjectShape*)objectPtr)->label = (char *) malloc (strlen(text)*sizeof(char) + 1);
              strcpy(((ObjectShape*)objectPtr)->label,text);
              theDoc.AddObject(objectPtr);
              PlaceLabel(sx,sy,text,TRUE);
            } else
              PlaceLabel(sx,sy,text,FALSE);
            delete origDC;
            delete BasicDC;
        }

        Placing = TRUE;
        Typing = TRUE;

        whichDoc = &theDoc;
```

```
        whichWin = thewin;
        BasicDC = new TClientDC(*thewin);
        SnapToGrid(point, &theDoc);
        sx = point.x;
        sy = point.y;

        origBmp = NULL;
        origDC = NULL;

        origDC = new TMemoryDC(*BasicDC);

        text[0] = '_';
        text[1] = 0;
        PlaceLabel(sx,sy,text);
    } else
    if (ShapeType == SH_BITMAP) {
      MessageBeep(-1);
      TOpenSaveDialog::TData data (OFN_HIDEREADONLY | OFN_FILEMUSTEXIST | OFN_NOREADONLYRETURN,
                                   "Bitmap files (*.BMP)|*.bmp|", 0, "", "BMP");
      if (TFileOpenDialog(thewin, data).Execute() == IDOK) {
         char filename[100];
         TOpenSaveDialog::GetFileTitle(data.FileName, filename, 100);
//       if (ReadBitmap(filename))
         {
            BasicObject* objectPtr;
            objectPtr = new ObjectShape(SH_BITMAP);
            SnapToGrid(point, &theDoc);
            objectPtr->x = point.x;
            objectPtr->y = point.y;
            objectPtr->w = 40;
            objectPtr->h = 40;
            ((ObjectShape*)objectPtr)->dibFilename = (char *) malloc (strlen(filename)*sizeof(char) +
1);
            strcpy(((ObjectShape*)objectPtr)->dibFilename,filename);
            try {((ObjectShape*)objectPtr)->Dib = new TDib(filename); }
            catch(TGdiBase::TXGdi) {
              ((ObjectShape*)objectPtr)->Dib = 0;
            }

            theDoc.AddObject(objectPtr);
            thewin->Invalidate();
         }
      }

    } else

    if (!Placing) {
        Placing = TRUE;
        thewin->SetCapture();

        // Initialize Variables Used
        BasicDC = new TClientDC(*thewin);

        SnapToGrid(point, &theDoc);
        sx = dx = point.x;
        sy = dy = point.y;

        BasicDC->SetROP2(R2_NOT);

        switch (ShapeType) {
            case SH_LINE:      BasicDC->MoveTo(sx,sy); BasicDC->LineTo(dx,dy); break;
            case SH_SQUARE:    DrawRectangle(BasicDC,sx,sy,dx,dy); break;
            case SH_RECTANGLE: DrawRectangle(BasicDC,sx,sy,dx,dy); break;
            case SH_CIRCLE:    BasicDC->Ellipse(sx,sy,dx,dy); DrawRectangle(BasicDC,sx,sy,dx,dy);
break;
            case SH_ELLIPSE:   BasicDC->Ellipse(sx,sy,dx,dy); DrawRectangle(BasicDC,sx,sy,dx,dy);
break;
            case SH_TEXT:   break;
            case SH_BITMAP: break;
        };
    }
}

/* ================================================ */
/* MouseMove                                        */
/* ------------------------------------------------ */
/* called when the mouse is moved in the user       */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void ShapeTool::MouseMove(TWindow*, uint, TPoint& point,BondGraphDocument& theDoc)
{
    // If currently placing an object.
```

```
    if (!Typing)
    if (Placing) {
        switch (ShapeType) {
            case SH_LINE:      BasicDC->MoveTo(sx,sy); BasicDC->LineTo(dx,dy); break;
            case SH_SQUARE:    DrawRectangle(BasicDC,sx,sy,dx,dy); break;
            case SH_RECTANGLE: DrawRectangle(BasicDC,sx,sy,dx,dy); break;
            case SH_CIRCLE:    BasicDC->Ellipse(sx,sy,dx,dy); DrawRectangle(BasicDC,sx,sy,dx,dy);
break;
            case SH_ELLIPSE:   BasicDC->Ellipse(sx,sy,dx,dy); DrawRectangle(BasicDC,sx,sy,dx,dy);
break;
            case SH_TEXT:   break;
            case SH_BITMAP: break;
        };

        SnapToGrid(point, &theDoc);
        dx = point.x;
        dy = point.y;

        switch (ShapeType) {
            case SH_LINE:      BasicDC->MoveTo(sx,sy); BasicDC->LineTo(dx,dy); break;
            case SH_SQUARE:    DrawRectangle(BasicDC,sx,sy,dx,dy); break;
            case SH_RECTANGLE: DrawRectangle(BasicDC,sx,sy,dx,dy); break;
            case SH_CIRCLE:    BasicDC->Ellipse(sx,sy,dx,dy); DrawRectangle(BasicDC,sx,sy,dx,dy);
break;
            case SH_ELLIPSE:   BasicDC->Ellipse(sx,sy,dx,dy); DrawRectangle(BasicDC,sx,sy,dx,dy);
break;
            case SH_TEXT:   break;
            case SH_BITMAP: break;
        };
    }
}

/* ================================================ */
/* LeftMouseUp                                      */
/* ------------------------------------------------ */
/* called when the mouse is released in the user    */
/* window.                                          */
/*                                                  */
/* input:  TWindow* thewin                          */
/*         uint                                     */
/*         TPoint&                                  */
/*         BondGraphDocument&                       */
/* ================================================ */
void ShapeTool::LeftMouseUp(TWindow* thewin, uint, TPoint& point, BondGraphDocument& theDoc)
{
    // If currently placing an object stop placing it.
    if (!Typing)
    if (Placing) {
        Placing = FALSE;
        thewin->ReleaseCapture();

        switch (ShapeType) {
            case SH_LINE:      BasicDC->MoveTo(sx,sy); BasicDC->LineTo(dx,dy); break;
            case SH_SQUARE:    DrawRectangle(BasicDC,sx,sy,dx,dy); break;
            case SH_RECTANGLE: DrawRectangle(BasicDC,sx,sy,dx,dy); break;
            case SH_CIRCLE:    BasicDC->Ellipse(sx,sy,dx,dy); DrawRectangle(BasicDC,sx,sy,dx,dy);
break;
            case SH_ELLIPSE:   BasicDC->Ellipse(sx,sy,dx,dy); DrawRectangle(BasicDC,sx,sy,dx,dy);
break;
            case SH_TEXT:   break;
            case SH_BITMAP: break;
        };

        // Get Final Position And Place Object
        SnapToGrid(point, &theDoc);
        dx = point.x;
        dy = point.y;

        // Free Memory Used For Placement
        delete BasicDC;

        // Add Object To Document
        BasicObject* objectPtr;
        objectPtr = new ObjectShape(ShapeType);
        if (ShapeType == SH_LINE) {
          objectPtr->x = sx;
          objectPtr->y = sy;
          objectPtr->w = dx;
          objectPtr->h = dy;
        } else {
          if (sx < dx) {
             if (sy < dy) {
                objectPtr->x = sx;
                objectPtr->y = sy;
                objectPtr->w = dx-sx;
                objectPtr->h = dy-sy;
             } else {
```

```
                objectPtr->x = sx;
                objectPtr->y = dy;
                objectPtr->w = dx-sx;
                objectPtr->h = sy-dy;
            }
          } else {
            if (sy < dy) {
                objectPtr->x = dx;
                objectPtr->y = sy;
                objectPtr->w = sx-dx;
                objectPtr->h = dy-sy;
            } else {
                objectPtr->x = dx;
                objectPtr->y = dy;
                objectPtr->w = sx-dx;
                objectPtr->h = sy-dy;
            }
          }
        }

        theDoc.AddObject(objectPtr);

        // Redraw Area With New Object
//      int minx = sx; if (dx < sx) minx = dx; int w = abs(dx - sx);
//      int maxx = dx; if (dx < sx) maxx = sx;
//      int miny = sy; if (dy < sy) miny = dy; int h = abs(dy - sy);
//      int maxy = dy; if (dy < sy) maxy = sy;
//      thewin->InvalidateRect(TRect(minx,miny,maxx,maxy));
        thewin->Invalidate();
    }
}
```

bttltlst.h

```
#ifndef TOOLLST_H
#define TOOLLST_H

#include "bttltool.h"

class ToolList {
public:
    ToolList();
    ~ToolList();

    void AddTool(BasicTool* t);

    void  CurrentTool(int id);
    BasicTool *CurrentTool();

    // Change to an owl array later.
    BasicTool* tl[30];
    int   num;
    int   activeTool;
};

#endif TOOLLST_H
```

bttltlst.cpp

```
#include "bttltlst.h"

ToolList::ToolList()
{
    num = 1;
    tl[0] = NULL;
    activeTool = 0;
}

ToolList::~ToolList()
{}

void ToolList::AddTool(BasicTool* t)
{
    tl[num] = t;
    num++;
}

void ToolList::CurrentTool(int id)
{
    int i;
    activeTool = 0;
    for (i=0;i<num;i++)
        if (tl[i])
            if (tl[i]->Id == id)
```

```
                activeTool = i;
}

BasicTool* ToolList::CurrentTool()
{
    return (tl[activeTool]);
}
```

msoutlin.h

```
#if !defined(__msoutlin_h)
#define __msoutlin_h

//----------------------------------------------------------
// MSOUTLIN.H generated from MSOUTLIN.VBX by
// VbxGen 1.0 - Borland International
//----------------------------------------------------------

#ifdef __OWL_OWLDEFS_H
  #include <owl\vbxctl.h>
#endif

//----------------------------------------------------------
// Outline (VB1)
//----------------------------------------------------------

// properties
#define Prop_Outline_CtlName 0
#define Prop_Outline_Index 1
#define Prop_Outline_BackColor 2
#define Prop_Outline_Left 3
#define Prop_Outline_Top 4
#define Prop_Outline_Width 5
#define Prop_Outline_Height 6
#define Prop_Outline_Visible 7
#define Prop_Outline_Parent 8
#define Prop_Outline_DragMode 9
#define Prop_Outline_DragIcon 10
#define Prop_Outline_Tag 11
#define Prop_Outline_List 12
#define Prop_Outline_Indent 13
#define Prop_Outline_PicturePlus 14
#define Prop_Outline_PictureMinus 15
#define Prop_Outline_PictureLeaf 16
#define Prop_Outline_PictureOpen 17
#define Prop_Outline_PictureClosed 18
#define Prop_Outline_ListIndex 19
#define Prop_Outline_ListCount 20
#define Prop_Outline_FullPath 21
#define Prop_Outline_PathSeparator 22
#define Prop_Outline_Style 23
#define Prop_Outline_Expand 24
#define Prop_Outline_ItemData 25
#define Prop_Outline_About 26
#define Prop_Outline_TopIndex 27
#define Prop_Outline_PictureType 28
#define Prop_Outline_HasSubItems 29
#define Prop_Outline_IsItemVisible 30
#define Prop_Outline_FontName 31
#define Prop_Outline_FontSize 32
#define Prop_Outline_FontBold 33
#define Prop_Outline_FontItalic 34
#define Prop_Outline_FontStrikethru 35
#define Prop_Outline_FontUnderline 36
#define Prop_Outline_TabStop 37
#define Prop_Outline_TabIndex 38
#define Prop_Outline_Enabled 39
#define Prop_Outline_MousePointer 40
#define Prop_Outline_ForeColor 41
#define Prop_Outline_BorderStyle 42
#define Prop_Outline_Text 43

// events
#define Event_Outline_Click 0
#define Event_Outline_Collapse 1
#define Event_Outline_DblClick 2
#define Event_Outline_DragDrop 3
#define Event_Outline_DragOver 4
#define Event_Outline_GotFocus 5
#define Event_Outline_PictureClick 6
#define Event_Outline_PictureDblClick 7
#define Event_Outline_KeyDown 8
#define Event_Outline_KeyPress 9
#define Event_Outline_KeyUp 10
#define Event_Outline_LostFocus 11
#define Event_Outline_MouseDown 12
```

```
#define Event_Outline_MouseMove 13
#define Event_Outline_MouseUp 14
#define Event_Outline_Expand 15

// default form data
//
// Some VBX controls do not operate correctly when created without
// a form file.  This occurs when a program creates a control
// dynamically rather than as part of a dialog resource.  If this
// control exhibits problems in this mode, try creating it with the
// following form data:
//
// For OWL programs:
//
//   TVbxOutline* c = new TVbxOutline(..., sizeof(OutlineData), OutlineData);
//
// For C/C++ programs:
//
//   HFORMFILE file = VBXCreateFormFile(sizeof(OutlineData), OutlineData);
//   HCTL c = VBXCreate(..., file);
//   VBXDeleteFormFile(file);
//
// Note that the VBXGEN_DATA or Outline_DATA symbol must be
// defined in order to use the default form data.
//
extern BYTE OutlineData[1225L];
#if defined(VBXGEN_DATA) || defined(Outline_DATA)
  BYTE OutlineData[1225L]={
    0x00,0x07,0x4f,0x75,0x74,0x6c,0x69,0x6e,
    0x65,0x01,0x00,0x00,0x02,0xff,0xff,0xff,
    0x00,0x03,0x07,0xff,0xff,0x09,0x00,0x0b,
    0x00,0x0e,0xde,0x00,0x42,0x4d,0xde,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x76,0x00,
    0x00,0x00,0x28,0x00,0x00,0x00,0x0d,0x00,
     0x00,0x00,0x0d,0x00,0x00,0x00,0x01,0x00,
    0x04,0x00,0x00,0x00,0x00,0x00,0x68,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x10,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x80,0x00,0x00,0x80,0x00,0x00,0x00,0x80,
    0x80,0x00,0x80,0x00,0x00,0x00,0x80,0x00,
    0x80,0x00,0x80,0x80,0x00,0x00,0xc0,0xc0,
    0xc0,0x00,0x80,0x80,0x80,0x00,0x00,0x00,
    0xff,0x00,0x00,0xff,0x00,0x00,0x00,0xff,
    0xff,0x00,0xff,0x00,0x00,0x00,0xff,0x00,
     0xff,0x00,0xff,0xff,0x00,0x00,0xff,0xff,
    0xff,0x00,0xff,0xff,0xff,0xff,0xff,0xff,
    0x0f,0x91,0xff,0xff,0xff,0xff,0xff,0xff,
     0x0f,0x00,0xff,0xff,0xf0,0x00,0xff,0xff,
    0x0f,0x00,0xff,0xff,0xf0,0xf0,0xff,0xff,
    0x0f,0x00,0xff,0xff,0xf0,0xf0,0xff,0xff,
    0x0f,0x00,0xff,0x00,0x00,0xf0,0x00,0x0f,
    0x0f,0x23,0xff,0x0f,0xff,0xff,0xff,0x0f,
    0x0f,0x00,0xff,0x00,0x00,0xf0,0x00,0x0f,
    0x0f,0x00,0xff,0xff,0xf0,0xf0,0xff,0xff,
    0x0f,0x29,0xff,0xff,0xf0,0xf0,0xff,0xff,
    0x0f,0x97,0xff,0xff,0xf0,0x00,0xff,0xff,
    0x0f,0x00,0xff,0xff,0xff,0xff,0xff,0xff,
    0x0f,0x00,0xff,0xff,0xff,0xff,0xff,0xff,
    0x0f,0x00,0x0f,0xde,0x00,0x42,0x4d,0xde,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x76,
    0x00,0x00,0x00,0x28,0x00,0x00,0x00,0x0d,
    0x00,0x00,0x00,0x0d,0x00,0x00,0x00,0x01,
    0x00,0x04,0x00,0x00,0x00,0x00,0x00,0x68,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x80,0x00,0x00,0x80,0x00,0x00,0x00,
    0x80,0x80,0x00,0x80,0x00,0x00,0x00,0x80,
     0x00,0x80,0x00,0x80,0x80,0x00,0x00,0xc0,
    0xc0,0xc0,0x00,0x80,0x80,0x80,0x00,0x00,
    0x00,0xff,0x00,0x00,0xff,0x00,0x00,0x00,
     0xff,0xff,0x00,0xff,0x00,0x00,0x00,0xff,
     0x00,0xff,0x00,0xff,0xff,0x00,0x00,0xff,
     0xff,0xff,0x00,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x91,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x00,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x00,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x00,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x00,0xff,0x00,0x00,0x00,0x00,
     0x0f,0x0f,0x23,0xff,0x0f,0xff,0xff,0xff,
     0x0f,0x0f,0x00,0xff,0x00,0x00,0x00,0x00,
     0x0f,0x0f,0x00,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x29,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x97,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x00,0xff,0xff,0xff,0xff,0xff,
     0xff,0x0f,0x00,0xff,0xff,0xff,0xff,0xff,
```

```
0xff,0x0f,0x00,0x10,0xde,0x00,0x42,0x4d,
0xde,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x76,0x00,0x00,0x00,0x28,0x00,0x00,0x00,
0x0d,0x00,0x00,0x00,0x0d,0x00,0x00,0x00,
0x01,0x00,0x04,0x00,0x00,0x00,0x00,0x00,
0x68,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x10,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x80,0x00,0x00,0x80,0x00,0x00,
0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x00,
0x80,0x00,0x80,0x00,0x80,0x80,0x00,0x00,
0xc0,0xc0,0xc0,0x00,0x80,0x80,0x80,0x00,
0x00,0x00,0xff,0x00,0x00,0xff,0x00,0x00,
0x00,0xff,0xff,0x00,0xff,0x00,0x00,0x00,
0xff,0x00,0xff,0x00,0xff,0xff,0x00,0x00,
0xff,0xff,0xff,0x00,0xff,0xff,0xff,0xff,
0xff,0xff,0x0f,0x91,0xf0,0x00,0x00,0x00,
0x00,0x0f,0x0f,0x00,0xf0,0xff,0xff,0xff,
0xff,0x0f,0x0f,0x00,0xf0,0xff,0xff,0xff,
0xff,0x0f,0x0f,0x00,0xf0,0xff,0x44,0x44,
0xff,0x0f,0x0f,0x00,0xf0,0xff,0xff,0xff,
0xff,0x0f,0x0f,0x23,0xf0,0xff,0x44,0x44,
0xff,0x0f,0x0f,0x00,0xf0,0xff,0xff,0xff,
0xff,0x0f,0x0f,0x00,0xf0,0xff,0x44,0x4f,
0x00,0x0f,0x0f,0x29,0xf0,0xff,0xff,0xff,
0x07,0x0f,0x0f,0x97,0xf0,0xff,0xff,0xff,
0x00,0xff,0x0f,0x00,0xf0,0x00,0x00,0x00,
0x0f,0xff,0x0f,0x00,0xff,0xff,0xff,0xff,
0xff,0xff,0x0f,0x00,0x11,0xde,0x00,0x42,
0x4d,0xde,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x76,0x00,0x00,0x00,0x28,0x00,0x00,
0x00,0x0d,0x00,0x00,0x00,0x0d,0x00,0x00,
0x00,0x01,0x00,0x04,0x00,0x00,0x00,0x00,
0x00,0x68,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x10,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x80,0x00,0x00,0x80,0x00,
0x00,0x00,0x80,0x80,0x00,0x80,0x00,0x00,
0x00,0x80,0x00,0x80,0x00,0x80,0x80,0x00,
0x00,0xc0,0xc0,0xc0,0x00,0x80,0x80,0x80,
0x00,0x00,0x00,0xff,0x00,0x00,0xff,0x00,
0x00,0x00,0xff,0xff,0x00,0xff,0x00,0x00,
0x00,0xff,0x00,0xff,0x00,0xff,0xff,0x00,
0x00,0xff,0xff,0xff,0x00,0xff,0xff,0xff,
0xff,0xff,0xff,0x0f,0xff,0xf0,0x00,0x00,
0x00,0x00,0xff,0x0f,0xff,0x00,0x77,0x77,
0x77,0x77,0x0f,0x0f,0xff,0x07,0x0f,0xbf,
0xbf,0xbf,0x0f,0x0f,0x00,0x07,0x0b,0xfb,
0xfb,0xfb,0xf0,0x0f,0x00,0x07,0x70,0xbf,
0xbf,0xbf,0xb0,0x0f,0x00,0x07,0x70,0xfb,
0xfb,0xfb,0xfb,0x00,0x00,0x07,0x77,0x00,
0x00,0x00,0x00,0x00,0x00,0x07,0x77,0x77,
0x77,0x77,0x0f,0x0f,0x00,0x07,0x77,0x70,
0x00,0x00,0xff,0x0f,0x00,0xf0,0x00,0x0f,
0xff,0xff,0xff,0x0f,0x00,0xff,0xff,0xff,
0xff,0xff,0xff,0x0f,0x00,0xff,0xff,0xff,
0xff,0xff,0xff,0x0f,0x00,0x12,0xde,0x00,
0x42,0x4d,0xde,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x76,0x00,0x00,0x00,0x28,0x00,
0x00,0x00,0x0d,0x00,0x00,0x00,0x0d,0x00,
0x00,0x00,0x01,0x00,0x04,0x00,0x00,0x00,
0x00,0x00,0x68,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x10,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x80,0x00,0x00,0x80,
0x00,0x00,0x00,0x80,0x80,0x00,0x80,0x00,
0x00,0x00,0x80,0x00,0x80,0x00,0x80,0x80,
0x00,0x00,0xc0,0xc0,0xc0,0x00,0x80,0x80,
0x80,0x00,0x00,0x00,0xff,0x00,0x00,0xff,
0x00,0x00,0x00,0xff,0xff,0x00,0xff,0x00,
0x00,0x00,0xff,0x00,0xff,0x00,0xff,0xff,
0x00,0x00,0xff,0xff,0xff,0x00,0xff,0xff,
0xff,0xff,0xff,0xff,0x0f,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0x0f,0xff,0xf0,0x00,
0x00,0x00,0x00,0x00,0x0f,0xff,0x07,0x77,
0x77,0x77,0x77,0x70,0x0f,0x00,0x0b,0xfb,
0xfb,0xfb,0xfb,0xf0,0x0f,0x00,0x0f,0xbf,
0xbf,0xbf,0xbf,0xb0,0x0f,0x00,0x0b,0xfb,
0xfb,0xfb,0xfb,0xf0,0x0f,0x00,0x0f,0xbf,
0xbf,0xbf,0xbf,0xb0,0x0f,0x00,0x0b,0xfb,
0xfb,0xfb,0xfb,0xf0,0x0f,0x00,0x00,0x00,
0x00,0x00,0x00,0x0f,0x0f,0x00,0xf0,0xbf,
0xbf,0x0f,0xff,0xff,0x0f,0x00,0xff,0x00,
0x00,0xff,0xff,0xff,0x0f,0x00,0xff,0xff,
0xff,0xff,0xff,0xff,0x0f,0x00,0x16,0x01,
0x5c,0x17,0x03,0x1f,0x40,0x01,0x07,0x00,
0x00,0x00,0x00,0x00,0xbc,0x02,0x00,0x00,
```

```
    0x00,0x00,0x01,0x02,0x02,0x22,0x53,0x79,
    0x73,0x74,0x65,0x6d,0x00,0x37,0x50,0x00,
    0xbe,0x31,0x50,0x00,0xbe,0x31,0xc7,0x37,
    0x09,0x27,0x04,0x1a,0xaf,0x37,0x50,0x00,
    0xbe,0x31,0x10,0x18,0x20,0x39,0x25,0x00,
    0x00,0x26,0x00,0x00,0x27,0xff,0xff,0x28,
    0x00,0x29,0x00,0x00,0x00,0x00,0x2a,0x01,
    0xff,
};
#endif

#ifdef __OWL_VBXCTL_H

// OWL class
class TVbxOutline : public TVbxControl {
  public:

    // constructors
    TVbxOutline(TWindow* parent, int id, LPCSTR title,
        int x, int y, int w, int h,
        long initLen=0, void far* initData=0, TModule* module=0) :
      TVbxControl(parent, id, "MSOUTLIN.VBX", "Outline",
          title, x, y, w, h, initLen, initData, module) {}

    TVbxOutline(TWindow* parent, int resourceId, TModule* module=0) :
        TVbxControl(parent, resourceId, module) {}

#if 0
    // event handlers
    void EvClick(VBXEVENT FAR*)
    {
      // no arguments
    }

    void EvCollapse(VBXEVENT FAR*)
    {
      // ListIndex As Integer
    }

    void EvDblClick(VBXEVENT FAR*)
    {
      // no arguments
    }

    void EvDragDrop(VBXEVENT FAR*)
    {
      // Source As Control,X As Integer,Y As Integer
    }

    void EvDragOver(VBXEVENT FAR*)
    {
      // Source As Control,X As Integer,Y As Integer,State As Integer
    }

    void EvGotFocus(VBXEVENT FAR*)
    {
      // no arguments
    }

    void EvPictureClick(VBXEVENT FAR*)
    {
      // ListIndex As Integer
    }

    void EvPictureDblClick(VBXEVENT FAR*)
    {
      // ListIndex As Integer
    }

    void EvKeyDown(VBXEVENT FAR*)
    {
      // KeyCode As Integer,Shift As Integer
    }

    void EvKeyPress(VBXEVENT FAR*)
    {
      // KeyAscii As Integer
    }

    void EvKeyUp(VBXEVENT FAR*)
    {
      // KeyCode As Integer,Shift As Integer
    }

    void EvLostFocus(VBXEVENT FAR*)
    {
      // no arguments
```

```
    }

    void EvMouseDown(VBXEVENT FAR*)
    {
      // Button As Integer,Shift As Integer,X As Integer,Y As Integer
    }

    void EvMouseMove(VBXEVENT FAR*)
    {
      // Button As Integer,Shift As Integer,X As Integer,Y As Integer
    }

    void EvMouseUp(VBXEVENT FAR*)
    {
      // Button As Integer,Shift As Integer,X As Integer,Y As Integer
    }

    void EvExpand(VBXEVENT FAR*)
    {
      // ListIndex As Integer
    }

#endif

    // enumerations
    enum { // values for property DragMode
      DragMode_0_Manual,
      DragMode_1_Automatic,
    };

    enum { // values for property Style
      Style_0_Text_Only,
      Style_1_Picture_and_Text,
      Style_2_PlusMinus_and_Text,
      Style_3_PlusMinus_Picture_and_Text,
      Style_4_Treelines_and_Text,
      Style_5_Treelines_Picture_and_Text,
    };

    enum { // values for property MousePointer
      MousePointer_0_Default,
      MousePointer_1_Arrow,
      MousePointer_2_Cross,
      MousePointer_3_IBeam,
      MousePointer_4_Icon,
      MousePointer_5_Size,
      MousePointer_6_Size_NE_SW,
      MousePointer_7_Size_NS,
      MousePointer_8_Size_NW_SE,
      MousePointer_9_Size_WE,
      MousePointer_10_Up_Arrow,
      MousePointer_11_Hourglass,
      MousePointer_12_No_Drop,
    };

    enum { // values for property BorderStyle
      BorderStyle_0_None,
      BorderStyle_1_Fixed_Single,
    };


    // properties
    BOOL GetPropCtlName(string& v) {return GetProp(0,v);}
    BOOL SetPropCtlName(string& v) {return SetProp(0,v);}

    BOOL GetPropIndex(int& v) {return GetProp(1,v);}
    BOOL SetPropIndex(int v) {return SetProp(1,v);}

    BOOL GetPropBackColor(COLORREF& v) {return GetProp(2,v);}
    BOOL SetPropBackColor(COLORREF v) {return SetProp(2,v);}

    BOOL GetPropLeft(long& v) {return GetProp(3,v);}
    BOOL SetPropLeft(long v) {return SetProp(3,v);}

    BOOL GetPropTop(long& v) {return GetProp(4,v);}
    BOOL SetPropTop(long v) {return SetProp(4,v);}

    BOOL GetPropWidth(long& v) {return GetProp(5,v);}
    BOOL SetPropWidth(long v) {return SetProp(5,v);}

    BOOL GetPropHeight(long& v) {return GetProp(6,v);}
    BOOL SetPropHeight(long v) {return SetProp(6,v);}

    BOOL GetPropVisible(BOOL& v) {return GetProp(7,v);}
    BOOL SetPropVisible(BOOL v) {return SetProp(7,v);}

    BOOL GetPropParent(int& v) {return GetProp(8,v);}
```

```
BOOL SetPropParent(int v) {return SetProp(8,v);}

BOOL GetPropDragMode(ENUM& v) {return GetProp(9,v);}
BOOL SetPropDragMode(ENUM v) {return SetProp(9,v);}

BOOL GetPropDragIcon(HPIC& v) {return GetProp(10,v);}
BOOL SetPropDragIcon(HPIC v) {return SetProp(10,v);}

BOOL GetPropTag(string& v) {return GetProp(11,v);}
BOOL SetPropTag(string& v) {return SetProp(11,v);}

BOOL GetPropList(string& v, int i) {return GetProp(12,v,i);}
BOOL SetPropList(string& v, int i) {return SetProp(12,v,i);}

BOOL GetPropIndent(int& v, int i) {return GetProp(13,v,i);}
BOOL SetPropIndent(int v, int i) {return SetProp(13,v,i);}

BOOL GetPropPicturePlus(HPIC& v) {return GetProp(14,v);}
BOOL SetPropPicturePlus(HPIC v) {return SetProp(14,v);}

BOOL GetPropPictureMinus(HPIC& v) {return GetProp(15,v);}
BOOL SetPropPictureMinus(HPIC v) {return SetProp(15,v);}

BOOL GetPropPictureLeaf(HPIC& v) {return GetProp(16,v);}
BOOL SetPropPictureLeaf(HPIC v) {return SetProp(16,v);}

BOOL GetPropPictureOpen(HPIC& v) {return GetProp(17,v);}
BOOL SetPropPictureOpen(HPIC v) {return SetProp(17,v);}

BOOL GetPropPictureClosed(HPIC& v) {return GetProp(18,v);}
BOOL SetPropPictureClosed(HPIC v) {return SetProp(18,v);}

BOOL GetPropListIndex(int& v) {return GetProp(19,v);}
BOOL SetPropListIndex(int v) {return SetProp(19,v);}

BOOL GetPropListCount(int& v) {return GetProp(20,v);}
BOOL SetPropListCount(int v) {return SetProp(20,v);}

BOOL GetPropFullPath(string& v, int i) {return GetProp(21,v,i);}
BOOL SetPropFullPath(string& v, int i) {return SetProp(21,v,i);}

BOOL GetPropPathSeparator(string& v) {return GetProp(22,v);}
BOOL SetPropPathSeparator(string& v) {return SetProp(22,v);}

BOOL GetPropStyle(ENUM& v) {return GetProp(23,v);}
BOOL SetPropStyle(ENUM v) {return SetProp(23,v);}

BOOL GetPropExpand(int& v, int i) {return GetProp(24,v,i);}
BOOL SetPropExpand(int v, int i) {return SetProp(24,v,i);}

BOOL GetPropItemData(long& v, int i) {return GetProp(25,v,i);}
BOOL SetPropItemData(long v, int i) {return SetProp(25,v,i);}

BOOL GetPropAbout(string& v) {return GetProp(26,v);}
BOOL SetPropAbout(string& v) {return SetProp(26,v);}

BOOL GetPropTopIndex(int& v) {return GetProp(27,v);}
BOOL SetPropTopIndex(int v) {return SetProp(27,v);}

BOOL GetPropPictureType(int& v, int i) {return GetProp(28,v,i);}
BOOL SetPropPictureType(int v, int i) {return SetProp(28,v,i);}

BOOL GetPropHasSubItems(BOOL& v, int i) {return GetProp(29,v,i);}
BOOL SetPropHasSubItems(BOOL v, int i) {return SetProp(29,v,i);}

BOOL GetPropIsItemVisible(BOOL& v, int i) {return GetProp(30,v,i);}
BOOL SetPropIsItemVisible(BOOL v, int i) {return SetProp(30,v,i);}

BOOL GetPropFontName(string& v) {return GetProp(31,v);}
BOOL SetPropFontName(string& v) {return SetProp(31,v);}

BOOL GetPropFontSize(float& v) {return GetProp(32,v);}
BOOL SetPropFontSize(float v) {return SetProp(32,v);}

BOOL GetPropFontBold(BOOL& v) {return GetProp(33,v);}
BOOL SetPropFontBold(BOOL v) {return SetProp(33,v);}

BOOL GetPropFontItalic(BOOL& v) {return GetProp(34,v);}
BOOL SetPropFontItalic(BOOL v) {return SetProp(34,v);}

BOOL GetPropFontStrikethru(BOOL& v) {return GetProp(35,v);}
BOOL SetPropFontStrikethru(BOOL v) {return SetProp(35,v);}

BOOL GetPropFontUnderline(BOOL& v) {return GetProp(36,v);}
BOOL SetPropFontUnderline(BOOL v) {return SetProp(36,v);}

BOOL GetPropTabStop(BOOL& v) {return GetProp(37,v);}
```

```
    BOOL SetPropTabStop(BOOL v) {return SetProp(37,v);}

    BOOL GetPropTabIndex(int& v) {return GetProp(38,v);}
    BOOL SetPropTabIndex(int v) {return SetProp(38,v);}

    BOOL GetPropEnabled(BOOL& v) {return GetProp(39,v);}
    BOOL SetPropEnabled(BOOL v) {return SetProp(39,v);}

    BOOL GetPropMousePointer(ENUM& v) {return GetProp(40,v);}
    BOOL SetPropMousePointer(ENUM v) {return SetProp(40,v);}

    BOOL GetPropForeColor(COLORREF& v) {return GetProp(41,v);}
    BOOL SetPropForeColor(COLORREF v) {return SetProp(41,v);}

    BOOL GetPropBorderStyle(ENUM& v) {return GetProp(42,v);}
    BOOL SetPropBorderStyle(ENUM v) {return SetProp(42,v);}

    BOOL GetPropText(string& v) {return GetProp(43,v);}
    BOOL SetPropText(string& v) {return SetProp(43,v);}

#if 0
  DECLARE_RESPONSE_TABLE(TVbxOutline);
#endif
};

#if 0
// OWL response table
DEFINE_RESPONSE_TABLE1(TVbxOutline, TVbxControl)
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_Click, EvClick ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_Collapse, EvCollapse ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_DblClick, EvDblClick ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_DragDrop, EvDragDrop ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_DragOver, EvDragOver ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_GotFocus, EvGotFocus ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_PictureClick, EvPictureClick ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_PictureDblClick, EvPictureDblClick ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_KeyDown, EvKeyDown ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_KeyPress, EvKeyPress ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_KeyUp, EvKeyUp ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_LostFocus, EvLostFocus ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_MouseDown, EvMouseDown ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_MouseMove, EvMouseMove ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_MouseUp, EvMouseUp ),
  EV_VBXEVENTINDEX( IDC_Outline, Event_Outline_Expand, EvExpand ),
END_RESPONSE_TABLE;
#endif

#endif

#endif
```

koutlin.h

```
#if !defined(koutlin_h)
#define koutlin_h

#include <owl\owlpch.h>
#include "msoutlin.h"

typedef struct {
  string text;
  char   indent;
} OutlineItem;

// An array is used to hold the objects.
#include <classlib\arrays.h>
typedef TArray<OutlineItem *> TItemArray;
typedef TArrayAsVectorIterator<OutlineItem *> TItemArrayIterator;

class KOutline : public TVbxOutline {
public:
    TItemArray* items;
    int count;

public:
    KOutline(TWindow* parent, int id, LPCSTR title, int x, int y, int w, int h,
             long initLen=0, void far* initData=0, TModule* module=0);

    KOutline(TWindow* parent, int resourceId, TModule* module=0);

    void AddItem(string& text, int indent);
    void GetFullPathName(string& text, int index);
    void GetItem(string& text, int index);
};
```

```
#endif
```

koutlin.cpp

```
#include "koutlin.h"

KOutline::KOutline(TWindow* parent, int id, LPCSTR title, int x, int y, int w, int h,
                   long initLen, void far* initData, TModule* module)
    : TVbxOutline(parent,id,title,x,y,w,h,initLen,initData,module)
{
  items = new TItemArray(5,1,5);
  count = 1;
}

KOutline::KOutline(TWindow* parent, int resourceId, TModule* module)
    : TVbxOutline(parent,resourceId,module)
{
  items = new TItemArray(5,1,5);
  count = 1;
}

void KOutline::GetItem(string& text, int index)
{
  text = (*items)[index]->text;
}

void KOutline::GetFullPathName(string& text, int index)
{
  int done = FALSE;
  text = (*items)[index]->text;
  int oldIndent = (*items)[index]->indent;
  for (int i=index-1;i>0 && ~done;i--) {
      int newIndent = (*items)[i]->indent;
      if (newIndent == 1)
         done = TRUE;

      if (newIndent != oldIndent) {
         oldIndent = newIndent;
         text = (*items)[i]->text + '\\' + text;
      }
  }
}

void KOutline::AddItem(string& text, int indent)
{
  OutlineItem* i = new OutlineItem;
  i->indent = indent;
  i->text = text;

  items->Add(i);
  SetPropList(text,count);
  SetPropIndent(indent,count);
  count++;
}
```

bttlbtn.h

```
#if !defined(__bttlbtn_h)              // Sentry, use file only if it's not already included.
#define __bttlbtn_h

/*  Project bottle
     Massachusetts Institute of Technology
     Copyright © 1995. All Rights Reserved.

     SUBSYSTEM:    bottle.apx Application
     FILE:         bttlbtn.h
     AUTHOR:       Richard K. Branton


     OVERVIEW
     ========
     Class definition for BottleBitmapBtn (TButton).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "bttlapp.rh"            // Definition of all resources.

//{{TButton = BottleBitmapBtn}}
class BottleBitmapBtn : public TButton {
protected:
    TDib     *dib;
```

```
    TPalette *palette;

public:
    BottleBitmapBtn (TWindow* parent, int id, int rID, TModule* module = 0);
    virtual ~BottleBitmapBtn ();

//{{BottleBitmapBtnVIRTUAL_BEGIN}}
public:
    virtual void ODADrawEntire (DRAWITEMSTRUCT far& drawInfo);
    virtual void ODAFocus (DRAWITEMSTRUCT far& drawInfo);
    virtual void ODASelect (DRAWITEMSTRUCT far& drawInfo);
//{{BottleBitmapBtnVIRTUAL_END}}
};    //{{BottleBitmapBtn}}

#endif                                          // __bttlbtn_h sentry.
```

bttlbtn.cpp

```
/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.apx Application
    FILE:         bttlbtn.cpp
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Source file for implementation of BottleBitmapBtn (TButton).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "bttlbtn.h"


//{{BottleBitmapBtn Implementation}}


BottleBitmapBtn::BottleBitmapBtn (TWindow* parent, int id, int rID, TModule* module):
    TButton(parent, id, module)
{
    // Change the window's background color
    // SetBkgndColor(RGB(0xff, 0xff, 0xff));

    // INSERT>> Your constructor code here.
    dib = new TDib (GetApplication()->GetInstance(),rID);
    palette = new TPalette (*dib);
}


BottleBitmapBtn::~BottleBitmapBtn ()
{
    Destroy();

    // INSERT>> Your destructor code here.
}


void BottleBitmapBtn::ODADrawEntire (DRAWITEMSTRUCT far& drawInfo)
{
    TButton::ODADrawEntire(drawInfo);

    // INSERT>> Your code here.
    TDC dc(drawInfo.hDC);
    TBitmap ddb(*dib, palette);
    TMemoryDC memDC(dc);

    dc.SelectObject(*palette,false);
    memDC.SelectObject(*palette,false);
    memDC.SelectObject(ddb);

    dc.BitBlt(0, 0, drawInfo.rcItem.right, drawInfo.rcItem.bottom, memDC, 0, 0, SRCCOPY);
}


void BottleBitmapBtn::ODAFocus (DRAWITEMSTRUCT far& drawInfo)
{
    TButton::ODAFocus(drawInfo);

    // INSERT>> Your code here.
    ODADrawEntire(drawInfo);
}
```

```
void BottleBitmapBtn::ODASelect (DRAWITEMSTRUCT far& drawInfo)
{
     TButton::ODASelect(drawInfo);

     // INSERT>> Your code here.
    ODADrawEntire(drawInfo);
}
```

bttlpal.h

```
/* ================================================ */
/* BottlePalette Class                              */
/* ------------------------------------------------ */
/*                                                  */
/* ------------------------------------------------ */
/*   Code by Richard K. Branton - MIT co 94/95 EECS */
/*                Copyright (c) 1995                */
/*       Massachusetts Institute of Technology      */
/*         Mechanical Engineering Department        */
/* ================================================ */

#if !defined(__bttlpal_h)              // Sentry, use file only if it's not already included.
#define __bttlpal_h

/* ================================================ */
/* Include Files                                    */
/* ------------------------------------------------ */
#include <owl\owlpch.h>
#include <owl\window.h>

class SizerButton : public TButton {
public:
    TFloatingFrame*   thewin;
    TScreenDC* dc;
    int        sx, sy, pw, ph;
    int        oldHeight, maxHeight;

public:
    SizerButton(TFloatingFrame* parent, int x, int y, int w, int h, int maxh);

    void EvLButtonDown (uint modKeys, TPoint& point);
    void EvMouseMove (uint modKeys, TPoint& point);
    void EvLButtonUp (uint modKeys, TPoint& point);
//  bool EvSetCursor (HWND hWndCursor, uint hitTest, uint mouseMsg);

DECLARE_RESPONSE_TABLE(SizerButton);
};

/* ================================================ */
/* BottlePalette Class Definition                   */
/* ------------------------------------------------ */
class BottlePalette : public TFloatingFrame {
public:
    TToolBox*     toolbox;
    TDialog*      ResDialog;
    SizerButton*  ResizeButton;
    int           x,y,*visible,visnotused;

public:
    // Constructors and Destructors
    BottlePalette(TWindow*, char* title="UNTITLED", int* visible_ptr=0, int x=-1, int y=-1);
  ~BottlePalette();

    // Palette Maintanence
    void AddButton(int resID, int butID, TButtonGadget::TState pos = TButtonGadget::Up);
    void AddSeparator();
    void FixSize();

    virtual void SizeDialog();
    virtual void InitDialog();

    void SetupPalette(int resID = -1, int create = TRUE);

    // Event Handlers
    bool ShowWindow();
    bool HideWindow();

    bool CanClose();
};

#endif                                      // __bttlpal_h sentry.
```

bttlpal.cpp

```
#include "bttlpal.h"
#include "bttlapp.rh"
#include "palettes.h"

#define PAL_RESIZE 5800

SizerButton::SizerButton(TFloatingFrame* parent, int x, int y, int w, int h, int maxh)
    : TButton(parent, PAL_RESIZE, "", x, y, w, h)
{
  thewin = parent;
  dc = NULL;
  maxHeight = maxh;
}

void DrawRectangle(TDC*, int, int, int, int);

DEFINE_RESPONSE_TABLE1(SizerButton, TButton)
     EV_WM_LBUTTONDOWN,
     EV_WM_MOUSEMOVE,
     EV_WM_LBUTTONUP,
END_RESPONSE_TABLE;

void SizerButton::EvLButtonDown(uint, TPoint& point)
{
  if (dc == NULL) {
     dc = new TScreenDC();
     SetCapture();

     TRect parentRect = thewin->GetWindowRect();

     sx = parentRect.TopLeft().x;
     sy = parentRect.TopLeft().y;
     pw  = parentRect.Width();
     oldHeight = parentRect.Height();
     ph = oldHeight - point.y;

     dc->SetROP2(R2_NOT);
     DrawRectangle(dc,sx,sy,sx+pw,sy+oldHeight);
  }
}

void SizerButton::EvMouseMove(uint, TPoint& point)
{
  if (dc != NULL) {
     int h = point.y + ph;
     if (h > maxHeight) h = maxHeight;
     if (h < 22) h = 22;
     if (h != oldHeight) {
        DrawRectangle(dc,sx,sy,sx+pw,sy+oldHeight);
        DrawRectangle(dc,sx,sy,sx+pw,sy+h);
        oldHeight = h;
     }
  }
}

void SizerButton::EvLButtonUp(uint, TPoint& point)
{
  if (dc != NULL) {
     ReleaseCapture();
     DrawRectangle(dc,sx,sy,sx+pw,sy+oldHeight);
     delete dc;
     dc = NULL;
     thewin->MoveWindow(TRect(TPoint(sx,sy),TSize(pw,oldHeight)),true);
     ((BottlePalette *) thewin)->FixSize();
  }
}

BottlePalette::BottlePalette(TWindow* thewin, char* title, int* visible_ptr, int tx, int ty)
    : TFloatingFrame(thewin,title,0, true, TFloatingFrame::DefaultCaptionHeight, true)
{
    // Set Window Attributes And Create Palette
    Attr.Style = WS_POPUP | WS_CAPTION | WS_BORDER | WS_VSCROLL;
    CloseBox = FALSE;
    if (tx<0 || ty<0) {
      TRect parentRect = thewin->GetWindowRect();
      tx = parentRect.TopLeft().x;
      ty = parentRect.TopLeft().y;
    }
    x = tx; y = ty;

    if (visible_ptr)
      visible = visible_ptr;
    else
```

```
      visible = &visnotused;
    *visible = FALSE;
}

void BottlePalette::SetupPalette(int resID, int create)
{
  if (create)
     Create();
  if (resID < 0) {  // Create Palette As Toolbox
     // Create Toolbox
     toolbox = new TToolBox(this,2,AS_MANY_AS_NEEDED,TToolBox::Vertical);
     toolbox->Create();

     // Resize Palette Window
     MoveWindow(TRect(TPoint(x,y),TSize((2+20+20+2),50)),true);

     ResDialog = NULL;
     ResizeButton = NULL;
  } else {          // Create Palette From Dialog
     // Create Dialog From Resource
     if (create==5439)
        ResDialog = new KDialog(this, resID);
     else
       ResDialog = new TDialog(this, resID);
     ResDialog->Create();

     // Add Scroller to Control Scrollbar
     int dH = ((TWindow*) ResDialog)->Attr.H;
     int oH = dH;
     int dW = ((TWindow*) ResDialog)->Attr.W;
     ResDialog->Scroller = new TScroller(ResDialog, 1, 5, 1, (oH+5+11)/5);
     ResDialog->Scroller->AutoMode = FALSE;

     // Create Resize Button
     if (dH+11+5+10 > 150) dH = 150;
     ResDialog->MoveWindow(TRect(TPoint(0,0),TSize(dW,dH)));

     ResizeButton = new SizerButton(this, 2, 2+dH, dW-1, 10, oH+11+5+10);
     ResizeButton->Create();

     // Resize Palette Window
     MoveWindow(TRect(TPoint(x,y),TSize(dW+5+1,dH+11+5+10)),true);
     toolbox = NULL;
  }
}

BottlePalette::~BottlePalette()
{
  *visible = FALSE;
  delete ResizeButton;
  delete ResDialog;
  delete toolbox;
}

bool BottlePalette::ShowWindow()
{
  *visible = TRUE;
  return TFloatingFrame::ShowWindow(SW_SHOWNORMAL);
}

bool BottlePalette::HideWindow()
{
  *visible = FALSE;
  return TFloatingFrame::ShowWindow(SW_HIDE);
}

bool BottlePalette::CanClose()
{
  bool result = TFloatingFrame::CanClose();
  if (result && visible)
     *visible = FALSE;
  return result;
}

void BottlePalette::AddButton(int resID, int butID, TButtonGadget::TState pos)
{
  if (toolbox) {
     toolbox->Insert(*new TButtonGadget(resID,butID,TButtonGadget::Exclusive,true,pos));
     toolbox->LayoutSession();
     FixSize();
  }
}

void BottlePalette::AddSeparator()
{
  if (toolbox) {
     toolbox->Insert(*new TButtonGadget(CM_EMPTY,0,TButtonGadget::Exclusive,false));
```

```
    toolbox->LayoutSession();
    FixSize();
  }
}

void BottlePalette::FixSize()
{
  if (ResDialog == NULL) {
    TRect toolboxRect = toolbox->GetWindowRect();
    TRect clientRect = GetClientRect();
    TRect oldRect = GetWindowRect();

    TRect newRect(TPoint(oldRect.TopLeft()),
                  TSize(toolboxRect.Width() + (oldRect.Width() - clientRect.Width()),
                        toolboxRect.Height() + (oldRect.Height() - clientRect.Height())));
    MoveWindow(newRect,true);
  } else {
    TRect clientRect = GetClientRect();
    TRect dialogRect = ResDialog->GetWindowRect();
    TRect buttonRect = ResizeButton->GetWindowRect();
    TRect newDRect(TPoint(0,0),TSize(dialogRect.Width(), clientRect.Height()-10));
    TRect newBRect(TPoint(2,newDRect.Height()+2),TSize(buttonRect.Width(),buttonRect.Height()));
    ResDialog->MoveWindow(newDRect,TRUE);
    ResizeButton->MoveWindow(newBRect,TRUE);
    SizeDialog();
  }
}

void BottlePalette::SizeDialog()
{}

void BottlePalette::InitDialog()
{}
```

palettes.h

```
#if !defined(palettes_h)
#define palettes_h

#include "koutlin.h"
#include "bttlpal.h"

void CreateShapePalette(TWindow*, BottlePalette**, int *);
void CreateBasicPalette(TWindow*, BottlePalette**, int *);
void CreateAttributePalette(TWindow*, BottlePalette**, int *);
void CreateCompoundPalette(TWindow*, BottlePalette**, int *);

#include "msoutlin.h"

class ColorButtonGadget : public TButtonGadget {
public:
  TColor color;
  ColorButtonGadget(TColor c, TResId bmpResId, int id, TType type = TButtonGadget::Command,
                    bool enabled = FALSE, TState state = TButtonGadget::Up, bool repeat =
false);
  void Paint(TDC& dc);
  void Redraw();
};

// Compound Object Palette
class CompoundPalette : public BottlePalette {
public:
    TControl*     CompoundToolsLoc;
    //TVbxOutline*  CompoundOutline;
    KOutline* CompoundOutline;
    TRadioButton* CompoundLink;


public:
    CompoundPalette(TWindow*, int resId, char* title="UNTITLED", int* visible_ptr=0, int x=-1, int y=-
1);
    void UpdateListBox (TWindow* thewin, /*TVbx*/KOutline* outline, char *filespec, int level, int&
which);
};


// Attributes Object Palette
#define ATTRIB_GRID       0
#define ATTRIB_ALIGN      1
#define ATTRIB_DISPLAY    2
#define ATTRIB_FONT       3
#define ATTRIB_FORE       4
#define ATTRIB_BACK       5
#define ATTRIB_BORDER 6
#define ATTRIB_SHADOW 7
```

```
class KKOutline : public TVbxOutline {
public:
BottlePalette* parent;
public:
  KKOutline(TWindow* parent, int id, LPCSTR title,
              int x, int y, int w, int h,
              long initLen=0, void far* initData=0, TModule* module=0) :
     TVbxOutline(parent,id,title,x,y,w,h,initLen,initData,module) {}

  KKOutline(TWindow* parent, int resourceId, TModule* module=0) :
          TVbxOutline(parent, resourceId, module) {}
  void EvOutlineClick(VBXEVENT FAR*);
  DECLARE_RESPONSE_TABLE(KKOutline);
};

class KDialog : public TDialog, public TVbxEventHandler {
public:
    BottlePalette* parent;
    KDialog(TWindow* thewin, int resid) : TDialog(thewin, resid) {}
    void CmAttribApply();
    void CmAttribColor(uint);

    DECLARE_RESPONSE_TABLE(KDialog);
};

class KScrollBar : public TScrollBar {
public:
    int which; // 0 = r, 1 = g, 2 = b
    ColorButtonGadget* colorBtn;

    TEdit* textEdit;
    KScrollBar(TWindow*, int, TEdit*, TModule* module = 0);
    void EvHScroll(uint scrollCode, uint thumbPos, HWND hWndCtl);
    DECLARE_RESPONSE_TABLE(KScrollBar);
};

class KEdit : public TEdit {
public:
    int which; // 0 = r, 1 = g, 2 = b
    ColorButtonGadget* colorBtn;

    TScrollBar* scrollBar;
    KEdit(TWindow*, int, TScrollBar*, TModule* module = 0);
    void EvChar(uint key, uint repeatCount, uint flags);
    DECLARE_RESPONSE_TABLE(KEdit);
};

class AttributePalette : public BottlePalette { //, public TVbxEventHandler {
public:
    KKOutline* AttributeOutline;
    TDialog* AttribDialogs[5];

    TRadioButton* radioBtns[17];
    KScrollBar*   scrollBar[3];
    TButton*      pushBtns[9];
    KEdit*        textEdit[3];
    TListBox*     listBox[2];
    TButton*      applyBtn;

    int TopAttrib, AttribType;
    int x1,y1, x2,y2;

    TToolBox* colorBox;
    TColor currentColor;
    TColor userColor;
    ColorButtonGadget* userColorButton;
    int currentColorId;

    void CmAttribColor(UINT id);
public:
    AttributePalette(TWindow*, int resId, char* title="UNTITLED", int* visible_ptr=0, int x=-1, int
y=-1);
//  void CmAttribApply();

    DECLARE_RESPONSE_TABLE(AttributePalette);
};

#endif
```

palettes.cpp

```
#include "palettes.h"
#include "bttlapp.rh"
#include <stdio.h>
#include <string.h>
```

```
ColorButtonGadget::ColorButtonGadget(TColor c, TResId bmpResId, int id, TType type, bool enabled,
TState state, bool repeat)
    : TButtonGadget(bmpResId, id, type, enabled, state, repeat)
{
  color = c;
}

void ColorButtonGadget::Paint(TDC& dc)
{
  TButtonGadget::Paint(dc);

  TRect bounds;
  GetInnerRect(bounds);

  dc.SelectObject(TBrush(color));
  dc.Rectangle(bounds);
  dc.RestoreBrush();
}


void CreateBasicPalette(TWindow* parent, BottlePalette** Palette, int *PaletteOpen)
{
  TRect winRect = parent->GetWindowRect();
  (*Palette) = new BottlePalette(parent, "Elements", PaletteOpen, 6+winRect.TopLeft().x,
76+winRect.TopLeft().y);
  (*Palette)->SetupPalette();

  // Column 1
  (*Palette)->AddButton(CM_ARROW, CM_ARROW, TButtonGadget::Down);
  (*Palette)->AddButton(CM_0JUNCTION, CM_0JUNCTION);
  (*Palette)->AddButton(CM_ESOURCE, CM_ESOURCE);
  (*Palette)->AddButton(CM_CAPACITANCE, CM_CAPACITANCE);
  (*Palette)->AddButton(CM_RESISTOR, CM_RESISTOR);
  (*Palette)->AddButton(CM_TRANSFORM, CM_TRANSFORM);
  (*Palette)->AddButton(CM_BOND,CM_BOND);
  (*Palette)->AddButton(CM_BONDCAUS,CM_BONDCAUS);

  // Column 2
  (*Palette)->AddSeparator();
  (*Palette)->AddButton(CM_1JUNCTION, CM_1JUNCTION);
  (*Palette)->AddButton(CM_FSOURCE, CM_FSOURCE);
  (*Palette)->AddButton(CM_INDUCTANCE, CM_INDUCTANCE);
  (*Palette)->AddSeparator();
  (*Palette)->AddButton(CM_GYRATOR, CM_GYRATOR);
  (*Palette)->AddButton(CM_SBOND,CM_SBOND);
  (*Palette)->AddButton(CM_BONDDIR,CM_BONDDIR);
}

void CreateShapePalette(TWindow* parent, BottlePalette** Palette, int *PaletteOpen)
{
  TRect winRect = parent->GetWindowRect();
  (*Palette) = new BottlePalette(parent, "Shapes", PaletteOpen,10+6+winRect.TopLeft().x,
10+76+winRect.TopLeft().y);
  (*Palette)->SetupPalette();

  // Column 1
  (*Palette)->AddButton(CM_ARROW, CM_ARROW, TButtonGadget::Down);
  (*Palette)->AddButton(CM_TEXT,CM_TEXT);
  (*Palette)->AddButton(CM_CIRCLE,CM_CIRCLE);
  (*Palette)->AddButton(CM_BITMAP,CM_BITMAP);

  // Column 2
  (*Palette)->AddSeparator();
  (*Palette)->AddButton(CM_LINE,CM_LINE);
  (*Palette)->AddButton(CM_SQUARE,CM_SQUARE);
}

void CreateAttributePalette(TWindow* parent, BottlePalette** Palette, int *PaletteOpen)
{
  TRect winRect = parent->GetWindowRect();
  (*Palette) = new AttributePalette(parent, IDD_ATTRIBUTEPALETTE, "Attributes", PaletteOpen,
20+6+winRect.TopLeft().x, 20+76+winRect.TopLeft().y);
}

void CreateCompoundPalette(TWindow* parent, BottlePalette** Palette, int *PaletteOpen)
{
  TRect winRect = parent->GetWindowRect();
  (*Palette) = new CompoundPalette(parent, IDD_COMPOUNDPALETTE, "Compound/Complex", PaletteOpen,
20+6+winRect.TopLeft().x, 20+76+winRect.TopLeft().y);
}

CompoundPalette::CompoundPalette(TWindow* thewin, int resId, char* title, int* visible_ptr, int x, int
y)
    : BottlePalette(thewin, title, visible_ptr, x, y)
{
    SetupPalette(resId);
```

```
    CompoundLink = new TRadioButton(ResDialog, IDC_COMPOUND_LINK);
    CompoundLink->Create();

    CompoundToolsLoc = new TControl(ResDialog, IDC_COMPOUND_TOOLS);
    CompoundToolsLoc->Create();

    // Create Toolbox
    toolbox = new TToolBox(ResDialog,4,2,TToolBox::Horizontal);
    toolbox->Create();
    toolbox->BringWindowToTop();

    // Add Buttons
    toolbox->Insert(*new
TButtonGadget(CM_ARROW,CM_ARROW,TButtonGadget::Exclusive,true,TButtonGadget::Down));
    toolbox->Insert(*new
TButtonGadget(CM_COMPOUND_EMPTY,CM_COMPOUND_EMPTY,TButtonGadget::Exclusive,true));
    toolbox->Insert(*new TButtonGadget(CM_COMPOUND,CM_COMPOUND,TButtonGadget::Exclusive,true));
    toolbox->Insert(*new TButtonGadget(CM_EMPTY,CM_EMPTY,TButtonGadget::Exclusive,false));
    toolbox->Insert(*new
TButtonGadget(CM_COMPOUND_TOGETHER,CM_COMPOUND_TOGETHER,TButtonGadget::Command,true));
    toolbox->Insert(*new
TButtonGadget(CM_COMPOUND_APART,CM_COMPOUND_APART,TButtonGadget::Command,true));
    toolbox->Insert(*new TButtonGadget(CM_INPUT,CM_INPUT,TButtonGadget::Command,true));
    toolbox->Insert(*new TButtonGadget(CM_OUTPUT,CM_OUTPUT,TButtonGadget::Command,true));
    toolbox->LayoutSession();

    // Center Toolbox in sunken rectangle
    TRect toolRect = toolbox->GetWindowRect();
    TRect compRect = CompoundToolsLoc->GetWindowRect();
    TPoint ulPoint(compRect.left, compRect.top);       ScreenToClient(ulPoint);
    TPoint lrPoint(compRect.right, compRect.bottom);   ScreenToClient(lrPoint);
    TRect newRect(TPoint(ulPoint.x+(lrPoint.x - ulPoint.x - toolRect.Width()) / 2,
                         ulPoint.y+(lrPoint.y - ulPoint.y - toolRect.Height())/ 2),
                  TSize (toolRect.Width(),toolRect.Height()));
    toolbox->MoveWindow(newRect,true);

    // Link to VBX Control
//  CompoundOutline = new TVbxOutline(ResDialog,IDC_COMPOUND);
    CompoundOutline = new KOutline(ResDialog,IDC_COMPOUND);
    CompoundOutline->Create();

    // Initialize Outline Object
    int which = 1;

    UpdateListBox(thewin, CompoundOutline, ".\\MODELS", 1, which);
    CompoundOutline->SetPropListIndex(1);
}

void CompoundPalette::UpdateListBox (TWindow* thewin, /*TVbx*/KOutline* outline, char *filespec, int
level, int& which)
{
    int  i,j,len;
    char tmpBuf[80], tmpBuf2[30];
    TListBox tmpBox(thewin,0,0,0,10,10);
    tmpBox.Attr.Style += ~WS_VISIBLE;
    tmpBox.Create();

    // Get Directories
    sprintf(tmpBuf,"%s\\*.*",filespec);
    tmpBox.DirectoryList(DDL_DIRECTORY+DDL_EXCLUSIVE,tmpBuf);

    j = tmpBox.GetCount();
    for (i=0;i<j;i++) {
      len = tmpBox.GetString(tmpBuf,i);
      if (len-1 > 0) {
      tmpBuf[len-1] = 0;
      if (strcmp("[..",tmpBuf)!=0) {
          outline->AddItem(&tmpBuf[1],level);
          which++;

          sprintf(tmpBuf2,"%s\\%s",filespec,&tmpBuf[1]);
          UpdateListBox(thewin,outline,tmpBuf2,level+1,which);
      }
      }
    }

    // Get Files
    tmpBox.ClearList();
    sprintf(tmpBuf,"%s\\*.*",filespec);
    tmpBox.DirectoryList(DDL_READWRITE,tmpBuf);

    j = tmpBox.GetCount();
    for (i=0;i<j;i++) {
      len = tmpBox.GetString(tmpBuf,i);
      outline->AddItem(tmpBuf,level);
      which++;
```

```
    }
}


int CALLBACK fillFontList(const LOGFONT *lf, const TEXTMETRIC *lptm, int nFontType, LPARAM lpData);

#define IDC_COLOR_1  401
#define IDC_COLOR_2  402
#define IDC_COLOR_3  403
#define IDC_COLOR_4  404
#define IDC_COLOR_5  405
#define IDC_COLOR_6  406
#define IDC_COLOR_7  407
#define IDC_COLOR_8  408
#define IDC_COLOR_9  409
#define IDC_COLOR_10 410
#define IDC_COLOR_11 411

AttributePalette::AttributePalette(TWindow* thewin, int resId, char* title, int* visible_ptr, int x,
int y)
    : BottlePalette(thewin, title, visible_ptr, x, y)
{
    SetupPalette(resId,5439);

    TControl placement(ResDialog, IDC_ATTRIBLOC);
    placement.Create();

    TRect compRect = placement.GetWindowRect();
    TPoint ulPoint(compRect.left, compRect.top);
    TPoint lrPoint(compRect.right, compRect.bottom);

    ResDialog->ScreenToClient(ulPoint);
    x1 = ulPoint.x; x2 = lrPoint.x;
    y1 = ulPoint.y; y2 = lrPoint.y;

    // Link to Subdialogs
    AttribDialogs[0] = new TDialog(ResDialog,IDD_ATTRIBGRID);
    AttribDialogs[0]->Create();
    AttribDialogs[0]->MoveWindow(TRect(TPoint(x1,y1),
                                                TSize(((TWindow*)AttribDialogs[0])->Attr.W,
                                                        ((TWindow*)AttribDialogs[0])->Attr.H)));

    AttribDialogs[1] = new TDialog(ResDialog,IDD_ATTRIBALIGN);
    AttribDialogs[1]->Create();
    AttribDialogs[1]->MoveWindow(TRect(TPoint(x1,y1),
                                                TSize(((TWindow*)AttribDialogs[1])->Attr.W,
                                                        ((TWindow*)AttribDialogs[1])->Attr.H)));

    AttribDialogs[2] = new KDialog(ResDialog,IDD_ATTRIBCOLOR);
    ((KDialog*)AttribDialogs[2])->parent = this;
    AttribDialogs[2]->Create();
    AttribDialogs[2]->MoveWindow(TRect(TPoint(x1,y1),
                                                TSize(((TWindow*)AttribDialogs[2])->Attr.W,
                                                        ((TWindow*)AttribDialogs[2])->Attr.H)));

    AttribDialogs[3] = new TDialog(ResDialog,IDD_ATTRIBDISPLAY);
    AttribDialogs[3]->Create();
    AttribDialogs[3]->MoveWindow(TRect(TPoint(x1,y1),
                                                TSize(((TWindow*)AttribDialogs[3])->Attr.W,
                                                        ((TWindow*)AttribDialogs[3])->Attr.H)));

    AttribDialogs[4] = new TDialog(ResDialog,IDD_ATTRIBFONT);
    AttribDialogs[4]->Create();
    AttribDialogs[4]->MoveWindow(TRect(TPoint(x1,y1),
                                                TSize(((TWindow*)AttribDialogs[4])->Attr.W,
                                                        ((TWindow*)AttribDialogs[4])->Attr.H)));

    AttribType = 1;
    TopAttrib = 0;
    AttribDialogs[TopAttrib]->ShowWindow(SW_SHOWNORMAL);


    // Link to VBX Control
    AttributeOutline = new KKOutline(ResDialog,IDC_OUTLINE4);
    AttributeOutline->Create();
    AttributeOutline->parent = this;

    // Initialize Outline Object
    AttributeOutline->SetPropList("Grid",1);
    AttributeOutline->SetPropList("Alignment",2);
    AttributeOutline->SetPropList("Color",3);
    AttributeOutline->SetPropList("Foreground",4);
    AttributeOutline->SetPropList("Background",5);
    AttributeOutline->SetPropList("Border",6);
    AttributeOutline->SetPropList("Shadow",7);
    AttributeOutline->SetPropList("Display",8);
```

```
AttributeOutline->SetPropList("Font",9);

AttributeOutline->SetPropIndent(1,1);
AttributeOutline->SetPropIndent(1,2);
AttributeOutline->SetPropIndent(1,3);
AttributeOutline->SetPropIndent(2,4);
AttributeOutline->SetPropIndent(2,5);
AttributeOutline->SetPropIndent(2,6);
AttributeOutline->SetPropIndent(2,7);
AttributeOutline->SetPropIndent(1,8);
AttributeOutline->SetPropIndent(1,9);
AttributeOutline->SetPropListIndex(1);

((KDialog*)ResDialog)->parent = this;
applyBtn = new TButton(ResDialog,IDC_ATTRIB_APPLY);
applyBtn->Create();

// Link to Controls
// Grid
radioBtns[0] = new TRadioButton(AttribDialogs[0],IDC_GRID_SHOW);
radioBtns[0]->Create();

radioBtns[1] = new TRadioButton(AttribDialogs[0],IDC_GRID_SNAP);
radioBtns[1]->Create();

// Alignment
radioBtns[2] = new TRadioButton(AttribDialogs[1],IDC_VERT_TOP);
radioBtns[2]->Create();

radioBtns[3] = new TRadioButton(AttribDialogs[1],IDC_VERT_CENTER);
radioBtns[3]->Create();

radioBtns[4] = new TRadioButton(AttribDialogs[1],IDC_VERT_BOTTOM);
radioBtns[4]->Create();

radioBtns[5] = new TRadioButton(AttribDialogs[1],IDC_VERT_NONE);
radioBtns[5]->Create();

radioBtns[6] = new TRadioButton(AttribDialogs[1],IDC_HORZ_LEFT);
radioBtns[6]->Create();

radioBtns[7] = new TRadioButton(AttribDialogs[1],IDC_HORZ_CENTER);
radioBtns[7]->Create();

radioBtns[8] = new TRadioButton(AttribDialogs[1],IDC_HORZ_RIGHT);
radioBtns[8]->Create();

radioBtns[9] = new TRadioButton(AttribDialogs[1],IDC_HORZ_NONE);
radioBtns[9]->Create();

radioBtns[10] = new TRadioButton(AttribDialogs[3],IDC_DISPLAY_NOBORDER);
radioBtns[10]->Create();

radioBtns[11] = new TRadioButton(AttribDialogs[3],IDC_DISPLAY_BORDER);
radioBtns[11]->Create();

radioBtns[12] = new TRadioButton(AttribDialogs[3],IDC_DISPLAY_SHADOW);
radioBtns[12]->Create();

// Font
listBox[0] = new TListBox(AttribDialogs[4],IDC_FONT_NAME);
listBox[0]->Create();

TWindowDC dc(*this);
dc.EnumFonts(NULL,fillFontList,(char *) listBox[0]);

listBox[1] = new TListBox(AttribDialogs[4],IDC_FONT_SIZE);
listBox[1]->Attr.Style |= ~LBS_SORT;
listBox[1]->Create();
listBox[1]->AddString("8");
listBox[1]->AddString("9");
listBox[1]->AddString("10");
listBox[1]->AddString("12");
listBox[1]->AddString("14");
listBox[1]->AddString("18");
listBox[1]->AddString("24");
listBox[1]->AddString("30");
listBox[1]->AddString("36");
listBox[1]->AddString("42");
listBox[1]->AddString("48");
listBox[1]->AddString("64");
listBox[1]->AddString("72");

radioBtns[13] = new TRadioButton(AttribDialogs[4],IDC_FONT_BOLD);
radioBtns[13]->Create();

radioBtns[14] = new TRadioButton(AttribDialogs[4],IDC_FONT_ITALIC);
```

```
    radioBtns[14]->Create();

    radioBtns[15] = new TRadioButton(AttribDialogs[4],IDC_FONT_UNDERLINE);
    radioBtns[15]->Create();

    radioBtns[16] = new TRadioButton(AttribDialogs[4],IDC_FONT_STRIKEOUT);
    radioBtns[16]->Create();

    // Color
    TControl ColorPlacement(AttribDialogs[2],IDC_COLOR_GROUP);
    ColorPlacement.Create();

    colorBox = new TToolBox(AttribDialogs[2],6);
    colorBox->Create();
    colorBox->Insert(*new
ColorButtonGadget(TColor::Black,CM_EMPTY,IDC_COLOR_1,TButtonGadget::Exclusive,true,TButtonGadget::Down
));
    colorBox->Insert(*new
ColorButtonGadget(TColor::Gray,CM_EMPTY,IDC_COLOR_2,TButtonGadget::Exclusive,true));
    colorBox->Insert(*new
ColorButtonGadget(TColor::LtBlue,CM_EMPTY,IDC_COLOR_3,TButtonGadget::Exclusive,true));
    colorBox->Insert(*new
ColorButtonGadget(TColor::LtCyan,CM_EMPTY,IDC_COLOR_4,TButtonGadget::Exclusive,true));
    colorBox->Insert(*new
ColorButtonGadget(TColor::LtGray,CM_EMPTY,IDC_COLOR_5,TButtonGadget::Exclusive,true));
    colorBox->Insert(*(userColorButton = new
ColorButtonGadget(TColor::Black,CM_EMPTY,IDC_COLOR_11,TButtonGadget::Exclusive,true)));

    colorBox->Insert(*new
ColorButtonGadget(TColor::LtGreen,CM_EMPTY,IDC_COLOR_6,TButtonGadget::Exclusive,true));
    colorBox->Insert(*new
ColorButtonGadget(TColor::LtMagenta,CM_EMPTY,IDC_COLOR_7,TButtonGadget::Exclusive,true));
    colorBox->Insert(*new
ColorButtonGadget(TColor::LtRed,CM_EMPTY,IDC_COLOR_8,TButtonGadget::Exclusive,true));
    colorBox->Insert(*new
ColorButtonGadget(TColor::LtYellow,CM_EMPTY,IDC_COLOR_9,TButtonGadget::Exclusive,true));
    colorBox->Insert(*new
ColorButtonGadget(TColor::White,CM_EMPTY,IDC_COLOR_10,TButtonGadget::Exclusive,true));
    colorBox->LayoutSession();
    colorBox->BringWindowToTop();

    // Center Toolbox in sunken rectangle
    TRect toolRect = colorBox->GetWindowRect();
    TRect ColorCompRect = ColorPlacement.GetWindowRect();
    TPoint ColorulPoint(ColorCompRect.left, ColorCompRect.top);       AttribDialogs[2]-
>ScreenToClient(ColorulPoint);
    TPoint ColorlrPoint(ColorCompRect.right, ColorCompRect.bottom);   AttribDialogs[2]-
>ScreenToClient(ColorlrPoint);
    TRect newRect(TPoint(ColorulPoint.x+(ColorlrPoint.x - ColorulPoint.x - toolRect.Width()) / 2,
                         ColorulPoint.y+(ColorlrPoint.y - ColorulPoint.y -
toolRect.Height())/ 2),
                  TSize (toolRect.Width(),toolRect.Height()));
    colorBox->MoveWindow(newRect,true);

    // Scrollbars and Text
    textEdit[0] = new KEdit(AttribDialogs[2],IDC_TEXT_RED,NULL);   textEdit[0]->Create();
    textEdit[1] = new KEdit(AttribDialogs[2],IDC_TEXT_GREEN,NULL); textEdit[1]->Create();
    textEdit[2] = new KEdit(AttribDialogs[2],IDC_TEXT_BLUE,NULL);  textEdit[2]->Create();

    scrollBar[0] = new KScrollBar(AttribDialogs[2],IDC_SCROLL_RED,textEdit[0]);   scrollBar[0]-
>Create();
    scrollBar[0]->SetRange(0,255); scrollBar[0]->SetPosition(0);
    scrollBar[1] = new KScrollBar(AttribDialogs[2],IDC_SCROLL_GREEN,textEdit[1]); scrollBar[1]-
>Create();
    scrollBar[1]->SetRange(0,255); scrollBar[1]->SetPosition(0);
    scrollBar[2] = new KScrollBar(AttribDialogs[2],IDC_SCROLL_BLUE,textEdit[2]);  scrollBar[2]-
>Create();
    scrollBar[2]->SetRange(0,255); scrollBar[2]->SetPosition(0);

    textEdit[0]->scrollBar = scrollBar[0]; textEdit[0]->which = 0;
    textEdit[1]->scrollBar = scrollBar[1]; textEdit[1]->which = 1;
    textEdit[2]->scrollBar = scrollBar[2]; textEdit[2]->which = 2;

    textEdit[0]->colorBtn = scrollBar[0]->colorBtn = userColorButton; scrollBar[0]->which = 0;
    textEdit[0]->colorBtn = scrollBar[1]->colorBtn = userColorButton; scrollBar[1]->which = 1;
    textEdit[0]->colorBtn = scrollBar[2]->colorBtn = userColorButton; scrollBar[2]->which = 2;

    userColor = TColor::Black;
    currentColor = TColor::Black;
    currentColorId = 1;
}

int CALLBACK fillFontList(const LOGFONT *lf, const TEXTMETRIC *lptm, int nFontType, LPARAM lpData)
{
  ((TListBox*)lpData)->AddString(lf->lfFaceName);
  return 1;
}
```

```
DEFINE_RESPONSE_TABLE1(KKOutline, TVbxOutline)
  EV_VBXEVENTNAME(IDC_OUTLINE4,"Click",EvOutlineClick),
END_RESPONSE_TABLE;

DEFINE_RESPONSE_TABLE1(AttributePalette, BottlePalette)
  EV_COMMAND_AND_ID(IDC_COLOR_1, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_2, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_3, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_4, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_5, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_6, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_7, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_8, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_9, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_10, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_11, CmAttribColor),
END_RESPONSE_TABLE;

DEFINE_RESPONSE_TABLE1(KScrollBar, TScrollBar)
  EV_WM_HSCROLL,
END_RESPONSE_TABLE;

DEFINE_RESPONSE_TABLE1(KEdit, TEdit)
  EV_WM_CHAR,
END_RESPONSE_TABLE;

void AttributePalette::CmAttribColor(UINT id)
{
/*  MessageBeep(-1);
  switch (id) {
     case IDC_COLOR_1:  currentColor = TColor::Black;     break;
     case IDC_COLOR_2:  currentColor = TColor::Gray;      break;
     case IDC_COLOR_3:  currentColor = TColor::LtBlue;    MessageBeep(-1); break;
     case IDC_COLOR_4:  currentColor = TColor::LtCyan;    break;
     case IDC_COLOR_5:  currentColor = TColor::LtGray;    break;
     case IDC_COLOR_11: currentColor = userColor;         break;
     case IDC_COLOR_6:  currentColor = TColor::LtGreen;   break;
     case IDC_COLOR_7:  currentColor = TColor::LtMagenta; break;
     case IDC_COLOR_8:  currentColor = TColor::LtRed;     break;
     case IDC_COLOR_9:  currentColor = TColor::LtYellow;  break;
     case IDC_COLOR_10: currentColor = TColor::White;     break;
  };
*/
}

void KDialog::CmAttribColor(UINT id)
{
  AttributePalette* tp = (AttributePalette*) parent;
  tp->currentColorId = id;
  switch (id) {
     case IDC_COLOR_1:  tp->currentColor = TColor::Black;     break;
     case IDC_COLOR_2:  tp->currentColor = TColor::Gray;      break;
     case IDC_COLOR_3:  tp->currentColor = TColor::LtBlue;    break;
     case IDC_COLOR_4:  tp->currentColor = TColor::LtCyan;    break;
     case IDC_COLOR_5:  tp->currentColor = TColor::LtGray;    break;
     case IDC_COLOR_11: tp->currentColor = tp->userColor;     break;
     case IDC_COLOR_6:  tp->currentColor = TColor::LtGreen;   break;
     case IDC_COLOR_7:  tp->currentColor = TColor::LtMagenta; break;
     case IDC_COLOR_8:  tp->currentColor = TColor::LtRed;     break;
     case IDC_COLOR_9:  tp->currentColor = TColor::LtYellow;  break;
     case IDC_COLOR_10: tp->currentColor = TColor::White;     break;
  };
}

KScrollBar::KScrollBar(TWindow* thewin, int resId, TEdit* te, TModule* module)
    : TScrollBar(thewin, resId, module)
{
  textEdit = te;
}

void KScrollBar::EvHScroll(uint scrollCode, uint thumbPos, HWND hWndCtl)
{
  TScrollBar::EvHScroll(scrollCode,thumbPos,hWndCtl);
  char tmps[10];
  sprintf(tmps,"%d",GetPosition());
  textEdit->Transfer(tmps,tdSetData /*TTransferDirection::tdSetData*/);
  int r,g,b;
  r = colorBtn->color.Red();
  g = colorBtn->color.Green();
  b = colorBtn->color.Blue();
  if (which == 0) r = GetPosition();
  else if (which == 1) g = GetPosition();
  else b = GetPosition();
  colorBtn->color = TColor(r,g,b);
  colorBtn->Redraw();
}
```

```
void ColorButtonGadget::Redraw()
{
  Invalidate();
}

KEdit::KEdit(TWindow* thewin, int resId, TScrollBar* ts, TModule* module)
    : TEdit(thewin,resId,3,module)
{
  scrollBar = ts;
  Transfer("0",tdGetData);
}

void KEdit::EvChar(uint key, uint repeatCount, uint flags)
{
  TEdit::EvChar(key,repeatCount,flags);
  char tmps[10];
  int newPosition;
  Transfer(tmps,/*TTransferDirection::*/tdGetData);
  sscanf(tmps,"%d",&newPosition);
  scrollBar->SetPosition(newPosition);
  int r,g,b;
  r = colorBtn->color.Red();
  g = colorBtn->color.Green();
  b = colorBtn->color.Blue();
  if (which == 0) r = newPosition;
  else if (which == 1) g = newPosition;
  else b = newPosition;
  colorBtn->color = TColor(r,g,b);
  colorBtn->Redraw();
}

DEFINE_RESPONSE_TABLE2(KDialog, TDialog, TVbxEventHandler)
  EV_COMMAND(IDC_ATTRIB_APPLY, CmAttribApply),
  EV_COMMAND_AND_ID(IDC_COLOR_1,CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_2, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_3, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_4, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_5, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_6, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_7, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_8, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_9, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_10, CmAttribColor),
  EV_COMMAND_AND_ID(IDC_COLOR_11, CmAttribColor),
END_RESPONSE_TABLE;

#include "bttlapp.h"

void KDialog::CmAttribApply()
{
  BasicObject* curObject = NULL;
  BondGraphDocument* theDoc;
  AttributePalette* tp = ((AttributePalette*) parent);
  BottleApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), BottleApp);
  switch (tp->AttribType) {
     case 1: // Grid Sub Palette
        if (theDoc = ((BondGraphDocument*) theApp->GetDocManager()->GetCurrentDoc())) {
          theDoc->gridShow = (tp->radioBtns[0]->GetCheck() == BF_CHECKED);
          theDoc->gridSnap = (tp->radioBtns[1]->GetCheck() == BF_CHECKED);
          theDoc->NextView(0)->GetWindow()->Invalidate();
        }
        break;

     case 2: // Align Sub Palette
        if (theDoc = ((BondGraphDocument*) theApp->GetDocManager()->GetCurrentDoc())) {
          // Get Bounds
          int nx,ny,minx=10000, maxx=-10000, miny=10000, maxy=-10000;
          theDoc->Start();
          while (curObject = theDoc->Current()) {
             if (curObject->selected) {
                if (curObject->x < minx) minx = curObject->x;
                if (curObject->y < miny) miny = curObject->y;
                if (curObject->x + curObject->w > maxx)
                     maxx = curObject->x + curObject->w;
                if (curObject->y + curObject->h > maxy)
                     maxy = curObject->y + curObject->h;
             }
             theDoc->Next();
          }

          // Vertical
          if (tp->radioBtns[2]->GetCheck() == BF_CHECKED) {//Top
             ny = miny;
          } else if (tp->radioBtns[3]->GetCheck() == BF_CHECKED) {//Center
             ny = miny + (maxy - miny) / 2 - 20;
          } else if (tp->radioBtns[4]->GetCheck() == BF_CHECKED) {//Bottom
```

```
            ny = maxy - 40;
        } else if (tp->radioBtns[5]->GetCheck() == BF_CHECKED) {//None
            ny = -1;
        }

        // Horizontal
        if (tp->radioBtns[6]->GetCheck() == BF_CHECKED) {//Left
            nx = minx;
        } else if (tp->radioBtns[7]->GetCheck() == BF_CHECKED) {//Center
            nx = minx + (maxx - minx) / 2 - 20;
        } else if (tp->radioBtns[8]->GetCheck() == BF_CHECKED) {//Right
            nx = maxx - 40;
        } else if (tp->radioBtns[9]->GetCheck() == BF_CHECKED) {//None
            nx = -1;
        }

        theDoc->Start();
        while (curObject = theDoc->Current()) {
            if (curObject->selected) {
                if (nx>=0) curObject->x = nx;
                if (ny>=0) curObject->y = ny;
            }
            theDoc->Next();
        }

        theDoc->NextView(0)->GetWindow()->Invalidate();
    }
    break;

case 4:
    if (theDoc = ((BondGraphDocument*) theApp->GetDocManager()->GetCurrentDoc())) {
        if (tp->currentColorId == IDC_COLOR_11) tp->currentColor = tp->userColorButton->color;

        theDoc->Start();
        while (curObject = theDoc->Current()) {
            if (curObject->selected) {
                curObject->foreground[0] = tp->currentColor.Red();
                curObject->foreground[1] = tp->currentColor.Green();
                curObject->foreground[2] = tp->currentColor.Blue();
            }
            theDoc->Next();
        }
        theDoc->NextView(0)->GetWindow()->Invalidate();
    }
    break;

case 5:
    if (theDoc = ((BondGraphDocument*) theApp->GetDocManager()->GetCurrentDoc())) {
        if (tp->currentColorId == IDC_COLOR_11) tp->currentColor = tp->userColorButton->color;

        theDoc->Start();
        while (curObject = theDoc->Current()) {
            if (curObject->selected) {
                curObject->background[0] = tp->currentColor.Red();
                curObject->background[1] = tp->currentColor.Green();
                curObject->background[2] = tp->currentColor.Blue();
            }
            theDoc->Next();
        }
        theDoc->NextView(0)->GetWindow()->Invalidate();
    }
    break;

case 6:
    if (theDoc = ((BondGraphDocument*) theApp->GetDocManager()->GetCurrentDoc())) {
        if (tp->currentColorId == IDC_COLOR_11) tp->currentColor = tp->userColorButton->color;

        theDoc->Start();
        while (curObject = theDoc->Current()) {
            if (curObject->selected) {
                curObject->border[0] = tp->currentColor.Red();
                curObject->border[1] = tp->currentColor.Green();
                curObject->border[2] = tp->currentColor.Blue();
            }
            theDoc->Next();
        }
        theDoc->NextView(0)->GetWindow()->Invalidate();
    }
    break;

case 7:
    if (theDoc = ((BondGraphDocument*) theApp->GetDocManager()->GetCurrentDoc())) {
        if (tp->currentColorId == IDC_COLOR_11) tp->currentColor = tp->userColorButton->color;

        theDoc->Start();
        while (curObject = theDoc->Current()) {
            if (curObject->selected) {
```

```
                curObject->shadow[0] = tp->currentColor.Red();
                curObject->shadow[1] = tp->currentColor.Green();
                curObject->shadow[2] = tp->currentColor.Blue();
            }
            theDoc->Next();
          }
          theDoc->NextView(0)->GetWindow()->Invalidate();
        }
        break;

    case 8: // Display
        if (theDoc = ((BondGraphDocument*) theApp->GetDocManager()->GetCurrentDoc())) {
          int b;
          if (tp->radioBtns[10]->GetCheck() == BF_CHECKED) // Display Mode No Border
             b = DISPLAY_NOBORDER;
          else if (tp->radioBtns[11]->GetCheck() == BF_CHECKED) // Display Mode No Border
             b = DISPLAY_BOX;
          else if (tp->radioBtns[12]->GetCheck() == BF_CHECKED) // Display Mode No Border
             b = DISPLAY_SHADOW;

          theDoc->Start();
          while (curObject = theDoc->Current()) {
             if (curObject->selected) {
                if (curObject->typeId != BG_BOND) {
                  curObject->displayMode = b;
                }
             }
             theDoc->Next();
          }

          theDoc->NextView(0)->GetWindow()->Invalidate();
        }
        break;

    case 9: // Font
        if (theDoc = ((BondGraphDocument*) theApp->GetDocManager()->GetCurrentDoc())) {
          int thesize,bold=FW_NORMAL, italic=FALSE, strikeout=FALSE, underline=FALSE;

          if (tp->radioBtns[13]->GetCheck() == BF_CHECKED)
             bold = FW_BOLD;
          italic    = (tp->radioBtns[14]->GetCheck() == BF_CHECKED);
          underline = (tp->radioBtns[15]->GetCheck() == BF_CHECKED);
          strikeout = (tp->radioBtns[16]->GetCheck() == BF_CHECKED);
          char fontName[30];
          tp->listBox[1]->GetSelString(fontName,29);
          sscanf(fontName,"%d",&thesize);
          tp->listBox[0]->GetSelString(fontName,29);

          theDoc->Start();
          while (curObject = theDoc->Current()) {
             if (curObject->selected) {
                if (curObject->typeId != BG_BOND) {
                  curObject->labelFont[4] = bold;
                  curObject->labelFont[5] = italic;
                  curObject->labelFont[6] = underline;
                  curObject->labelFont[7] = strikeout;
                  curObject->labelFont[1] = thesize;
                  free (curObject->labelFontName);
                  (curObject->labelFontName) = (char *) malloc (strlen(fontName)*sizeof(char));
                  strcpy(curObject->labelFontName,fontName);
                }
             }
             theDoc->Next();
          }

          theDoc->NextView(0)->GetWindow()->Invalidate();
        }
        break;
  };
}

void KKOutline::EvOutlineClick(VBXEVENT FAR*)
{
  AttributePalette* tparent = (AttributePalette*) parent;
  tparent->AttributeOutline->GetPropListIndex(tparent->AttribType);
  int oi = tparent->TopAttrib;
  switch (tparent->AttribType) {
     case 1: tparent->TopAttrib = 0; break; // Grid
     case 2: tparent->TopAttrib = 1; break; // Align
     case 3: break;
     case 4: tparent->TopAttrib = 2; break; // Foreground
     case 5: tparent->TopAttrib = 2; break; // Background
     case 6: tparent->TopAttrib = 2; break; // Border
     case 7: tparent->TopAttrib = 2; break; // Shadow
     case 8: tparent->TopAttrib = 3; break; // Display
     case 9: tparent->TopAttrib = 4; break; // Font
  };
```

```
  if (oi != tparent->TopAttrib) {
     tparent->AttribDialogs[tparent->TopAttrib]->ShowWindow(SW_SHOWNORMAL);
     tparent->AttribDialogs[oi]->ShowWindow(SW_HIDE);
  }
}
```

bttldoc.h

```
#if !defined(__bttldoc_h)              // Sentry, use file only if it's not already included.
#define __bttldoc_h

/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.apx Application
    FILE:         bttldoc.h
    AUTHOR:       Richard K. Branton


     OVERVIEW
     ========
     Class definition for BondGraphDocument (TFileDocument).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "bttlapp.rh"            // Definition of all resources.
#include "bttlobj.h"

// An array is used to hold the objects.
#include <classlib\arrays.h>
typedef TArray<BasicObject *> TObjectArray;
typedef TArrayAsVectorIterator<BasicObject *> TObjectArrayIterator;

//{{TFileDocument = BondGraphDocument}}
class BondGraphDocument : public TFileDocument {
//protected:
public:
    // The actual object data is stored in an array
    TObjectArray* ObjectArray;
    TObjectArrayIterator* ObjectIterator;
    int gridShow, gridSnap;
    int isSubDocument;
    BasicObject* parentObject;
   int numberingShow;
   BOOL ReadFromifStream(ifstream* is, TObjectArray* theArray);

public:
    BondGraphDocument(TDocument* parent = 0);
    ~BondGraphDocument();

    // Virtual functions of TDocument and TFileDocument
    // These are used to support the OWL document-view structure
    BOOL IsOpen();
    BOOL Open(int mode, /*LPCSTR*/const char far* path = 0);
    BOOL Commit(BOOL force = FALSE);
    BOOL Revert(BOOL clear = FALSE);
    BOOL Close();

//  BOOL ReadFromStream(/*TInStream**/istrstream*, TObjectArray* theArray);
//  BOOL WriteToStream(/*TOutStream**/ostrstream*, TObjectArray* theArray, BOOL writeAll = TRUE);

    // Data access functions provided for the various views which
    // might use this document.
    const BasicObject* GetObject(UINT index);
    void               AddObject(BasicObject* object);
    BOOL                  DeleteObject(UINT index);

    // An iterator provided to loop through the objects in the document
    void                  Start();
    /*const*/ BasicObject* Current();
    void               Next();
    void               Reset();

    BOOL                  inClipboard;  // Should paste be enabled
    BOOL                  inSelected; // Should cut,copy,delete be enabled

    void DeleteObjects(BOOL AllObjects = TRUE);
    void DuplicateObjects(BOOL AllObjects = TRUE);

    BasicObject* inputObject;
    BasicObject* outputObject;
   BasicObject* docObject;
};    //{{BondGraphDocument}}
```

```
BOOL ReadFromStream(istrstream*, TObjectArray* theArray);
BOOL WriteToStream(ostrstream*, TObjectArray* theArray, BOOL writeAll = TRUE, BOOL writeCount = TRUE);


#endif                                          // __bttldoc_h sentry.
```

bttldoc.cpp

```
/*  Project bottle
     Massachusetts Institute of Technology
     Copyright © 1995. All Rights Reserved.

     SUBSYSTEM:    bottle.apx Application
     FILE:         bttldoc.cpp
     AUTHOR:       Richard K. Branton


     OVERVIEW
     ========
     Source file for implementation of BondGraphDocument (TFileDocument).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "bttldoc.h"
#include "obj_comp.h"

#include <stdio.h>
#include <stdlib.h>

//{{BondGraphDocument Implementation}}

BondGraphDocument::BondGraphDocument (TDocument* parent):
     TFileDocument(parent)
{
    // INSERT>> Your constructor code here.
    ObjectArray = 0;      // Initialize ObjectArray and Iterator
    ObjectIterator = 0;
    inClipboard = FALSE;
    inSelected = FALSE;
    gridShow = FALSE;
    gridSnap = FALSE;
    isSubDocument = FALSE;
    parentObject = 0;
    numberingShow = FALSE;

//  ObjectArray = new TObjectArray(10,0,10);
}

BondGraphDocument::~BondGraphDocument ()
{
    // INSERT>> Your destructor code here.
    delete ObjectIterator;    // Free memory used by ObjectArray and Iterator
    delete ObjectArray;
}

// Virtual functions of TDocument and TFileDocument.
// These are used to support the OWL document-view structure.

BOOL BondGraphDocument::IsOpen()
{
    return (ObjectArray != 0);
}

BOOL /*BondGraphDocument::*/ ReadFromStream(istrstream* /*TInStream* */ is, TObjectArray* theArray)
{
    // Read in the number of objects in the file.
    UINT numObjects;
    *is >> numObjects;

    // Read in each object and add to array.
    BasicObject* objectPtr;
    int objectTypeID;// In the future, might make this a string to simplify
                     // for human readers.

    char c;
    while (numObjects--) {
        *is >> c >> objectTypeID; // Read in the unique object type id

        // Allocate space for the specific type of object.
        objectPtr = NewObject(objectTypeID);
        if (!objectPtr) return FALSE;
```

```
        // In the future, should add checks to make sure object was
        // read in correctly.
        *is >> (*objectPtr);      // Read in the specific object

        // Add the object to the document
        theArray->Add(objectPtr);
    }

    // Fix Links For Bonds
    numObjects = theArray->GetItemsInContainer();
    for (int i=numObjects-1; i>=0; i--)
      if ((*theArray)[i]->typeId == BG_BOND)
          ((ObjectBond*)((*theArray)[i]))->FixLinks(theArray);

    // Fix Links For Compound Objects
    TObjectArray* tmpArray = new TObjectArray(4,0,4);
    numObjects = theArray->GetItemsInContainer();
    for (i=0;i<numObjects;i++)
      if ((*theArray)[i]->typeId == BG_COMPOUND)
          tmpArray->Add((*theArray)[i]);

    numObjects = tmpArray->GetItemsInContainer();
    for (i=numObjects-1;i>=0;i--) {
      ((ObjectCompound*)((*tmpArray)[i]))->FixLinks(theArray);
      tmpArray->Detach(i);
    }
    delete tmpArray;

    // Empty Distinct Ids
    numObjects = theArray->GetItemsInContainer();
    for (i=numObjects-1; i>=0; i--)
        (*theArray)[i]->distinctId = -1;

    return TRUE;
}

BOOL /*BondGraphDocument::*/ ReadFromifStream(ifstream* is, TObjectArray* theArray)
{
    FILE *fp =  fopen ("test.txt","w");
    fprintf(fp,"numobj...\n"); fflush(fp);
    // Read in the number of objects in the file.
    UINT numObjects;
    *is >> numObjects;

    // Read in each object and add to array.
    BasicObject* objectPtr;
    int objectTypeID;// In the future, might make this a string to simplify
                          // for human readers.

    fprintf(fp,"numObjects=%d\n",numObjects);
    fprintf(fp,"loop start..\n"); fflush(fp);

    char c;
    while (numObjects--) {
        *is >> c >> objectTypeID; // Read in the unique object type id
        fprintf(fp,"numobjects = %d, objectTypeId = %d\n",numObjects,objectTypeID); fflush(fp);

        // Allocate space for the specific type of object.
        objectPtr = NewObject(objectTypeID);
        if (!objectPtr) return FALSE;

        // In the future, should add checks to make sure object was
        // read in correctly.
        *is >> (*objectPtr);      // Read in the specific object

        // Add the object to the document
        theArray->Add(objectPtr);
    }
    fclose(fp);

    // Fix Links For Bonds
    numObjects = theArray->GetItemsInContainer();
    for (int i=numObjects-1; i>=0; i--)
      if ((*theArray)[i]->typeId == BG_BOND)
          ((ObjectBond*)((*theArray)[i]))->FixLinks(theArray);

    // Fix Links For Compound Objects
    TObjectArray* tmpArray = new TObjectArray(4,0,4);
    numObjects = theArray->GetItemsInContainer();
    for (i=0;i<numObjects;i++)
      if ((*theArray)[i]->typeId == BG_COMPOUND)
          tmpArray->Add((*theArray)[i]);

    numObjects = tmpArray->GetItemsInContainer();
    for (i=numObjects-1;i>=0;i--) {
      ((ObjectCompound*)((*tmpArray)[i]))->FixLinks(theArray);
```

```
      tmpArray->Detach(i);
    }
    delete tmpArray;

    // Empty Distinct Ids
    numObjects = theArray->GetItemsInContainer();
    for (i=numObjects-1; i>=0; i--)
        (*theArray)[i]->distinctId = -1;

    return TRUE;
}

int CountSubObjects (ObjectCompound* object)
{
  int total = 0;
  int max = object->subobjects->GetItemsInContainer();
  for (int i=0;i<max;i++) {
     total++;
     if ((*object->subobjects)[i]->typeId == BG_COMPOUND)
        if (((ObjectCompound *)(*object->subobjects)[i])->fileType == 1)
          total += CountSubObjects((ObjectCompound *)object);
  }
  return total;
}

BOOL /*BondGraphDocument::*/WriteToStream(ostrstream* os /*TOutStream* os*/, TObjectArray* theArray,
BOOL writeAll, BOOL writeCount)
{
    // In the future, might want to right a version number and file identifier
    // type first, as well as miscellaneous information. This stuff could be
    // stored in an object and thus to add new settings and stuff, we just
    // need to change the object instead of altering the document class.

    if (theArray) {
        // Write the number of objects to the file.
        int numObjects = theArray->GetItemsInContainer();

        int total=0;
        for (int i=0; i<numObjects; i++)
            if (writeAll == TRUE || (*theArray)[i]->selected == TRUE) {
                total += 1;

                if ((*theArray)[i]->typeId == BG_COMPOUND)
                  total += CountSubObjects((ObjectCompound *)(*theArray)[i]);
            }

        if (writeCount)
          *os << total << '\n';

        // Write each object except bonds.
        for (i=0; i<numObjects; i++)
            if (writeAll || (*theArray)[i]->selected == TRUE)
              if ((*theArray)[i]->typeId != BG_BOND)
                  *os << *((*theArray)[i]) << '\n';

        // Write bonds last.
        for (i=0; i<numObjects; i++)
            if (writeAll || (*theArray)[i]->selected == TRUE)
              if ((*theArray)[i]->typeId == BG_BOND)
                  *os << *((*theArray)[i]) << '\n';

        if (writeCount)
          *os << '\0';
    }
    return TRUE;
}

BOOL BondGraphDocument::Open(int /* mode */, LPCSTR /* path */)
{
    // Allocate memory to store objects in.
    // Use an array initially 10 elements long starting at 0 and growing by 10.
//  delete ObjectArray;
    ObjectArray = new TObjectArray(10,0,10);

    // If the file is not new then load it.

    if (GetDocPath()) {

        TInStream* is = InStream(ofRead); // Create Input Stream
        if (!is) return FALSE;

        const int buflen = 1000;
        char buf[buflen];
        memset(buf,0,buflen);
        is->read(buf,buflen);

        istrstream* iis = new istrstream(buf);
```

```
        if (!ReadFromStream(iis,ObjectArray))
            return FALSE;

        delete iis;
        delete is;   // Free memory used by stream
    }

    SetDirty(FALSE); // Initialize the document to clean since it hasn't been
                             // altered yet and doesn't need to be saved.
    return TRUE;
}


BOOL BondGraphDocument::Commit(BOOL force)
{
    // Commit is another name for save, so write document to file...

    //  but only if it needs to be written or you are forced to.
    if (!IsDirty() && !force)
        return TRUE;

    if (!TFileDocument::Commit(force)) // Notify parent class.
        return FALSE;

    if (isSubDocument) {
      // put objects back into compound object and close
      // must save children first
      int numObjects = ObjectArray->GetItemsInContainer();
      for (int i=numObjects-1;i>0;i--) {
         if ((*ObjectArray)[i]->typeId == BG_COMPOUND)
            if (((ObjectCompound*)(*ObjectArray)[i])->childOpen) {
              // if sub document is open then show error message telling
              // to close it and return false.
              return FALSE;
            }
      }

      // remove objects from document and place in object
      for (i=numObjects-1;i>0;i--) {
            parentObject->subobjects->Add((*ObjectArray)[i]);
            ObjectArray->Detach(i);
      }

      ((ObjectCompound*)(*ObjectArray)[i])->childOpen = FALSE;
      SetDirty(FALSE);
      Close();
    } else {
      TOutStream* os = OutStream(ofWrite); // Create the output stream
      if (!os)   return FALSE;

      const buflen = 1000;
      char  buf[buflen];
      ostrstream* oss = new ostrstream(buf,buflen);

      WriteToStream(oss,ObjectArray);

      *os << buf;

      delete oss;
      delete os; // Free memory used by stream
    }

    SetDirty(FALSE); // Set to clean since the document doesn't need to be
                             // saved since it just was.
    return TRUE;
}

BOOL BondGraphDocument::Revert(BOOL clear)
{
    // Revert is equivalent to closing without saving and loading the
    // same file. So that is just what we do.

    if (!TFileDocument::Revert(clear)) // Notify parent class.
        return FALSE;

    // I think, I need to free the memory used first, but this might be done
    // in TFileDocument::Revert.

    if (!clear)  // Clear tells us if we should just empty the file or load it.
        Open(0); // I think, I might have to call open anyway unless it is done
                 // by TFileDocument::Revert.

    return TRUE;
}

BOOL BondGraphDocument::Close()
```

```
{
    if (!TFileDocument::Close())  // Notify parent class.
        return FALSE;

    // I think, I need to free the memory used by each object before I
    // free the memory for the ObjectArray, but ObjectArray might do this.

    if (isSubDocument) {
      int numObjects = ObjectArray->GetItemsInContainer();
      for (int i=numObjects-1;i>=0;i--) {
         parentObject->subobjects->Add((*ObjectArray)[i]);
         ObjectArray->Detach(i);
      }
      ((ObjectCompound*)parentObject)->childOpen = FALSE;
      parentObject = 0;
    }
//  int numObjects = ObjectArray->GetItemsInContainer();
    // Delete each shape
//  for (int i = 0; i<=numObjects; i++)
//    delete (*ObjectArray)[i];

    ObjectArray->Flush();
    delete ObjectArray;       // Free memory used by the Array.
    ObjectArray = 0;          // Get rid of the pointer so no mistakes are made.

    delete ObjectIterator;
    ObjectIterator = 0;

    return TRUE;
}

// Data access functions provided for the various views which might use
// this document.

/*
const BasicObject* BondGraphDocument::GetObject(UINT index)
{
    if (!IsOpen() && !Open(0))
        return (NULL);

    // Use iterator to search through objects for specified object id.
    // In the future, I could use a traditional for loop since that
    // would be faster, but this way uses better abstraction.
    for (Start(); Current(); Next())
        if (Current()->id == index)
            return(Current());

    return (NULL);
}
*/

void BondGraphDocument::AddObject(BasicObject* object)
{
    if (object->typeId == BG_BOND)
          ObjectArray->AddAt(object,0);
    else ObjectArray->Add(object);
    if (isSubDocument==FALSE)
      SetDirty(TRUE);// The document needs to be saved since it was changed.
}

/*
BOOL BondGraphDocument::DeleteObject(UINT index)
{
    BasicObject* object;
    int numObjects = ObjectArray->GetItemsInContainer();

    // Search through objects for specified object id.
    for (int i = 0; i<=numObjects; i++)
        if ((*ObjectArray)[i]->id == index) {

            object = (*ObjectArray)[i];   // Remove the object from the array
            ObjectArray->Detach(i);
            delete object;                        // And free the memory

            return TRUE;
        }

    return FALSE;
}
*/

void BondGraphDocument::DeleteObjects(BOOL AllObjects)
{
    BasicObject* object;
    int numObjects = ObjectArray->GetItemsInContainer();

    for (int i=numObjects-1;i>=0;i--)
```

```
        if (AllObjects || (*ObjectArray)[i]->selected) {
            object = (*ObjectArray)[i];
            ObjectArray->Detach(i);
            delete object;
        }
}

void BondGraphDocument::DuplicateObjects(BOOL AllObjects)
{
    const buflen = 1000;
    char  buf[buflen]; // Should make this variable in future.
    memset(buf,0,buflen);

    ostrstream* os = new ostrstream(buf,buflen);
    if (os) {
        WriteToStream(os,ObjectArray,AllObjects);

        istrstream* is = new istrstream(buf,buflen);
        if (is)
            ReadFromStream(is,ObjectArray);
        delete is;
    }
    delete os;
}

// An iterator provided to loop through the objects in the document
void BondGraphDocument::Start()
{
    if (!IsOpen() && !Open(0))
        return;

    if (ObjectIterator)
        delete ObjectIterator;
    ObjectIterator = 0;

    if (ObjectArray)
      ObjectIterator = new TObjectArrayIterator(*ObjectArray);
}

/*const*/ BasicObject* BondGraphDocument::Current()
{
    if (ObjectIterator)
        if (*ObjectIterator)
          return (ObjectIterator->Current());

    return (NULL);
}

void BondGraphDocument::Next()
{
    if (ObjectIterator)
        (*ObjectIterator)++;
}

void BondGraphDocument::Reset()
{
    if (ObjectIterator)
        ObjectIterator->Restart();
    else
        Start();
}
```

bttlview.h

```
#if !defined(__bttlview_h)              // Sentry, use file only if it's not already included.
#define __bttlview_h

/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         bttlview.h
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Class definition for BondGraphView (TWindowView).
*/


#include <owl\owlpch.h>
#pragma hdrstop
```

```
#include "bttlapp.rh"            // Definition of all resources.
#include "bttldoc.h"
#include "bttltlst.h"

//{{TWindowView = BondGraphView}}
class BondGraphView : public TWindowView {
public:
     ToolList*          tl;
     BondGraphDocument* theDoc;
     float              theScale;
     char               tmptxt[50];
     TFont*                theMediumFont;
     int                xoffset, yoffset;

public:
     BondGraphView (BondGraphDocument& doc, TWindow* parent = 0);
     virtual ~BondGraphView ();

//{{BondGraphViewVIRTUAL_BEGIN}}
public:
     virtual void Paint (TDC& dc, bool erase, TRect& rect);
     virtual void SetupWindow ();
     virtual bool CanClose ();
//{{BondGraphViewVIRTUAL_END}}
//{{BondGraphViewRSP_TBL_BEGIN}}
protected:
     void EvGetMinMaxInfo (MINMAXINFO far& minmaxinfo);
     void EvLButtonDown (uint modKeys, TPoint& point);
     void EvMouseMove (uint modKeys, TPoint& point);
     void EvLButtonUp (uint modKeys, TPoint& point);
     void EvSize (uint sizeType, TSize& size);
     void EvHScroll (uint scrollCode, uint thumbPos, HWND hWndCtl);
     void EvVScroll (uint scrollCode, uint thumbPos, HWND hWndCtl);
     void EvRButtonDown (uint modKeys, TPoint& point);
     void CmEditCopy ();
     void CmEditCopyEn (TCommandEnabler &tce);
     void CmEditCut ();
     void CmEditCutEn (TCommandEnabler &tce);
     void CmEditDelete ();
     void CmEditDeleteEn (TCommandEnabler &tce);
     void CmEditPaste ();
     void CmEditPasteEn (TCommandEnabler &tce);
     bool EvSetCursor (HWND hWndCursor, uint hitTest, uint mouseMsg);
     void CmEditDuplicate ();
     void CmEditDuplicateEn (TCommandEnabler &tce);
     void EvChar (uint key, uint repeatCount, uint flags);
    void EvLButtonDblClk (uint modKeys, TPoint& point);
//{{BondGraphViewRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(BondGraphView);
};    //{{BondGraphView}}


#endif                                        // __bttlview_h sentry.
```

bttlview.cpp

```
/*  Project bottle
     Massachusetts Institute of Technology
     Copyright © 1995. All Rights Reserved.

     SUBSYSTEM:    bottle.exe Application
     FILE:         bttlview.cpp
     AUTHOR:       Richard K. Branton


     OVERVIEW
     ========
     Source file for implementation of BondGraphView (TWindowView).
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include "bttlapp.h"
#include "bttlview.h"
#include <stdio.h>
#include <windows.h>
#include <cstring.h>
#include <strstream.h>

//{{BondGraphView Implementation}}

//
// Build a response table for all messages/commands handled
// by BondGraphView derived from TWindowView.
```

```
//
DEFINE_RESPONSE_TABLE1(BondGraphView, TWindowView)
//{{BondGraphViewRSP_TBL_BEGIN}}
      EV_WM_GETMINMAXINFO,
      EV_WM_LBUTTONDOWN,
      EV_WM_MOUSEMOVE,
      EV_WM_LBUTTONUP,
      EV_WM_SIZE,
      EV_WM_HSCROLL,
      EV_WM_VSCROLL,
      EV_WM_RBUTTONDOWN,
      EV_COMMAND(CM_EDITCOPY, CmEditCopy),
      EV_COMMAND_ENABLE(CM_EDITCOPY, CmEditCopyEn),
      EV_COMMAND(CM_EDITCUT, CmEditCut),
      EV_COMMAND_ENABLE(CM_EDITCUT, CmEditCutEn),
      EV_COMMAND(CM_EDITDELETE, CmEditDelete),
      EV_COMMAND_ENABLE(CM_EDITDELETE, CmEditDeleteEn),
      EV_COMMAND(CM_EDITPASTE, CmEditPaste),
      EV_COMMAND_ENABLE(CM_EDITPASTE, CmEditPasteEn),
      EV_WM_SETCURSOR,
    EV_COMMAND(CM_EDITDUPLICATE, CmEditDuplicate),
    EV_COMMAND_ENABLE(CM_EDITDUPLICATE, CmEditDuplicateEn),
    EV_WM_CHAR,
    EV_WM_LBUTTONDBLCLK,
//{{BondGraphViewRSP_TBL_END}}
END_RESPONSE_TABLE;

#define PAGEWIDTH  19
#define PAGEHEIGHT 25
#define GRIDWIDTH  40
#define GRIDHEIGHT 40

////////////////////////////////////////////////////////
// BondGraphView
// ===========
// Construction/Destruction handling.
BondGraphView::BondGraphView (BondGraphDocument& doc, TWindow* parent)
      : TWindowView(doc, parent), theDoc(&doc)
{
      // INSERT>> Your constructor code here.
      theScale = 1.0;
      theMediumFont = 0;

      Attr.Style |= WS_VSCROLL | WS_HSCROLL;
      Scroller = new TScroller(this, 10, 10, PAGEWIDTH*40/10, PAGEHEIGHT*40/10);
      Scroller->AutoMode = FALSE;
      xoffset = 0;
      yoffset = 0;
}

BondGraphView::~BondGraphView ()
{
      // INSERT>> Your destructor code here.
      delete theMediumFont;
}

#include <math.h>

void SnapToGrid(POINT& pt, BondGraphDocument* doc)
{
  if (doc->gridSnap) {
      pt.x = floor(pt.x / 10.0) * 10;
      pt.y = floor(pt.y / 10.0) * 10;
  }
}

//
// Paint routine for Window, Printer, and PrintPreview for a TWindowView client.
//
void BondGraphView::Paint (TDC& dc, bool, TRect& rect)
{
      int x;
      BottleApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), BottleApp);
      if (theApp) {
            // Only paint if we're printing and we have something to paint, otherwise do nothing.
            if (theApp->Printing && theApp->Printer && !rect.IsEmpty()) {
                  // Use pageSize to get the size of the window to render into.  For a Window it's the
client area,
                  // for a printer it's the printer DC dimensions and for print preview it's the layout
window.
                  TSize   pageSize(rect.right - rect.left, rect.bottom - rect.top);

                  TPrintDialog::TData &printerData = theApp->Printer->GetSetup();

                  // Compute the number of pages to print.
                  printerData.MinPage = 1;
                  printerData.MaxPage = 1;
```

```
                // Draw Grid First.
                dc.SelectObject(TPen(TColor::Black,1,PS_DOT));
                dc.SelectObject(TBrush(TColor::Black));
                if (theDoc->gridShow) {
                  // Draw Grid
                  for (x=0;x<=PAGEWIDTH*40;x+=40) {
                     dc.MoveTo(x,0);  dc.LineTo(x,PAGEHEIGHT*40);
                  }

                  for (x=0;x<=PAGEHEIGHT*40;x+=40) {
                     dc.MoveTo(0,x);  dc.LineTo(PAGEWIDTH*40,x);
                  }
                }

                // Draw Page
                dc.MoveTo(0,0);
                dc.LineTo(PAGEWIDTH*40,0);
                dc.LineTo(PAGEWIDTH*40,PAGEHEIGHT*40);
                dc.LineTo(0,PAGEHEIGHT*40);
                dc.LineTo(0,0);

                // Draw Shadow
                dc.Rectangle(TRect(TPoint(PAGEWIDTH*40+1,8),TSize(8,PAGEHEIGHT*40)));
                dc.Rectangle(TRect(TPoint(8,PAGEHEIGHT*40+1),TSize(PAGEWIDTH*40,8)));

                dc.RestoreBrush();
                dc.RestorePen();

                // Special printing code
                theDoc->Start();
                BasicObject* theObj;
                while (theObj = theDoc->Current()) {
                    theObj->Paint(&dc, rect, theScale);
                    theDoc->Next();
                }

        } else {
                // Draw Grid First.
                if (theDoc->gridShow) {
                  dc.SelectObject(TPen(TColor::Black,1,PS_DOT));
                  for (x=0;x<=PAGEWIDTH*40;x+=40) {
                     dc.MoveTo(x,0);  dc.LineTo(x,PAGEHEIGHT*40);
                  }

                  for (x=0;x<=PAGEHEIGHT*40;x+=40) {
                     dc.MoveTo(0,x);  dc.LineTo(PAGEWIDTH*40,x);
                  }
                  dc.RestorePen();
                }

                // Loop over objects in document and paint to window
                theDoc->Start();
                BasicObject* theObj;
                while (theObj = theDoc->Current()) {
                    theObj->Draw(&dc, rect, theObj->selected, theScale);
                    theDoc->Next();
                }

        }
    }
}


void BondGraphView::EvGetMinMaxInfo (MINMAXINFO far& minmaxinfo)
{
     BottleApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), BottleApp);
     if (theApp) {
          if (theApp->Printing) {
                minmaxinfo.ptMaxSize = TPoint(32000, 32000);
                minmaxinfo.ptMaxTrackSize = TPoint(32000, 32000);
                return;
          }
     }
     TWindowView::EvGetMinMaxInfo(minmaxinfo);
}

void BondGraphView::EvLButtonDown (uint modKeys, TPoint& point)
{
     TWindowView::EvLButtonDown(modKeys, point);
     // INSERT>> Your code here.
     point.x += xoffset; point.y += yoffset;
     if (tl->CurrentTool())
        tl->CurrentTool()->LeftMouseDown(this,modKeys,point,*theDoc);
}

void BondGraphView::EvMouseMove (uint modKeys, TPoint& point)
```

```
{
     TWindowView::EvMouseMove(modKeys, point);
     // INSERT>> Your code here.
     point.x += xoffset; point.y += yoffset;
     if (tl->CurrentTool())
        tl->CurrentTool()->MouseMove(this,modKeys,point,*theDoc);
}

void BondGraphView::EvLButtonUp (uint modKeys, TPoint& point)
{
     TWindowView::EvLButtonUp(modKeys, point);
     // INSERT>> Your code here.
     point.x += xoffset; point.y += yoffset;
     if (tl->CurrentTool())
        tl->CurrentTool()->LeftMouseUp(this,modKeys,point,*theDoc);
}

void BondGraphView::SetupWindow ()
{
    TWindowView::SetupWindow();
    // INSERT>> Your code here.
}

void BondGraphView::EvSize (uint sizeType, TSize& size)
{
    TWindowView::EvSize(sizeType, size);

    // INSERT>> Your code here.
    if (sizeType == SIZENORMAL || sizeType == SIZEFULLSCREEN) {
    }
}

void BondGraphView::EvHScroll (uint scrollCode, uint thumbPos, HWND hWndCtl)
{
     TWindowView::EvHScroll(scrollCode, thumbPos, hWndCtl);
     xoffset = (int)Scroller->XPos * Scroller->XUnit;
}

void BondGraphView::EvVScroll (uint scrollCode, uint thumbPos, HWND hWndCtl)
{
     TWindowView::EvVScroll(scrollCode, thumbPos, hWndCtl);
     yoffset = (int)Scroller->YPos * Scroller->YUnit;
}


void BondGraphView::EvRButtonDown (uint modKeys, TPoint& point)
{
     TWindowView::EvRButtonDown(modKeys, point);
     // INSERT>> Your code here.
     point.x += xoffset; point.y += yoffset;
     if (tl->CurrentTool())
        tl->CurrentTool()->RightMouseDown(this,modKeys,point,*theDoc);
}


void BondGraphView::CmEditCopy ()
{
    // INSERT>> Your code here.
    const buflen = 1000;
    char  buf[buflen]; // Should make this variable in future.
    ostrstream /*TOutStream*/ * os = new ostrstream(buf,buflen);
    if (os) {
        /*theDoc->*/WriteToStream(os,FALSE);

        HGLOBAL hGlobalMemory = GlobalAlloc(GHND,(DWORD) strlen(buf)+1);
        void FAR* lpGlobalMemory = GlobalLock(hGlobalMemory);

        for (int n=0;n<strlen(buf);n++)
            *((char *) lpGlobalMemory + n) = buf[n];
        GlobalUnlock(hGlobalMemory);

        TClipboard& clipboard = OpenClipboard();
        clipboard.EmptyClipboard();
        clipboard.SetClipboardData(CF_TEXT,hGlobalMemory);
        clipboard.CloseClipboard();
     }
     delete os;
}

void BondGraphView::CmEditCopyEn (TCommandEnabler &tce)
{
     // INSERT>> Your code here.
     tce.Enable(theDoc->inSelected);
}

void BondGraphView::CmEditCut ()
{
```

```
     // INSERT>> Your code here.
/*
     const buflen = 1000;
     char  buf[buflen]; // Should make this variable in future.
     ostrstream* os = new ostrstream(buf,buflen);
     if (os) {
          theDoc->WriteToStream(os);

          HGLOBAL hGlobalMemory = GlobalAlloc(GHND,(DWORD) strlen(buf)+1);
          void FAR* lpGlobalMemory = GlobalLock(hGlobalMemory);

          for (int n=0;n<strlen(buf);n++)
               *((char *) lpGlobalMemory + n) = buf[n];
          GlobalUnlock(hGlobalMemory);

          TClipboard& clipboard = OpenClipboard();
          clipboard.EmptyClipboard();
          clipboard.SetClipboardData(CF_TEXT,hGlobalMemory);
          clipboard.CloseClipboard();
     }
     delete os;
*/
     CmEditCopy();
     // Loop Down Deleting Selected
     CmEditDelete();
}

void BondGraphView::CmEditCutEn (TCommandEnabler &tce)
{
     // INSERT>> Your code here.
     tce.Enable(theDoc->inSelected);
}

void BondGraphView::CmEditDelete ()
{
     // INSERT>> Your code here.
     // Loop Down Deleting Selected
     theDoc->DeleteObjects(FALSE);
     Invalidate();
}

void BondGraphView::CmEditDeleteEn (TCommandEnabler &tce)
{
     // INSERT>> Your code here.
     tce.Enable(theDoc->inSelected);
}

void BondGraphView::CmEditPaste ()
{
     // INSERT>> Your code here.
     // Unselect All
     BasicObject* objptr;
     theDoc->Start();
     while (objptr = theDoc->Current()) {
          objptr->selected = FALSE;
          theDoc->Next();
     }
     int numObjects = theDoc->ObjectArray->GetItemsInContainer();

     // Paste
     TClipboard& clipboard = OpenClipboard();
     HGLOBAL hClipMemory = clipboard.GetClipboardData(CF_TEXT);

     const buflen = 1000;
     char buf[buflen];
     memset(buf,0,buflen);

     void FAR* lpClipMemory = GlobalLock(hClipMemory);

     for (int n=0;n<strlen((char *)lpClipMemory);n++)
          buf[n] = *((char *) lpClipMemory + n);
     buf[n] = 0;

     GlobalUnlock(hClipMemory);
     clipboard.CloseClipboard();

     istrstream* is = new istrstream(buf,buflen);

     if (is)
          /*theDoc->*/ReadFromStream(is,theDoc->ObjectArray);
     delete is;

     // Select Pasted
     int maxObjects = theDoc->ObjectArray->GetItemsInContainer();
     for (int i=numObjects;i<maxObjects;i++) {
          (*theDoc->ObjectArray)[i]->selected = TRUE;
     }
```

```
    Invalidate();
}

void BondGraphView::CmEditPasteEn (TCommandEnabler &tce)
{
     // INSERT>> Your code here.
     TClipboard& clipboard = OpenClipboard();
     tce.Enable(clipboard.IsClipboardFormatAvailable(CF_TEXT)); //theDoc->inClipboard);
     clipboard.CloseClipboard();
}

bool BondGraphView::CanClose ()
{
     bool result;
     result = TWindowView::CanClose();
     // INSERT>> Your code here.
     return result;
}

bool BondGraphView::EvSetCursor (HWND hWndCursor, uint hitTest, uint mouseMsg)
{
     bool result = TRUE;
     //result = TWindowView::EvSetCursor(hWndCursor, hitTest, mouseMsg);
     // INSERT>> Your code here.
     if (!(tl->CurrentTool() &&   tl->CurrentTool()->SetCursor(this,hitTest)))
        TWindowView::EvSetCursor(hWndCursor, hitTest, mouseMsg);
     return result;
}


void BondGraphView::CmEditDuplicate ()
{
     // INSERT>> Your code here.
    theDoc->DuplicateObjects(FALSE);
    Invalidate();
}


void BondGraphView::CmEditDuplicateEn (TCommandEnabler &tce)
{
     // INSERT>> Your code here.
     tce.Enable(theDoc->inSelected);
}


void BondGraphView::EvChar (uint key, uint repeatCount, uint flags)
{
     TWindowView::EvChar(key, repeatCount, flags);

     // INSERT>> Your code here.
     if (tl->CurrentTool())
        tl->CurrentTool()->CharTyped(this,key,repeatCount,flags,*theDoc);

}


void BondGraphView::EvLButtonDblClk (uint modKeys, TPoint& point)
{
    TWindowView::EvLButtonDblClk(modKeys, point);

    // INSERT>> Your code here.
     point.x += xoffset; point.y += yoffset;
     if (tl->CurrentTool())
        tl->CurrentTool()->RightMouseDown(this,modKeys,point,*theDoc);
}
```

apxprev.h

```
#if !defined(__apxprev_h)              // Sentry, use file only if it's not already included.
#define __apxprev_h

/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         APXPrev.H
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Class definition for PreviewWindow (Print Preview).
*/
```

```
#include <owl\owlpch.h>
#pragma hdrstop

#include "apxprint.h"
#include "bttlapp.rh"


//{{TDecoratedFrame = PreviewWindow}}
class PreviewWindow : public TDecoratedFrame {
public:
    PreviewWindow (TWindow *parentWindow, TPrinter *printer, TWindow* currWindow, const char far*
title, TLayoutWindow* client);
    ~PreviewWindow ();

    int             PageNumber;

    TWindow         *CurrWindow;
    TControlBar     *PreviewSpeedBar;
    TPreviewPage    *Page1;
    TPreviewPage    *Page2;
    TPrinter        *Printer;

    TPrintDC        *PrnDC;
    TSize           *PrintExtent;
    APXPrintOut     *Printout;

private:
    TLayoutWindow   *Client;

    void SpeedBarState ();
    void PPR_PreviousEnable (TCommandEnabler &tce);
    void PPR_NextEnable (TCommandEnabler &tce);
    void PPR_Previous ();
    void PPR_Next ();
    void PPR_OneUp ();
    void PPR_TwoUpEnable (TCommandEnabler &tce);
    void PPR_TwoUp ();
    void PPR_Done ();
    void CmPrintEnable (TCommandEnabler &tce);
    void CmPrint ();

//{{PreviewWindowVIRTUAL_BEGIN}}
protected:
    virtual void SetupWindow ();
//{{PreviewWindowVIRTUAL_END}}

//{{PreviewWindowRSP_TBL_BEGIN}}
protected:
//{{PreviewWindowRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(PreviewWindow);
};    //{{PreviewWindow}}


#endif      // __apxprev_h sentry.
```

apxprev.cpp

```
/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         APXPrev.CPP
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Source file for implementation of Print Preview.
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include "apxprev.h"

#include "bttlapp.rh"


//{{PreviewWindow Implementation}}

DEFINE_RESPONSE_TABLE1(PreviewWindow, TDecoratedFrame)
    EV_COMMAND_ENABLE(APX_PPR_PREVIOUS, PPR_PreviousEnable),
    EV_COMMAND_ENABLE(APX_PPR_NEXT, PPR_NextEnable),
```

```
    EV_COMMAND(APX_PPR_PREVIOUS, PPR_Previous),
    EV_COMMAND(APX_PPR_NEXT, PPR_Next),
    EV_COMMAND(APX_PPR_ONEUP, PPR_OneUp),
    EV_COMMAND_ENABLE(APX_PPR_TWOUP, PPR_TwoUpEnable),
    EV_COMMAND(APX_PPR_TWOUP, PPR_TwoUp),
    EV_COMMAND(APX_PPR_DONE, PPR_Done),
    EV_COMMAND(CM_FILEPRINT, CmPrint),
    EV_COMMAND_ENABLE(CM_FILEPRINT, CmPrintEnable),
//{{PreviewWindowRSP_TBL_BEGIN}}
//{{PreviewWindowRSP_TBL_END}}
END_RESPONSE_TABLE;


PreviewWindow::PreviewWindow (TWindow *parentWindow, TPrinter *printer, TWindow* currWindow, const
char far* title, TLayoutWindow* client) :
                TDecoratedFrame(parentWindow, title, client)
{
    CurrWindow = currWindow;
    Printer = printer;
    Client = client;
    Page1 = 0;
    Page2 = 0;

    TPrintDialog::TData& data = Printer->GetSetup();
    PrnDC = new TPrintDC(data.GetDriverName(),
                         data.GetDeviceName(),
                         data.GetOutputName(),
                         data.GetDevMode());

    PrintExtent = new TSize(PrnDC->GetDeviceCaps(HORZRES), PrnDC->GetDeviceCaps(VERTRES));
    Printout = new APXPrintOut(Printer, "Print Preview", currWindow, true);

    SetBkgndColor(::GetSysColor(COLOR_APPWORKSPACE));

    //
    // Create default toolbar New and associate toolbar buttons with commands.
    //
    PreviewSpeedBar = new TControlBar(this);
    PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_PREVIOUS, APX_PPR_PREVIOUS,
TButtonGadget::Command, true));
    PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_NEXT, APX_PPR_NEXT, TButtonGadget::Command,
true));
    PreviewSpeedBar->Insert(*new TSeparatorGadget(6));
    PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_ONEUP, APX_PPR_ONEUP, TButtonGadget::Exclusive,
true, TButtonGadget::Down));
    PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_TWOUP, APX_PPR_TWOUP, TButtonGadget::Exclusive,
true));
    PreviewSpeedBar->Insert(*new TSeparatorGadget(12));
    PreviewSpeedBar->Insert(*new TTextGadget(APX_PPR_CURRPAGE, TGadget::Recessed, TTextGadget::Left,
10, "Page 1"));
    PreviewSpeedBar->Insert(*new TSeparatorGadget(20));
    PreviewSpeedBar->Insert(*new TButtonGadget(CM_FILEPRINT, CM_FILEPRINT, TButtonGadget::Command,
true));
    PreviewSpeedBar->Insert(*new TSeparatorGadget(20));
    PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_DONE, APX_PPR_DONE, TButtonGadget::Command,
true));
    Insert(*PreviewSpeedBar, TDecoratedFrame::Top);

    // We want a window that cannot be sized, maximized, or minimized.
    Attr.Style = (WS_VISIBLE | WS_POPUP);

    // Don't show the border of the preview window.
    Attr.X = 0;
    Attr.Y = -1;
    Attr.W = Parent->GetClientRect().Width();
    Attr.H = Parent->GetClientRect().Height() + 1;
    parentWindow->MapWindowPoints(HWindow, (TPoint *)&(Attr.X), 1);
}


PreviewWindow::~PreviewWindow ()
{
    delete Page1;
    Page1 = 0;
    delete Page2;
    Page2 = 0;

    delete PrnDC;
    PrnDC = 0;
    delete PrintExtent;
    PrintExtent = 0;
    delete Printout;
    Printout = 0;
}


void PreviewWindow::SetupWindow ()
```

```
{
    TDecoratedFrame::SetupWindow();

    TPrintDialog::TData& data = Printer->GetSetup();
    Page1 = new TPreviewPage(Client, *Printout, *PrnDC, *PrintExtent, 1);
    Page1->SetPageNumber(1);
    data.FromPage = 1;
    data.ToPage = 1;
    data.MinPage = 1;
    data.MaxPage = 1;

    Page2 = 0;

    TLayoutMetrics metrics1;

    metrics1.X.Set(lmLeft, lmRightOf, lmParent, lmLeft, 15);
    metrics1.Y.Set(lmTop, lmBelow, lmParent, lmTop, 15);

    //
    // Determine major axis of preview page, have that follow parent size.
    // Make minor axis a percentage (aspect ratio) of the page's major axis
    //
    TRect r = Client->GetClientRect();
    long ratio;

    if (PrintExtent->cx > PrintExtent->cy)
        ratio = ((long)PrintExtent->cy * 100) / PrintExtent->cx;
    else
        ratio = ((long)PrintExtent->cx * 100) / PrintExtent->cy;

    bool xMajor = (((r.Width() * ratio) / 100) > r.Height());
    if (xMajor){
        metrics1.Height.Set(lmBottom, lmAbove, lmParent, lmBottom, 15);
        metrics1.Width.PercentOf(Page1, (int)((long)PrintExtent->cx * 95 / PrintExtent->cy),
lmHeight);
    } else {
        metrics1.Height.PercentOf(Page1, (int)((long)PrintExtent->cy * 95 / PrintExtent->cx),
lmWidth);
        metrics1.Width.Set(lmRight, lmLeftOf, lmParent, lmRight, 15);
    }

    Page1->Create();

    Client->SetChildLayoutMetrics(*Page1, metrics1);
    Client->Layout();
}


void PreviewWindow::SpeedBarState ()
{
    TPrintDialog::TData &printerData = Printer->GetSetup();

    // Update the page count.
    TTextGadget *theTGadget = TYPESAFE_DOWNCAST(PreviewSpeedBar->GadgetWithId(APX_PPR_CURRPAGE),
TTextGadget);
    if (theTGadget) {
        char    buffer[32];
        if (Page2 && (printerData.FromPage != printerData.ToPage))
            wsprintf(buffer, "Page %d - %d", printerData.FromPage, printerData.ToPage);
        else
            wsprintf(buffer, "Page %d", printerData.FromPage);
        theTGadget->SetText(buffer);
    }
}


void PreviewWindow::PPR_PreviousEnable (TCommandEnabler &tce)
{
    // Only have previous on if we're not at the first page.
    TPrintDialog::TData &printerData = Printer->GetSetup();
    tce.Enable(printerData.FromPage != 1);
}


void PreviewWindow::PPR_NextEnable (TCommandEnabler &tce)
{
    // Only have next on if we're not at the last page.
    TPrintDialog::TData &printerData = Printer->GetSetup();
    tce.Enable(printerData.ToPage != printerData.MaxPage);
}


void PreviewWindow::PPR_Previous ()
{
    TPrintDialog::TData &printerData = Printer->GetSetup();

    if (printerData.FromPage > printerData.MinPage) {
```

```
        printerData.FromPage--;
        printerData.ToPage--;

        Page1->SetPageNumber(printerData.FromPage);
        if (Page2)
            Page2->SetPageNumber(printerData.ToPage);
    }

    SpeedBarState();
}


void PreviewWindow::PPR_Next ()
{
    TPrintDialog::TData &printerData = Printer->GetSetup();

    if (printerData.ToPage < printerData.MaxPage) {
        printerData.FromPage++;
        printerData.ToPage++;

        Page1->SetPageNumber(printerData.FromPage);
        if (Page2)
            Page2->SetPageNumber(printerData.ToPage);
    }

    SpeedBarState();
}


void PreviewWindow::PPR_OneUp ()
{
    if (Page2) {
        Client->RemoveChildLayoutMetrics(*Page2);

        delete Page2;
        Page2 = 0;

        Client->Layout();

        TPrintDialog::TData &printerData = Printer->GetSetup();
        printerData.ToPage = printerData.FromPage;

        SpeedBarState();
    }
}


void PreviewWindow::PPR_TwoUpEnable (TCommandEnabler &tce)
{
    // Two up is only available for portrait mode.
    tce.Enable(PrintExtent->cx <= PrintExtent->cy);
}


void PreviewWindow::PPR_TwoUp ()
{
    if (Page2 == 0) {
        Page2 = new TPreviewPage(Client, *Printout, *PrnDC, *PrintExtent, PageNumber + 1);
        Page2->Create();

        TLayoutMetrics metrics2;

        metrics2.X.Set(lmLeft, lmRightOf, Page1, lmRight, 30);
        metrics2.Y.SameAs(Page1, lmTop);

        // Assume portrait
        //
        metrics2.Width.SameAs(Page1, lmWidth);
        metrics2.Height.SameAs(Page1, lmBottom);

        Client->SetChildLayoutMetrics(*Page2, metrics2);
        Client->Layout();

        TPrintDialog::TData &printerData = Printer->GetSetup();

        // Page 2 is the next page.  If the next page is outside of our
        // range then set the first page back one and the 2nd page is
        // the current page.  If the document is only 1 page long then
        // the 2nd page is empty.
        if (printerData.FromPage == printerData.MaxPage) {
            if (printerData.FromPage > 1) {
                printerData.FromPage--;
                printerData.ToPage = printerData.FromPage + 1;
                Page1->SetPageNumber(printerData.FromPage);
                Page2->SetPageNumber(printerData.ToPage);
            } else
                Page2->SetPageNumber(0);
```

```
        } else {
            printerData.ToPage = printerData.FromPage + 1;
            Page2->SetPageNumber(printerData.ToPage);
        }

        SpeedBarState();
    }
}


void PreviewWindow::PPR_Done ()
{
    // Don't call the base class EvClose we do not want PreviewWindow to be destructed.
    GetApplication()->EndModal(IDCANCEL);
}


void PreviewWindow::CmPrint ()
{
    TWindow *client = GetApplication()->GetMainWindow()->GetClientWindow();

    if (client)
        client->SendMessage(WM_COMMAND, CM_FILEPRINT, 0);
}


void PreviewWindow::CmPrintEnable (TCommandEnabler &tce)
{
    tce.Enable(true);
}
```

apxprint.h

```
#if !defined(__apxprint_h)              // Sentry use file only if it's not already included.
#define __apxprint_h

/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         APXPrint.H
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Class definition for APXPrintOut (TPrintOut).
*/


#include <owl\owlpch.h>
#pragma hdrstop


class APXPrintOut : public TPrintout {
public:
    APXPrintOut (TPrinter *printer, const char far *title, TWindow* window, bool scale = true) :
TPrintout(title)
      { Printer = printer; Window = window; Scale = scale; MapMode = MM_ANISOTROPIC; }

    void GetDialogInfo (int& minPage, int& maxPage, int& selFromPage, int& selToPage);
    void BeginPrinting ();
    void BeginPage (TRect &clientR);
    void PrintPage (int page, TRect& rect, unsigned flags);
    void EndPage ();
    void SetBanding (bool b)        { Banding = b; }
    bool HasPage (int pageNumber);

protected:
    TWindow     *Window;
    bool        Scale;
    TPrinter    *Printer;
    int         MapMode;

    int         PrevMode;
    TSize       OldVExt, OldWExt;
    TRect       OrgR;
};


#endif          // __apxprint_h sentry.
```

apxprint.cpp

```
/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         APXPrint.CPP
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Source file for implementation of Printing.
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include "apxprint.h"


// Do not enable page range in the print dialog since only one page is
// available to be printed
//
void APXPrintOut::GetDialogInfo (int& minPage, int& maxPage, int& selFromPage, int& selToPage)
{
    minPage = maxPage = 0;
    selFromPage = selToPage = 0;
}


void APXPrintOut::BeginPrinting ()
{
    TRect clientR;

    BeginPage(clientR);

    HFONT   hFont = (HFONT)Window->GetWindowFont();
    TFont   font("Arial", -12);
    if (hFont == 0)
      DC->SelectObject(font);
    else
      DC->SelectObject(TFont(hFont));

    TEXTMETRIC  tm;
    int fHeight = (DC->GetTextMetrics(tm) == true) ? tm.tmHeight + tm.tmExternalLeading : 10;

    DC->RestoreFont();

    // How many lines of this font can we fit on a page.
    int linesPerPage = MulDiv(clientR.Height(), 1, fHeight);

    TPrintDialog::TData &printerData = Printer->GetSetup();

    // GetMinMaxInfo event is overrided to return the number of lines when printing.
    MINMAXINFO minmaxinfo;
    Window->SendMessage(WM_GETMINMAXINFO, 0, (long)&minmaxinfo);
    int maxPg = ((minmaxinfo.ptMaxSize.y / linesPerPage) + 1.0);


    // Compute the number of pages to print.
    printerData.MinPage = 1;
    printerData.MaxPage = maxPg;

    EndPage();

    TPrintout::BeginPrinting();
}


void APXPrintOut::BeginPage (TRect &clientR)
{
    TScreenDC screenDC;
    TSize screenRes(screenDC.GetDeviceCaps(LOGPIXELSX),
                    screenDC.GetDeviceCaps(LOGPIXELSY));
    TSize printRes(DC->GetDeviceCaps(LOGPIXELSX),
                   DC->GetDeviceCaps(LOGPIXELSY));

    // Temporarily change the window size (so any WM_PAINT queries on the total window size
(GetClientRect) is
    // the window size for the WM_PAINT of the window and the printer page size when Paint is called
from
    // PrintPage. Notice, we don't use AdjustWindowRect because its harder and not accurate.  Instead
we
    // compute the difference (in pixels) between the client window and the frame window.  This
difference
    // is then added to the clientRect to compute the new frame window size for SetWindowPos.
```

```
    clientR = Window->GetClientRect();
    Window->MapWindowPoints(HWND_DESKTOP, (TPoint*)&clientR, 2);

    // Compute extra X and Y pixels to bring a client window dimensions to equal the frame window.
    OrgR = Window->GetWindowRect();
    int adjX = OrgR.Width() - clientR.Width();
    int adjY = OrgR.Height() - clientR.Height();

    // Conditionally scale the DC to the window so the printout will resemble the window.
    if (Scale) {
        clientR = Window->GetClientRect();
        PrevMode = DC->SetMapMode(MapMode);
        DC->SetViewportExt(PageSize, &OldVExt);

        // Scale window to logical page size (assumes left & top are 0)
        clientR.right = MulDiv(PageSize.cx, screenRes.cx, printRes.cx);
        clientR.bottom = MulDiv(PageSize.cy, screenRes.cy, printRes.cy);

        DC->SetWindowExt(clientR.Size(), &OldWExt);
    }

    // Compute the new size of the window based on the printer DC dimensions.
    // Resize the window, notice position, order, and redraw are not done the window size changes but
the user
    // doesn't see any visible change to the window.
    Window->SetRedraw(false);
    Window->SetWindowPos(0, 0, 0, clientR.Width() + adjX, clientR.Height() + adjY,
                         SWP_NOMOVE | SWP_NOREDRAW | SWP_NOZORDER| SWP_NOACTIVATE);
}


void APXPrintOut::PrintPage (int page, TRect& bandRect, unsigned)
{
    TRect clientR;

    BeginPage(clientR);

    if (Scale)
        DC->DPtoLP(bandRect, 2);

    // Change the printer range to this current page.
    TPrintDialog::TData& printerData = Printer->GetSetup();
    int fromPg = printerData.FromPage;
    int toPg = printerData.ToPage;

    printerData.FromPage = page;
    printerData.ToPage = page;

    // Call the window to paint itself to the printer DC.
    Window->Paint(*DC, false, bandRect);

    printerData.FromPage = fromPg;
    printerData.ToPage = toPg;

    if (Scale)
        DC->LPtoDP(bandRect, 2);

    EndPage();
}


void APXPrintOut::EndPage ()
{
    // Resize to original window size, no one's the wiser.
    Window->SetWindowPos(0, 0, 0, OrgR.Width(), OrgR.Height(),
                         SWP_NOMOVE | SWP_NOREDRAW | SWP_NOZORDER| SWP_NOACTIVATE);
    Window->SetRedraw(true);

    // Restore changes made to the DC
    if (Scale) {
        DC->SetWindowExt(OldWExt);
        DC->SetViewportExt(OldVExt);
        DC->SetMapMode(PrevMode);
    }
}


bool APXPrintOut::HasPage (int pageNumber)
{
    TPrintDialog::TData &printerData = Printer->GetSetup();

    return (pageNumber >= printerData.MinPage) && (pageNumber <= printerData.MaxPage);
}
```

bttlapp.h

```
#if !defined(__bttlapp_h)              // Sentry, use file only if it's not already included.
#define __bttlapp_h

/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         bttlapp.h
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Class definition for BottleApp (TApplication).
*/


//#include <owl\owlpch.h>
//#pragma hdrstop

#include <classlib\bags.h>

#include "bttlmdi.h"

#include "bttlapp.rh"            // Definition of all resources.
#include "bttltlst.h"
#include "bttlpal.h"

// TFileDrop class Maintains information about a dropped file, its name, where it was dropped,
// and whether or not it was in the client area
class TFileDrop {
public:
    operator == (const TFileDrop& other) const {return this == &other;}

    char*   FileName;
    TPoint  Point;
    bool    InClientArea;

    TFileDrop (char*, TPoint&, bool, TModule*);
    ~TFileDrop ();

    const char* WhoAmI ();
private:
    //
    // hidden to prevent accidental copying or assignment
    //
    TFileDrop (const TFileDrop&);
     TFileDrop & operator = (const TFileDrop&);
};

typedef TIBagAsVector<TFileDrop> TFileList;
typedef TIBagAsVectorIterator<TFileDrop> TFileListIter;


//{{TApplication = BottleApp}}
class BottleApp : public TApplication {
public:
    TModule* ResModule;
     ToolList tl;
     void CmToolDispatch(UINT);

     void CmTogglePalette(UINT);
     void CeTogglePalette(TCommandEnabler&);
    void CmCreateCompound();

     int PaletteOpen[5];
     BottlePalette* Palette[5];

private:
     void SetupSpeedBar (TDecoratedMDIFrame *frame);
     void AddFiles (TFileList* files);
    void CmExtractCompound();
     void CmSetInputObject();
    void CmSetOutputObject();

public:
     BottleApp ();
     virtual ~BottleApp ();

     void CreateGadgets (TControlBar *cb, bool server = false);

     BottleMDIClient  *mdiClient;
```

```
     // Public data members used by the print menu commands and Paint routine in MDIChild.
     TPrinter        *Printer;              // Printer support.
     int             Printing;              // Printing in progress.

//{{BottleAppVIRTUAL_BEGIN}}
public:
     virtual void InitMainWindow ();
     virtual void InitInstance (); //{{BottleAppVIRTUAL_END}}

//{{BottleAppRSP_TBL_BEGIN}}
protected:
     void EvNewView (TView& view);
     void EvCloseView (TView& view);
     void CmHelpAbout ();
     void EvDropFiles (TDropInfo drop);
     void EvWinIniChange (char far* section);
//{{BottleAppRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(BottleApp);
};    //{{BottleApp}}


#endif                                        // __bttlapp_h sentry.
```

bttlapp.cpp

```
/*  Project bottle
     Massachusetts Institute of Technology
     Copyright © 1995. All Rights Reserved.

     SUBSYSTEM:    bottle.exe Application
     FILE:         bttlapp.cpp
     AUTHOR:       Richard K. Branton


     OVERVIEW
     ========
     Source file for implementation of BottleApp (TApplication).
*/

#include <owl\owlpch.h>
#include <owl\vbxctl.h>
#include <owl\applicat.h>
#include <owl\dialog.h>
#include <owl\framewin.h>

#include "bttldoc.h"
#include "bttltool.h"
#include "toolelem.h"
#include "toolslct.h"
#include "toolshap.h"
#include "toolcomp.h"

#include <dir.h>

#include "bttlapp.h"
#include "bttlmdi.h"
#include "bttlmdic.h"
#include "bttlview.h"
#include "bttlabtd.h"                          // Definition of about dialog.
#include "palettes.h"

//#define USE_RESOURCE_DLL

// Drag / Drop support:
TFileDrop::TFileDrop (char* fileName, TPoint& p, bool inClient, TModule*)
{
     char    exePath[MAXPATH];

     exePath[0] = 0;
     FileName = strcpy(new char[strlen(fileName) + 1], fileName);
     Point = p;
     InClientArea = inClient;
}

TFileDrop::~TFileDrop ()
{
     delete FileName;
}

const char *TFileDrop::WhoAmI ()
{
  return FileName;
}

//{{BottleApp Implementation}}
```

```
//{{DOC_VIEW}}
DEFINE_DOC_TEMPLATE_CLASS(BondGraphDocument, BondGraphView, BondGraphTemplate);
//{{DOC_VIEW_END}}

//{{DOC_MANAGER}}
BondGraphTemplate __dvt1("Bond Graphs (*.BG)", "*.BG", 0, "BG", dtAutoDelete | dtUpdateDir);
//{{DOC_MANAGER_END}}

//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(BottleApp, TApplication)
//{{BottleAppRSP_TBL_BEGIN}}
     EV_OWLVIEW(dnCreate, EvNewView),
     EV_OWLVIEW(dnClose,  EvCloseView),
     EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
     EV_WM_DROPFILES,
     EV_WM_WININICHANGE,

     // Object Tools
     EV_COMMAND_AND_ID(CM_ARROW, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_0JUNCTION, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_1JUNCTION, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_ESOURCE, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_FSOURCE, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_CAPACITANCE, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_INDUCTANCE, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_RESISTOR, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_GYRATOR, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_TRANSFORM, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_BOND, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_BONDCAUS, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_BONDDIR, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_SBOND, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_LINE, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_SQUARE, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_CIRCLE, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_TEXT, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_BITMAP, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_COMPOUND_EMPTY, CmToolDispatch),
     EV_COMMAND_AND_ID(CM_COMPOUND, CmToolDispatch),
     EV_COMMAND(CM_COMPOUND_TOGETHER, CmCreateCompound),
     EV_COMMAND(CM_COMPOUND_APART, CmExtractCompound),
     EV_COMMAND(CM_INPUT, CmSetInputObject),
     EV_COMMAND(CM_OUTPUT, CmSetOutputObject),

     // Palette Windows
     EV_COMMAND_AND_ID(CM_WINDOWBASIC_ELEMENTS_PALETTE, CmTogglePalette),
     EV_COMMAND_AND_ID(CM_WINDOWSHAPE_PALETTE, CmTogglePalette),
     EV_COMMAND_AND_ID(CM_WINDOWATTRIBUTES_PALETTE, CmTogglePalette),
     EV_COMMAND_AND_ID(CM_WINDOWCOMPOUNDCOMPLEX_PALETTE, CmTogglePalette),

     EV_COMMAND_ENABLE(CM_WINDOWBASIC_ELEMENTS_PALETTE, CeTogglePalette),
     EV_COMMAND_ENABLE(CM_WINDOWSHAPE_PALETTE, CeTogglePalette),
     EV_COMMAND_ENABLE(CM_WINDOWATTRIBUTES_PALETTE, CeTogglePalette),
     EV_COMMAND_ENABLE(CM_WINDOWCOMPOUNDCOMPLEX_PALETTE, CeTogglePalette),

//{{BottleAppRSP_TBL_END}}
END_RESPONSE_TABLE;

void BottleApp::CmToolDispatch(UINT id)
{
    if (tl.CurrentTool())
        tl.CurrentTool()->DeSelected();

    tl.CurrentTool(id);

    if (tl.CurrentTool())
        tl.CurrentTool()->Selected(this);
}

void BottleApp::CmTogglePalette(UINT id)
{
  int index;
  switch (id) {
        case CM_WINDOWBASIC_ELEMENTS_PALETTE: index = 0; break;
        case CM_WINDOWSHAPE_PALETTE:          index = 1; break;
        case CM_WINDOWATTRIBUTES_PALETTE:    index = 2; break;
        case CM_WINDOWCOMPOUNDCOMPLEX_PALETTE:index = 3; break;
  };

  if (PaletteOpen[index]) {
     Palette[index]->HideWindow();
     tl.CurrentTool(0);
  } else {
```

```
      Palette[index]->ShowWindow();
      tl.CurrentTool(CM_ARROW);
      tl.CurrentTool()->Selected(this);
  }
}

void BottleApp::CmSetInputObject()
{
  tl.CurrentTool(CM_ARROW);
  if (((SelectionTool*)tl.CurrentTool())->docptr) {
      BondGraphDocument* thedoc = ((SelectionTool*)tl.CurrentTool())->docptr;
      // Set First Selected Object To Input
      int found = FALSE;
      int numObjects = thedoc->ObjectArray->GetItemsInContainer();
      for (int j=0;j<numObjects;j++)
         if (found)
           (*thedoc->ObjectArray)[j]->ioInput = FALSE;
         else if ((*thedoc->ObjectArray)[j]->selected) {
              (*thedoc->ObjectArray)[j]->ioInput = TRUE;
              found = TRUE;
         } else (*thedoc->ObjectArray)[j]->ioInput = FALSE;
      ((SelectionTool*)tl.CurrentTool())->winptr->Invalidate();
  }
}

void BottleApp::CmSetOutputObject()
{
  tl.CurrentTool(CM_ARROW);
  if (((SelectionTool*)tl.CurrentTool())->docptr) {
      BondGraphDocument* thedoc = ((SelectionTool*)tl.CurrentTool())->docptr;
      // Set First Selected Object To Input
      int found = FALSE;
      int numObjects = thedoc->ObjectArray->GetItemsInContainer();
      for (int j=0;j<numObjects;j++)
         if (found)
           (*thedoc->ObjectArray)[j]->ioOutput = FALSE;
         else if ((*thedoc->ObjectArray)[j]->selected) {
              (*thedoc->ObjectArray)[j]->ioOutput = TRUE;
              found = TRUE;
         } else (*thedoc->ObjectArray)[j]->ioOutput = FALSE;
      ((SelectionTool*)tl.CurrentTool())->winptr->Invalidate();
  }
}

void BottleApp::CmCreateCompound()
{
  tl.CurrentTool(CM_ARROW);
  if (((SelectionTool*)tl.CurrentTool())->docptr) {
      // Add Object To Document
      BasicObject* objectPtr;
      objectPtr = new ObjectCompound(*((SelectionTool*)tl.CurrentTool())->docptr);
      ((SelectionTool*)tl.CurrentTool())->docptr->AddObject(objectPtr);
      ((SelectionTool*)tl.CurrentTool())->winptr->Invalidate();
  }
}

void BottleApp::CmExtractCompound()
{
  tl.CurrentTool(CM_ARROW);
  if (((SelectionTool*)tl.CurrentTool())->docptr) {

      int i = 0;
      BondGraphDocument* thedoc = ((SelectionTool*)tl.CurrentTool())->docptr;
      int numObjects = thedoc->ObjectArray->GetItemsInContainer();
      for (int j=numObjects-1;j>=0;j--)
         if ((*thedoc->ObjectArray)[j]->selected) {
           i++;
           break;
         }

      if (i==0) return;

      MessageBeep(-1);

      for (j=numObjects-1;j>=0;j--)
         if ((*thedoc->ObjectArray)[j]->selected)
           if ((*thedoc->ObjectArray)[j]->typeId == BG_COMPOUND)
              if (((ObjectCompound *)(*thedoc->ObjectArray)[j])->ExtractToArray(thedoc->ObjectArray)) {
                 // remove object after extracting
                 BasicObject* tmpObject = (* thedoc->ObjectArray)[j];
                 thedoc->ObjectArray->Detach(j);
                 delete tmpObject;
              }
      ((SelectionTool*)tl.CurrentTool())->winptr->Invalidate();
  }
}
```

```
void BottleApp::CeTogglePalette(TCommandEnabler &tce)
{
  int index = -1;
  switch (tce.Id) {
        case CM_WINDOWBASIC_ELEMENTS_PALETTE: index = 0; break;
        case CM_WINDOWSHAPE_PALETTE:          index = 1; break;
        case CM_WINDOWATTRIBUTES_PALETTE:    index = 2; break;
        case CM_WINDOWCOMPOUNDCOMPLEX_PALETTE:index = 3; break;
  };

  if (index != -1) {
    if (PaletteOpen[index])
          tce.SetCheck(TCommandEnabler::Checked);
    else tce.SetCheck(TCommandEnabler::Unchecked);
  }
}

//////////////////////////////////////////////////////
// BottleApp
// =====
//
BottleApp::BottleApp () : TApplication("Bottle - Modeling And Simulation")
{
//    ::VBXEnableDLL(*this,*ResModule);
     ResModule = 0;
     Printer = 0;
     Printing = 0;

     SetDocManager(new TDocManager(dmMDI, this));
}

BottleApp::~BottleApp ()
{
     if (Printer)
          delete Printer;

     // INSERT>> Your destructor code here.
}

void BottleApp::CreateGadgets (TControlBar *cb, bool server)
{
     if (!server) {
          cb->Insert(*new TButtonGadget(CM_MDIFILENEW, CM_MDIFILENEW));
          cb->Insert(*new TButtonGadget(CM_MDIFILEOPEN, CM_MDIFILEOPEN));
          cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
          cb->Insert(*new TSeparatorGadget(6));
     }

     cb->Insert(*new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
     cb->Insert(*new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
     cb->Insert(*new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
     cb->Insert(*new TSeparatorGadget(6));
     cb->Insert(*new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
     cb->Insert(*new TSeparatorGadget(6));
     cb->Insert(*new TButtonGadget(CM_EDITFIND, CM_EDITFIND));
     cb->Insert(*new TButtonGadget(CM_EDITFINDNEXT, CM_EDITFINDNEXT));

     if (!server) {
          cb->Insert(*new TSeparatorGadget(6));
          cb->Insert(*new TButtonGadget(CM_FILEPRINT, CM_FILEPRINT));
          cb->Insert(*new TButtonGadget(CM_FILEPRINTPREVIEW, CM_FILEPRINTPREVIEW));
     }

     // Add fly-over help hints.
     cb->SetHintMode(TGadgetWindow::EnterHints);
}

void BottleApp::SetupSpeedBar (TDecoratedMDIFrame *frame)
{
     //
     // Create default toolbar New and associate toolbar buttons with commands.
     //
     TControlBar* cb = new TControlBar(frame);
     CreateGadgets(cb);

     // Setup the toolbar ID used by OLE 2 for toolbar negotiation.
     cb->Attr.Id = IDW_TOOLBAR;

     frame->Insert(*cb, TDecoratedFrame::Top);
}


//////////////////////////////////////////////////////
// BottleApp
// =====
// Application intialization.
//
```

```
void BottleApp::InitMainWindow ()
{
     if (nCmdShow != SW_HIDE)
          nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;

     mdiClient = new BottleMDIClient(this);
     TDecoratedMDIFrame* frame = new TDecoratedMDIFrame(Name, MDI_MENU, *mdiClient, true, this);

     nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;

     //
     // Assign ICON w/ this application.
     //
     frame->SetIcon(this, IDI_MDIAPPLICATION);

     //
     // Menu associated with window and accelerator table associated with table.
     //
     frame->AssignMenu(MDI_MENU);

     //
     // Associate with the accelerator table.
     //
     frame->Attr.AccelTable = MDI_MENU;

     SetupSpeedBar(frame);

     TStatusBar *sb = new TStatusBar(frame, TGadget::Recessed,
                                                  TStatusBar::CapsLock        |
                                                  TStatusBar::NumLock         |
                                                  TStatusBar::ScrollLock      |
                                                  TStatusBar::Overtype);
     frame->Insert(*sb, TDecoratedFrame::Bottom);

     SetMainWindow(frame);

     frame->SetMenuDescr(TMenuDescr(MDI_MENU));

//   ResModule = new TModule("WORKRES.DLL");
//   ResModule = new TModule("WORKED3.DLL");
     ResModule = new TModule("WORKLIB1.DLL");
     ResModule = new TModule("WORKLIB2.DLL");

     CmHelpAbout();
}

////////////////////////////////////////////////////////
// BottleApp
// =====
// Response Table handlers:
//
void BottleApp::EvNewView (TView& view)
{
     TMDIClient *mdiClient = TYPESAFE_DOWNCAST(GetMainWindow()->GetClientWindow(), TMDIClient);
     if (mdiClient) {
          BottleMDIChild* child = new BottleMDIChild(*mdiClient, 0, view.GetWindow());

          // Associate ICON w/ this child window.
          child->SetIcon(this, IDI_DOC);
          child->Create();

          ((BondGraphView*) &view)->tl = &tl;
     }
}

void BottleApp::EvCloseView (TView&)
{
}

////////////////////////////////////////////////////////
// BottleApp
// ===========
// Menu Help About bottle.exe command
void BottleApp::CmHelpAbout ()
{
     //
     // Show the modal dialog.
     //
     BottleAboutDlg* dlg = new BottleAboutDlg(GetMainWindow(),IDD_ABOUT);
     dlg->Execute();
}

void BottleApp::InitInstance ()
{
     TApplication::InitInstance();

     // Accept files via drag/drop in the frame window.
```

```
    GetMainWindow()->DragAcceptFiles(true);

    // Reposition Main Window To Center of Screen

    TScreenDC dc;
    TRect     screenRect(0,0,dc.GetDeviceCaps(HORZRES),dc.GetDeviceCaps(VERTRES)),
              oldPositionRect(GetMainWindow()->GetWindowRect()),
              newPositionRect(TPoint((screenRect.Width() - oldPositionRect.Width()) / 2,
                                     (screenRect.Height() - oldPositionRect.Height()))/
2),
                              TSize(oldPositionRect.Width(), oldPositionRect.Height()));
    GetMainWindow()->MoveWindow(newPositionRect,true);

    CreateBasicPalette(GetMainWindow(),&Palette[0],&PaletteOpen[0]);
    CreateShapePalette(GetMainWindow(),&Palette[1],&PaletteOpen[1]);
    CreateAttributePalette(GetMainWindow(),&Palette[2],&PaletteOpen[2]);
    CreateCompoundPalette(GetMainWindow(),&Palette[3],&PaletteOpen[3]);

    tl.AddTool(new SelectionTool(CM_ARROW));
    tl.AddTool(new BasicElemTool(CM_0JUNCTION, BG_0JUNCTION));
    tl.AddTool(new BasicElemTool(CM_1JUNCTION, BG_1JUNCTION));
    tl.AddTool(new BasicElemTool(CM_ESOURCE,   BG_ESOURCE));
    tl.AddTool(new BasicElemTool(CM_FSOURCE,   BG_FSOURCE));
    tl.AddTool(new BasicElemTool(CM_RESISTOR,  BG_RESISTOR));
    tl.AddTool(new BasicElemTool(CM_CAPACITANCE, BG_CAPACITOR));
    tl.AddTool(new BasicElemTool(CM_INDUCTANCE,BG_INDUCTOR));
    tl.AddTool(new BasicElemTool(CM_TRANSFORM, BG_TRANSFORMER));
    tl.AddTool(new BasicElemTool(CM_GYRATOR,   BG_GYRATOR));
    tl.AddTool(new ShapeTool(CM_LINE,   SH_LINE));
    tl.AddTool(new ShapeTool(CM_SQUARE, SH_RECTANGLE));
    tl.AddTool(new ShapeTool(CM_CIRCLE, SH_ELLIPSE));
    tl.AddTool(new ShapeTool(CM_TEXT,   SH_TEXT));
    tl.AddTool(new ShapeTool(CM_BITMAP, SH_BITMAP));
    tl.AddTool(new BondTool(CM_BOND));
    tl.AddTool(new BondTool(CM_SBOND,1));
    tl.AddTool(new BondTool(CM_BONDDIR,3));
    tl.AddTool(new BondTool(CM_BONDCAUS,4));
    tl.AddTool(new CompoundTool(CM_COMPOUND_EMPTY));
    tl.AddTool(new CompoundTool(CM_COMPOUND,((CompoundPalette*)Palette[3])-
>CompoundOutline,((CompoundPalette*)Palette[3])->CompoundLink));
}

void BottleApp::EvDropFiles (TDropInfo drop)
{
    // Number of files dropped.
    int totalNumberOfFiles = drop.DragQueryFileCount();

    TFileList* files = new TFileList;

    for (int i = 0; i < totalNumberOfFiles; i++) {
        // Tell DragQueryFile the file interested in (i) and the length of your buffer.
        int     fileLength = drop.DragQueryFileNameLen(i) + 1;
        char    *fileName = new char[fileLength];

        drop.DragQueryFile(i, fileName, fileLength);

        // Getting the file dropped. The location is relative to your client coordinates,
        // and will have negative values if dropped in the non client parts of the window.
        //
        // DragQueryPoint copies that point where the file was dropped and returns whether
        // or not the point is in the client area.  Regardless of whether or not the file
        // is dropped in the client or non-client area of the window, you will still receive
        // the file name.
        TPoint  point;
        bool    inClientArea = drop.DragQueryPoint(point);
        files->Add(new TFileDrop(fileName, point, inClientArea, this));
    }

    // Open the files that were dropped.
    AddFiles(files);

    // Release the memory allocated for this handle with DragFinish.
    drop.DragFinish();
}

void BottleApp::AddFiles (TFileList* files)
{
    // Open all files dragged in.
    TFileListIter fileIter(*files);

    while (fileIter) {
        TDocTemplate* tpl = GetDocManager()->MatchTemplate(fileIter.Current()->WhoAmI());
        if (tpl)
            tpl->CreateDoc(fileIter.Current()->WhoAmI());
        fileIter++;
    }
}
```

```
void BottleApp::EvWinIniChange (char far* section)
{
     if (strcmp(section, "windows") == 0) {
          // If the device changed in the WIN.INI file then the printer
          // might have changed.  If we have a TPrinter (Printer) then
          // check and make sure it's identical to the current device
          // entry in WIN.INI.
          if (Printer) {
               char printDBuffer[255];
               LPSTR printDevice = printDBuffer;
               LPSTR devName;
               LPSTR driverName = 0;
               LPSTR outputName = 0;

               if (::GetProfileString("windows", "device", "", printDevice, sizeof(printDevice))) {
                    // The string which should come back is something like:
                    //
                    //      HP LaserJet III,hppcl5a,LPT1:
                    //
                    // Where the format is:
                    //
                    //      devName,driverName,outputName
                    //
                    devName = printDevice;
                    while (*printDevice) {
                         if (*printDevice == ',') {
                              *printDevice++ = 0;
                              if (!driverName)
                                   driverName = printDevice;
                              else
                                   outputName = printDevice;
                         } else
                              printDevice = ::AnsiNext(printDevice);
                    }

                    if ((Printer->GetSetup().Error != 0)                                    ||
                         (strcmp(devName, Printer->GetSetup().GetDeviceName()) != 0)     ||
                         (strcmp(driverName, Printer->GetSetup().GetDriverName()) != 0) ||
                         (strcmp(outputName, Printer->GetSetup().GetOutputName()) != 0)) {

                         // New printer installed so get the new printer device now.
                         delete Printer;
                         Printer = new TPrinter(this);
                    }
               } else {
                    // No printer installed (GetProfileString failed).
                    delete Printer;
                    Printer = new TPrinter(this);
               }
          }
     }
}

int OwlMain (int , char* [])
{
     try {
          TBIVbxLibrary vbxLib;
          BottleApp    app;
          return app.Run();
     }
     catch (xmsg& x) {
          ::MessageBox(0, x.why().c_str(), "Exception", MB_OK);
     }
     return -1;
}
```

bttlabtd.h

```
#if !defined(__bttlabtd_h)              // Sentry, use file only if it's not already included.
#define __bttlabtd_h

/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         bttlabtd.h
    AUTHOR:       Richard K. Branton


    OVERVIEW
     ========
     Class definition for BottleAboutDlg (TDialog).
*/
```

```
#include <owl\owlpch.h>
#pragma hdrstop

#include "bttlapp.rh"                  // Definition of all resources.
#include "bttlbtn.h"

//{{TDialog = BottleAboutDlg}}
class BottleAboutDlg : public TDialog {
protected:
     BottleBitmapBtn* AboutPicture;

public:
     BottleAboutDlg (TWindow *parent, TResId resId = IDD_ABOUT, TModule *module = 0);
     virtual ~BottleAboutDlg ();

//{{BottleAboutDlgVIRTUAL_BEGIN}}
public:
     void SetupWindow ();
//{{BottleAboutDlgVIRTUAL_END}}

//{{BottleAboutDlgRSP_TBL_BEGIN}}
protected:
    void EvTimer (uint timerId);
//{{BottleAboutDlgRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(BottleAboutDlg);
};    //{{BottleAboutDlg}}

#endif                                          // __bttlabtd_h sentry.
```

bttlabtd.cpp

```
/*  Project bottle
     Massachusetts Institute of Technology
     Copyright © 1995. All Rights Reserved.

     SUBSYSTEM:    bottle.exe Application
     FILE:         bttlabtd.cpp
     AUTHOR:       Richard K. Branton


     OVERVIEW
     ========
     Source file for implementation of BottleAboutDlg (TDialog).
*/


//#include <owl\owlpch.h>
//#pragma hdrstop

//#if !defined(__FLAT__)
//#include <ver.h>
//#endif

#include "bttlapp.h"
#include "bttlabtd.h"

//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(BottleAboutDlg, TDialog)
//{{BottleAboutDlgRSP_TBL_BEGIN}}
    EV_WM_TIMER,
//{{BottleAboutDlgRSP_TBL_END}}
END_RESPONSE_TABLE;


//{{BottleAboutDlg Implementation}}

//////////////////////////////////////////////////////////
// BottleAboutDlg
// ==========
// Construction/Destruction handling.
BottleAboutDlg::BottleAboutDlg (TWindow *parent, TResId resId, TModule *module)
     : TDialog(parent, resId, module)
{
     // INSERT>> Your constructor code here.
}


BottleAboutDlg::~BottleAboutDlg ()
{
     Destroy();

     // INSERT>> Your destructor code here.
```

```
}


void BottleAboutDlg::SetupWindow ()
{
    AboutPicture = new BottleBitmapBtn(this,IDOK,BMP_ABOUT);
    AboutPicture->Create();

    // Center Window On Screen
    TScreenDC dc;
    TRect     screenRect(0,0,dc.GetDeviceCaps(HORZRES),dc.GetDeviceCaps(VERTRES)),
              oldPositionRect(GetWindowRect()),
              newPositionRect(TPoint((screenRect.Width() - oldPositionRect.Width()) / 2,
                                     (screenRect.Height() - oldPositionRect.Height())/
2),
                              TSize(oldPositionRect.Width(), oldPositionRect.Height()));
    MoveWindow(newPositionRect,true);

    // Wait 6 Seconds and then go away.
    SetTimer(100,6000);

    TDialog::SetupWindow();
}

void BottleAboutDlg::EvTimer (uint timerId)
{
     TDialog::EvTimer(timerId);

     // INSERT>> Your code here.
     KillTimer(100);
     CloseWindow();
}
```

bttlmdic.h

```
#if !defined(__bttlmdic_h)              // Sentry, use file only if it's not already included.
#define __bttlmdic_h

/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         bttlmdic.h
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Class definition for BottleMDIChild (TMDIChild).
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include "bttlapp.rh"            // Definition of all resources.

//{{TMDIChild = BottleMDIChild}}
class BottleMDIChild : public TMDIChild {
public:
     BottleMDIChild (TMDIClient &parent, const char far *title, TWindow *clientWnd, bool
shrinkToClient = false, TModule* module = 0);
     virtual ~BottleMDIChild ();
};    //{{BottleMDIChild}}


#endif                                      // __bttlmdic_h sentry.
```

bttlmdic.cpp

```
/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         bttlmdic.cpp
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Source file for implementation of BottleMDIChild (TMDIChild).
*/
```

```
#include <owl\owlpch.h>
#pragma hdrstop

#include "bttlapp.h"
#include "bttlmdic.h"


//{{BottleMDIChild Implementation}}


//////////////////////////////////////////////////////
// BottleMDIChild
// ==========
// Construction/Destruction handling.
BottleMDIChild::BottleMDIChild (TMDIClient &parent, const char far *title, TWindow *clientWnd, bool
shrinkToClient, TModule *module)
     : TMDIChild (parent, title, clientWnd, shrinkToClient, module)
{
     // INSERT>> Your constructor code here.
}


BottleMDIChild::~BottleMDIChild ()
{
    Destroy();

    // INSERT>> Your destructor code here.

}
```

bttlmdi.h

```
#if !defined(__bttlmdi_h)              // Sentry, use file only if it's not already included.
#define __bttlmdi_h

/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         bttlmdi.h
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Class definition for BottleMDIClient (TMDIClient).
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include "bttlapp.rh"            // Definition of all resources.


//{{TMDIClient = BottleMDIClient}}
class BottleMDIClient : public TMDIClient {
public:
    int                     ChildCount;                 // Number of child window created.

    BottleMDIClient(TModule* module = 0);
    virtual ~BottleMDIClient ();

    void OpenFile (const char *fileName = 0);

private:
    void LoadTextFile ();

//{{BottleMDIClientVIRTUAL_BEGIN}}
protected:
    virtual void SetupWindow ();
//{{BottleMDIClientVIRTUAL_END}}

//{{BottleMDIClientRSP_TBL_BEGIN}}
protected:
    void CmFilePrint ();
    void CmFilePrintSetup ();
    void CmFilePrintPreview ();
    void CmPrintEnable (TCommandEnabler &tce);
    void EvDropFiles (TDropInfo);
//{{BottleMDIClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(BottleMDIClient);
};    //{{BottleMDIClient}}
```

```
#endif                                  // __bttlmdi_h sentry.
```

bttlmdi.cpp

```
/*  Project bottle
    Massachusetts Institute of Technology
    Copyright © 1995. All Rights Reserved.

    SUBSYSTEM:    bottle.exe Application
    FILE:         bttlmdi.cpp
    AUTHOR:       Richard K. Branton


    OVERVIEW
    ========
    Source file for implementation of BottleMDIClient (TMDIClient).
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include <dir.h>

#include "bttlapp.h"
#include "bttlmdic.h"
#include "bttlmdi.h"
#include "apxprint.h"
#include "apxprev.h"


//{{BottleMDIClient Implementation}}


//
// Build a response table for all messages/commands handled
// by BottleMDIClient derived from TMDIClient.
//
DEFINE_RESPONSE_TABLE1(BottleMDIClient, TMDIClient)
//{{BottleMDIClientRSP_TBL_BEGIN}}
    EV_COMMAND(CM_FILEPRINT, CmFilePrint),
    EV_COMMAND(CM_FILEPRINTERSETUP, CmFilePrintSetup),
    EV_COMMAND(CM_FILEPRINTPREVIEW, CmFilePrintPreview),
    EV_COMMAND_ENABLE(CM_FILEPRINT, CmPrintEnable),
    EV_COMMAND_ENABLE(CM_FILEPRINTERSETUP, CmPrintEnable),
    EV_COMMAND_ENABLE(CM_FILEPRINTPREVIEW, CmPrintEnable),
    EV_WM_DROPFILES,
//{{BottleMDIClientRSP_TBL_END}}
END_RESPONSE_TABLE;


//////////////////////////////////////////////////////////
// BottleMDIClient
// ===========
// Construction/Destruction handling.
BottleMDIClient::BottleMDIClient (TModule* module)
 : TMDIClient (module)
{
    // Change the window's background color
    SetBkgndColor(RGB(0xff, 0xff, 0xff));

    ChildCount = 0;

    // INSERT>> Your constructor code here.

}


BottleMDIClient::~BottleMDIClient ()
{
    Destroy();

    // INSERT>> Your destructor code here.

}


//////////////////////////////////////////////////////////
// BottleMDIClient
// ===========
// MDIClient site initialization.
void BottleMDIClient::SetupWindow ()
{
    // Default SetUpWindow processing.
```

```
    TMDIClient::SetupWindow ();

    // Accept files via drag/drop in the client window.
    DragAcceptFiles(true);
}


////////////////////////////////////////////////////////
// BottleMDIClient
// ==========
// Menu File Print command
void BottleMDIClient::CmFilePrint ()
{
    //
    // Create Printer object if not already created.
    //
    BottleApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), BottleApp);
    if (theApp) {
        if (!theApp->Printer)
            theApp->Printer = new TPrinter(GetApplication());

        //
        // Create Printout window and set characteristics.
        //
        APXPrintOut printout(theApp->Printer, Title, GetActiveMDIChild()->GetClientWindow(), true);

        theApp->Printing++;

        //
        // Bring up the Print dialog and print the document.
        //
        theApp->Printer->Print(GetWindowPtr(GetActiveWindow()), printout, true);

        theApp->Printing--;
    }
}


////////////////////////////////////////////////////////
// BottleMDIClient
// ==========
// Menu File Print Setup command
void BottleMDIClient::CmFilePrintSetup ()
{
    BottleApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), BottleApp);
    if (theApp) {
        if (!theApp->Printer)
            theApp->Printer = new TPrinter(GetApplication());

        //
        // Bring up the Print Setup dialog.
        //
        theApp->Printer->Setup(this);
    }
}


////////////////////////////////////////////////////////
// BottleMDIClient
// ==========
// Menu File Print Preview command
void BottleMDIClient::CmFilePrintPreview ()
{
    BottleApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), BottleApp);
    if (theApp) {
        if (!theApp->Printer)
            theApp->Printer = new TPrinter(GetApplication());

        theApp->Printing++;

        PreviewWindow *prevW = new PreviewWindow(Parent, theApp->Printer, GetActiveMDIChild()-
>GetClientWindow(), "Print Preview", new TLayoutWindow(0));
        prevW->Create();

        GetApplication()->BeginModal(GetApplication()->GetMainWindow());

        // We must destroy the preview window explicitly.  Otherwise, the window will not be destroyed
until
        // it's parent the MainWindow is destroyed.
        prevW->Destroy();
        delete prevW;

        theApp->Printing--;
    }
}
```

```
//////////////////////////////////////////////////////
// BottleMDIClient
// ===========
// Menu enabler used by Print, Print Setup and Print Preview.
void BottleMDIClient::CmPrintEnable (TCommandEnabler &tce)
{
    if (GetActiveMDIChild()) {
        BottleApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), BottleApp);
        if (theApp) {
            // If we have a Printer already created just test if all is okay.
            // Otherwise create a Printer object and make sure the printer
            // really exists and then delete the Printer object.
            if (!theApp->Printer) {
                theApp->Printer = new TPrinter(GetApplication());

                tce.Enable(theApp->Printer->GetSetup().Error == 0);
            } else
                tce.Enable(theApp->Printer->GetSetup().Error == 0);
        }
    } else
        tce.Enable(false);
}


void BottleMDIClient::EvDropFiles (TDropInfo)
{
    Parent->ForwardMessage();
}
```

References

[1] Youcef-Toumi, K., "Modelling and Simulation Tools for the Rapid Virtual Prototyping of Multi-Energy Domain Systems," Mechanical Engineering Department, MIT. 1994.

[2] Karnopp, D.C. and Rosenberg, R. "System Dynamics: a Unified Approach," John-Wiley & Sons. Newy-York, 1988.

[3] Paynter, H.M. "Analysis and Design of Engineering Systems," MIT Press, 1961.

[4] Youcef-Toumi, K., "Modelling and Control of Physical Systems," 2.151 course notes, graduate course at the Mechanical Engineering Department, MIT.

[5] Karnopp, D.C. "Alternative Bond Graph Causal Patterns and Equation Formulation for Dynamic Systems," ASME Journal of Dyanmic Systems, Measurement, and Control, Vol. 105, No.2, PP.58-63, 1983.

[6] Rosenberg, R.C. "State space formulation for bond graph models of multiport systems," Trans. ASME, J. Dynamic Syst. Measure. Control, Vol. 93, No.1, pp.35-40, 1971.

[7] Filippo, M.J., Delgado, M., Brie, C. and Paynter, H.M. "A Survey of Bond Graphs: Theory, Application and Programs," Journal of Franklin Institute, Vol.328, No.5/6, pp.565-606, 1991.

[8] Hogan, N. "Modeling, Analysis and Control of Physical Systems," Mechanical Engineering Department, MIT. 1989.

[9] Swan, T., Arnson, R. and Cantu, M. "ObjectWindows 2.0 Programming," Random House Electronic Publishing, NY ,1994.