A Case Study in the Software Development of an Icon Graphic Image Editor

Jiazhong Tu

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

April 2003

0-612-77722-7

ABSTRACT

A Case Study in the Software Development of an Icon Graphic Image Editor

Jiazhong Tu

This major report focus on the software development process from requirements through to implementation. The objective of the report is to create better object-oriented designs and to build reusable software. An Icon Graphic Image Editor is developed under Window NT/2000. The editor can add text and graphics freely in a variety of formatting styles. Iterative development process is followed in the software analysis and design. Software Requirements Specification of IGI Editor and Software Design Descriptions of IGI Editor are created. The editor is implemented with Microsoft Visual C++ (version 6.0).

Acknowledgements

I would like to acknowledge the assistance of my supervisor, Olga Ormandjieva. in offering advice and assistance during the preparation of this report.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. SOFTWARE REQUIREMENTS SPECIFICATION OF IGI EDITOR

1.1 Introduction

1.1.1 Purpose

The purpose of the Software Requirement Specification (SRS) is to define the software requirements of the Icon Graphic Image Editor (IGIEditor).

1.1.2 Problem Statement

Using the computer aided engineering technology, the complicated interactions of thousands of electrical, liquid, and gas system components can be illustrated in minute detail on a computer screen.

A software package "CADS (Computer Aided Design Simulator)" is assumed be developed as an engineering design tool through modeling and simulation.

The essential task of CADS is to take what is hidden and complex, and make it visible and understandable. This is accomplished by illustrating the behavior of physical systems through user-friendly graphical representation of how specific systems are likely to perform in actual operation. The simulations can then be run in real-time, or at user-defined accelerated or decelerated pace.

The main components of the CADS are SimCADE, SimENGINE, and SimMODEL.

SimCADE is a graphical user interface (GUI) that permits the creation of a system diagaram, consisting of a process and its related controls, to be designed by simply dropping the process and control objects from the model libraries onto the screen and connecting them together. The user is able to run single and/or multiple system diagrams and predict the interaction between them.

SimENGINE manages the calculations of the modeling equation. The user has the capability to start, stop, and run the process in real-time; then to accelerate or to slow the simulation by adjusting the time parameters in order to analyze fast dynamics or slow response processes. Instant capture of running simulations can be kept for future simulation restart points.

SimMODEL enables the user to customize the CADS built-in libraries and create user-defined model libraries that can be used directly in the simulations, thus providing an unsurpassed flexibility in the type of applications that can be built to suit the exact needs of the user. New models can be built using MS-Visual C++.

An Icon Graphic Image (IGI) is the graphical representation of the simulated model as displayed to the user on the screen.

The IGIEditor is one of the modules in the SimMODEL that is used to create an icon graphic image for a simulated model. The editor can add text and graphics freely in a variety of formatting styles to an IGI.

1.1.3 References

COMP354 Course Notes by Professor Olga Ormandjieva

- IEEE template for SRS Document

1.1.4 Overview

Section 1.2 of the specification describes the general factors that affect the product and its requirements. This section contains background information, not state specific requirements for the editor.

Section 1.3 describes all software requirements at a sufficient level of detail for designers to design the software satisfying the requirements and testers to verity that the software satisfies requirements.

Every stated requirement can be externally perceivable by users, operators or other external systems.

Section 1.4 lists all terms, acronyms, and abbreviations used in this specification in alphabetical order.

Future growth requirements included in this document will be identified with FUTURE GROWTH in parenthesis.

Paragraphs indicated as N/A or Not applicable or RESERVED are not applicable to this product variation, but are preserved for compatibility with other variations.

1.2 Overall Description

1.2.1 Product Perspective

The IGIEditor is used to create an icon graphic image for a simulated object. A user can edit an IGI via adding a primitive object to IGI edit area easily with graphic user interface. An IGI includes the following primitives:

Table 1-1 IGI Primitives

Primitive	Graphic symbol
Line	———
Rectangle	▭
Fill rectangle	▬
Triangle	◺
Fill triangle	◣
Ellipse	⬭
Fill Ellipse	⬬
Text	Text example
Tag name	Tag name
Process input stream	◀———
Process output stream	———▶
Logic input stream	◀- - - - - - -
Logic output stream	- - - - - - -▶

Primitive	Graphic symbol
Analog input stream	◀—·—·—·—·—·—
Analog output stream	—·—·—·—·—·—▶

An IGI has only one tag name that has default name “xxxxxxxxxxxxxx”. It will be changed to specific name when user builds a model for the simulated object. There is no limitation for other primitives and user can add a primitive object any time to an IGI depending on its necessary.

One of the following colors can be selected for each primitive object: Black, Blue, Green, Cyan, Red, Magenta, Yellow, White, Dark blue, Dark green, Dark cyan, Dark red, Dark magenta, Dark yellow, Dark gray, and Light gray. The following table shows the RGB value for each primitive object color.

Table 1-2 IGI Primitive color

Color	Red	Green	Blue	Symbol
Black	0	0	0	■
Blue	0	0	255	■
Green	0	255	0	■
Cyan	0	255	255	▨
Red	255	0	0	■
Magenta	255	0	255	■
Yellow	255	255	0	□
White	255	255	255	□
Dark blue	0	0	128	■
Dark green	0	128	0	■
Dark cyan	0	128	128	■
Dark red	128	0	0	■
Dark magenta	128	0	128	■
Dark yellow	128	128	0	■
Dark gray	128	128	128	■
Light gray	192	192	192	▨

1.2.1.1 System Interfaces

Not applicable.

1.2.1.2 User Interfaces

The IGIEditor shall provide a user interface by controlling the following:

- Menu
- Toolbar
    - Main toolbar
    - Primitive palette
    - Color palette
- Status bar
- Message
- IGI edit area

The layout of the user interface is shown below in Figure 1-1.

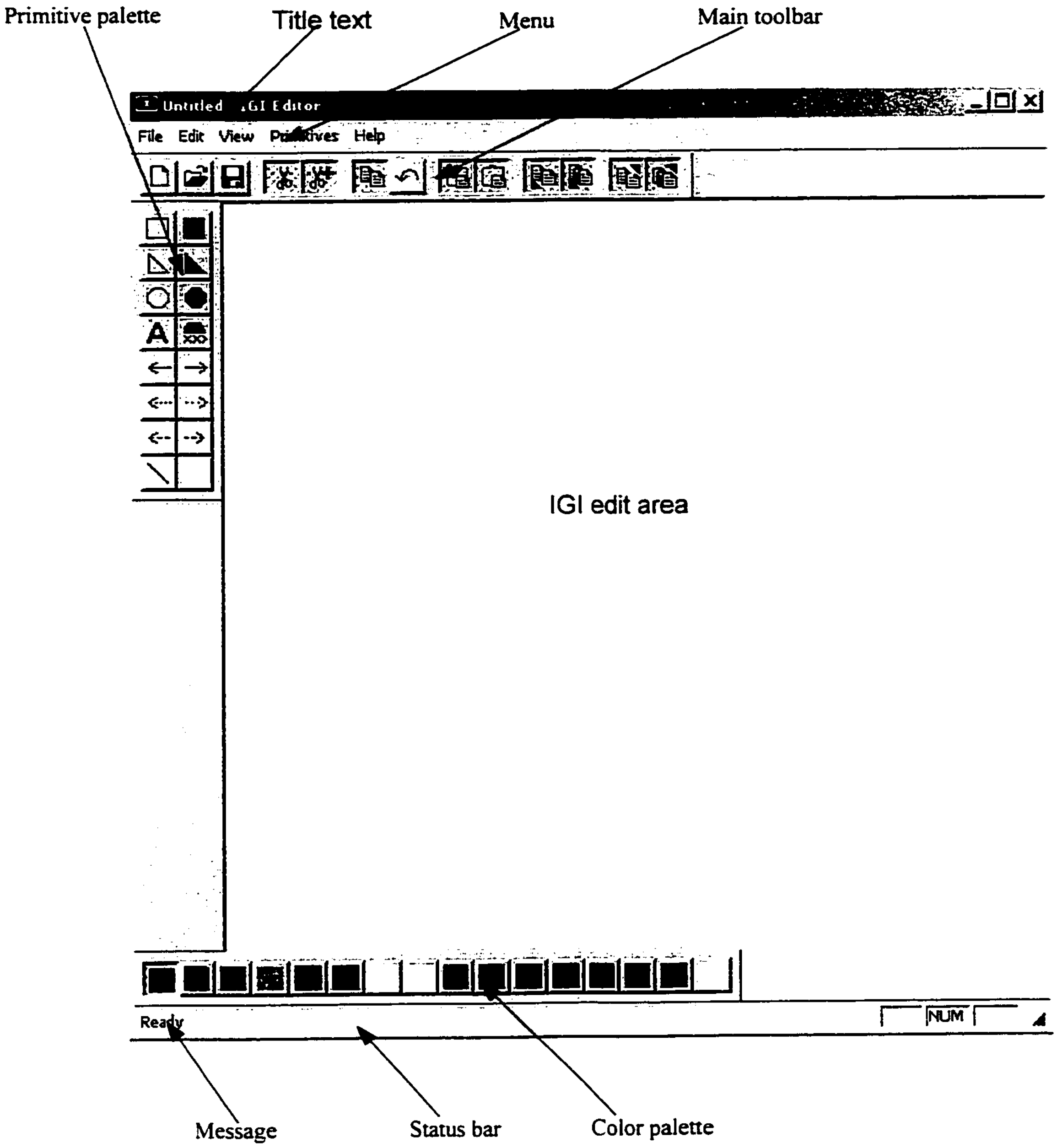


Figure 1-1 User Interface Layout

1.2.1.2.1 Menu configuration

1. The main menu includes: File, Edit, Tools, Primitives, and Help. (Refer to Figure 1. User Interface Layout).

2. User can select a submenu item to execute a command.

3. Menu configuration and their functionality are shown in the following table:

Table 1-3 Menu configuration and functionality

Main Menu	Submenu	Functionality
File	New	Create a new document
File	Open	Open an existing document
File	Save	Save the active document
File	Save as	Save the active document with a new name
File	Exit	Quit the application; prompts to save documents
Edit	Cut	Cut the selected primitive(s)
Edit	Cut All	Cut all primitives
Edit	Copy	Copy the selected primitive(s)
Edit	Undo	Undo the last action
Edit	Redo	Redo the previously undone action
Edit	Select All	Select all the primitives
Edit	Unselect All	Unselect all the primitives
Edit	Push Down	Push the selected primitive one layer down
Edit	Push To Bottom	Push a selected primitive to bottom
Edit	Pop Up	Pop the selected primitive one layer up
Edit	Pop To Top	Pop a selected primitive to top
View	Toolbars	Configure all the Toolbars
View	Status Bar	Enable/disable Status Bar
Primitives	Rectangle	Create a new rectangle
Primitives	Fill Rectangle	Create a new fill rectangle
Primitives	Triangle	Create a new triangle
Primitives	Fill Triangle	Create a new fill triangle
Primitives	Ellipse	Create a new ellipse
Primitives	Fill Ellipse	Create a new fill ellipse
Primitives	Process input stream	Create a new process input stream
Primitives	Process output stream	Create a new process output stream
Primitives	Logic input stream	Create a new logic input stream
Primitives	Logic output stream	Create a new logic output stream
Primitives	Analog input stream	Create a new analog input stream
Primitives	Analog output stream	Create a new analog output stream
Primitives	Text	Create a new text
Primitives	Tag Name	Create a new tag name

Main Menu	Submenu	Functionality
Primitives	Line	Create a new line
Help	About IGIEditor	Display program information, version number and copyright

1.2.1.2.2 Toolbar configuration

1. User can enable/disable the main toolbar, primitive palette, and color palette.
2. User can show or hide the tool tips over the toolbar.
3. User can set the column number of primitive palette to 2, 3, 4, or 6.
4. User can move the main toolbar, primitive palette, and color palette to any where on the screen.

1.2.1.2.3 Status Bar configuration

1. Users can enable/disable the Status Bar.

1.2.1.2.4 Message Display

When the Status Bar is enabled:

1. The related message is displayed on the Status Bar when the mouse is over the submenu item, main toolbar, primitive palette, and color palette.

1.2.1.3 Hardware Interfaces

The hardware for the IGIEditor is formed of a standard personal computer, including a graphic screen, a keyboard and a two or three button mouse. The computer must preferably have a Pentium II processor or better, and have a minimum of 128 Megs of memory.

1.2.1.4 Software Interfaces

The Software will be run under Window NT/2000. It is developed with Microsoft Visual C++ (version 6.0) and Microsoft Foundation Class Library 4.0.

IGIEditor can only add process/logic/analog input/output stream according to the model structure that is defined with SimMODEL.

SimCADE can use the drawing functions in the IGIEditor to display an icon graphic image on the screen.

1.2.1.5 Communications Interfaces

Not applicable.

1.2.1.6 Memory Constraints

To run this program, the computer shall have a minimum of 128 Megs of memory.

1.2.1.7 Operations

There is no any special operation required by the user to use this editor.

1.2.1.8 Site Adaptation Requirements

Not applicable.

1.2.2 Product Functions

The major functions that the software will perform are IGI document management and IGI document edits.

The figure 1-2 shows the product use case diagram.

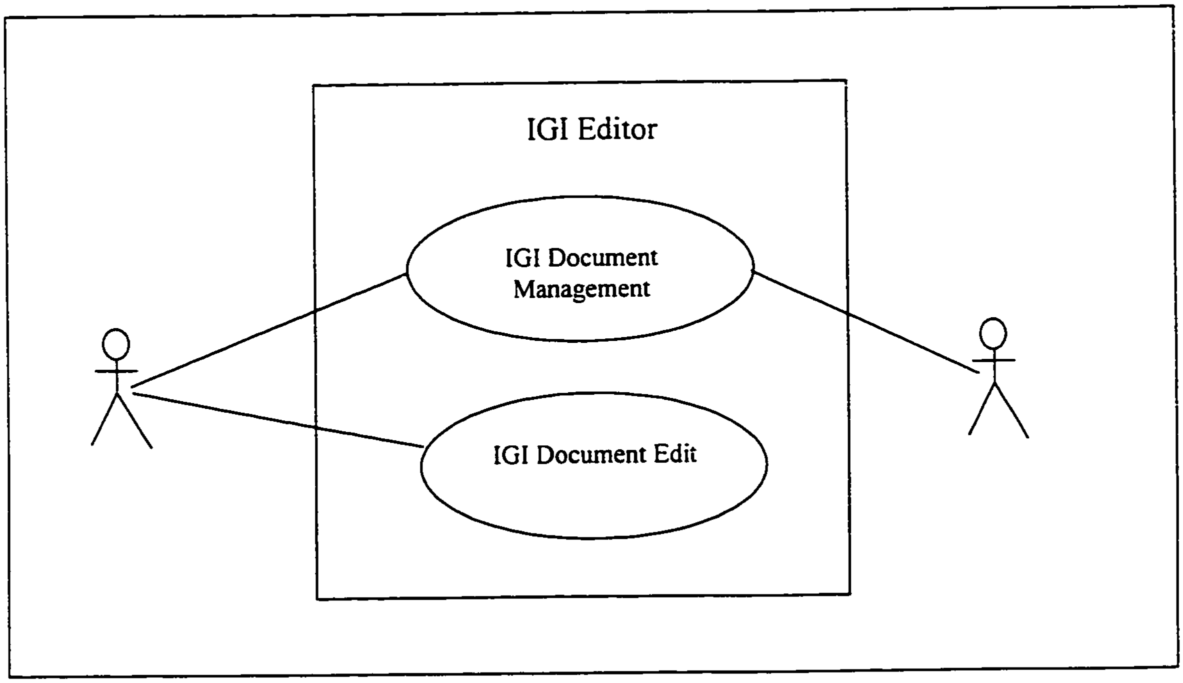


Figure 1-2 Use Case Diagram

1.2.2.1 IGI document management

User can:

1. create a new IGI document.
2. open an existing IGI document.
3. save an active IGI document.
4. save an active IGI document with a new name.

1.2.2.2 IGI document edit

After an IGI document is created or opened, user can:

1. add any primitive object to an IGI.
2. select any primitive object in the edit area one by one.
3. move any selected primitive object to any position in IGI edit area.
4. select all primitive objects in the edit area once.
5. unselect all selected primitive object in the edit area once.
6. delete any selected primitive object once.
7. delete all primitive objects once without select action.
8. undo the last edit action.
9. redo the previous undo action.
10. push a selected primitive object down one level each time
11. push a selected primitive object down to bottom once.
12. pop a selected primitive object up one level each time.
13. pop a selected primitive object up to top once.
14. copy any selected primitive objects continually.

15. change any selected primitive objects color.

16. rotate a selected stream object.

17. modify a selected primitive object.

1.2.3 User Characteristics

This software is intended to be used by the engineer, operator or other professional who has no training, or even experience, in engineering project simulation.

1.2.4 Constraints

There is no constraint for this program.

1.2.5 Assumptions and Dependencies

There is no assumptions and dependencies for this program.

1.3 Specific Requirements

1.3.1 External Interface Requirements

There are no specific external interface requirements for this project.

1.3.2 Software Product Features

1.3.2.1 Use case 1: User creates a new IGI document

1.3.2.1.1 Actor

An IGI Editor User.

1.3.2.1.2 Purpose

Actor creates a new untitled IGI document.

1.3.2.1.3 Preconditions

N/A.

1.3.2.1.4 Typical Course of Events

Table 1-4 Typical course of events of use case 1

Actor Action	System Response
1. Select "New" menu item, or click "New" toolbar button, or press "Ctrl+N" on keyboard.	
	2. The edit area becomes empty.
	3. The title text changes to "untitled".

1.3.2.1.5 Interaction Diagram for the main scenario

1.3.2.1.5.1 Collaboration diagram

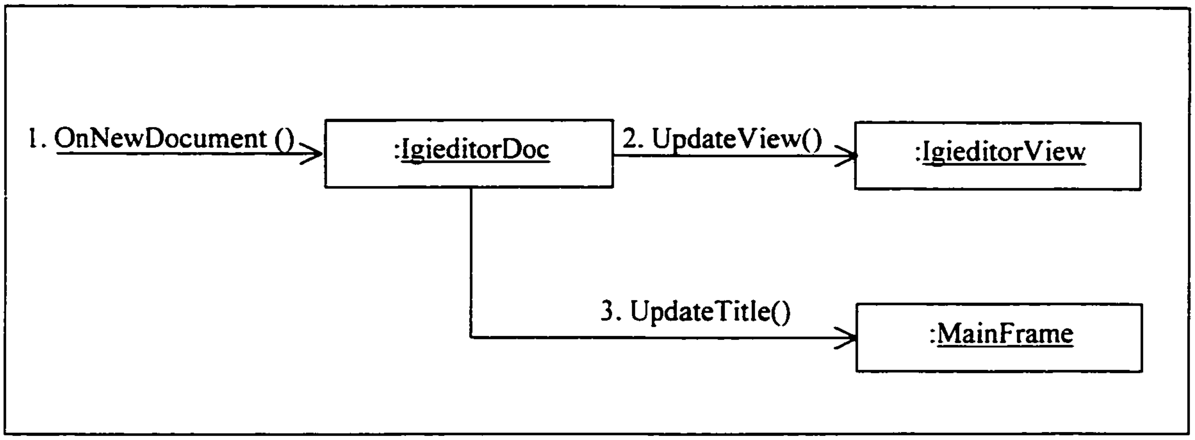


Figure 1-3 Collaboration diagram for use case 1

1.3.2.1.5.2 Sequence diagram

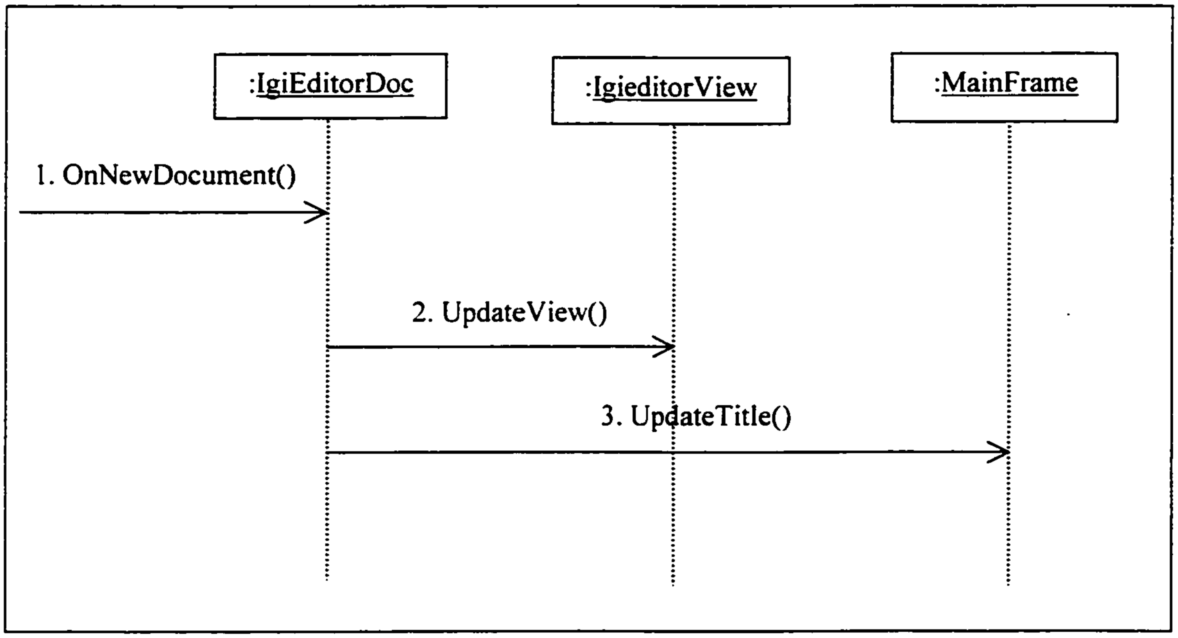


Figure 1-4 Sequence diagram for use case 1

1.3.2.2 Use case 2: User opens an existing IGI document

1.3.2.2.1 Actor

An IGI Editor User.

1.3.2.2.2 Purpose

Actor loads an existing IGI document into IGI Editor.

1.3.2.2.3 Preconditions

1. An active IGI exists in the edit area.
2. The active IGI name is displayed on the title bar.
3. The IGI that will be loaded exists on the hard disk.

1.3.2.2.4 Typical Course of Events

Table 1-5 Typical course of events of use case 2

Actor Action	System Response
1. Select "Open" menu item, or click "Open" toolbar button, or presses "Ctrl+O" on keyboard.	
	2. If the active IGI is not saved, pop up message window.
3. If a message window pops up, Click "Yes" to save file or "No" to not save file.	
	4. All the primitive objects are deleted in the edit area.
	5. The loaded IGI is displayed in the edit area.
	6. The title text changes to the loaded IGI name.

1.3.2.2.5 Interaction Diagram for the main scenario

1.3.2.2.5.1 Collaboration diagram

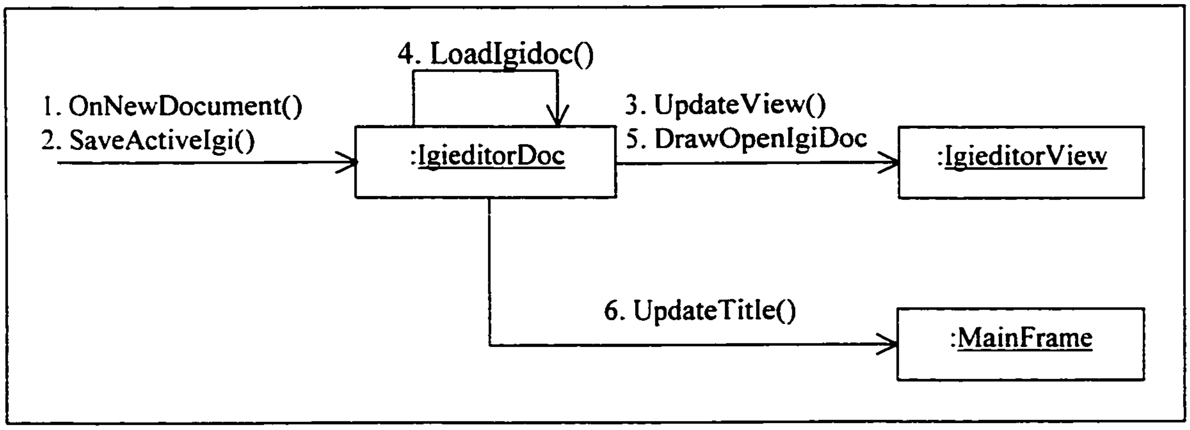


Figure 1-5 Collaboration diagram for use case 2

1.3.2.2.5.2 Sequence diagram

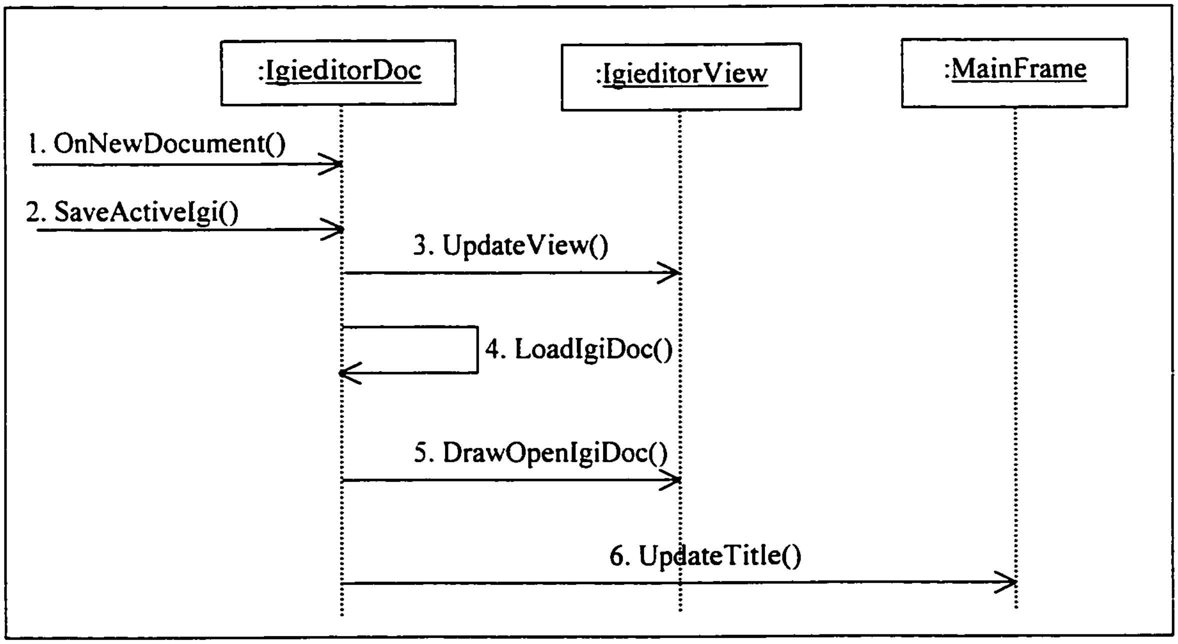


Figure 1-6 Sequence diagram for use case 2

1.3.2.3 Use case 3: User saves an active IGI document

1.3.2.3.1 Actor

An IGI Editor User.

1.3.2.3.2 Purpose

Actor saves a new IGI or a modified IGI to hard disk.

1.3.2.3.3 Preconditions

Some primitive objects are displayed in the edit area.

1.3.2.3.4 Typical Course of Events

Table 1-6 Typical course of events of use case 3

Actor Action	System Response
1. Select "Save" menu item, or click "Save" toolbar button, or presses "Ctrl+S" on keyboard.	
	2. If the active IGI is untitled, pop up "Save As" window.
3. If "Save As" message window pops up, select a folder and type a new IGI name. Click on "Save" button or type "Enter" key.	
	4. An IGI file is saved on the hard disk.
	5. If the active IGI is untitled, the title text changes to new IGI name.

Alternative Course

- Line 3: Click on "Cancel" button to cancel "Save" action.

1.3.2.3.5 Interaction Diagram for the main scenario

1.3.2.3.5.1 Collaboration diagram

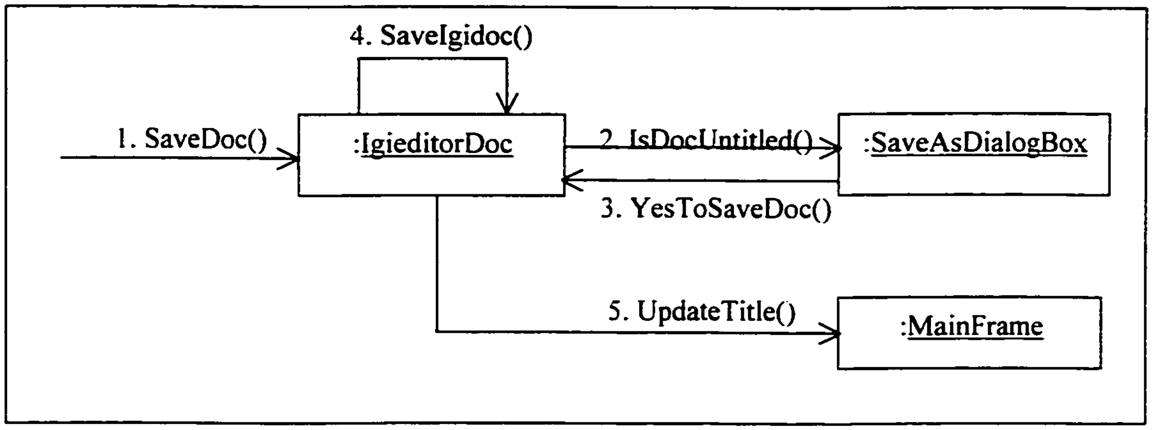


Figure 1-7 Collaboration diagram for use case 3

1.3.2.3.5.2 Sequence diagram

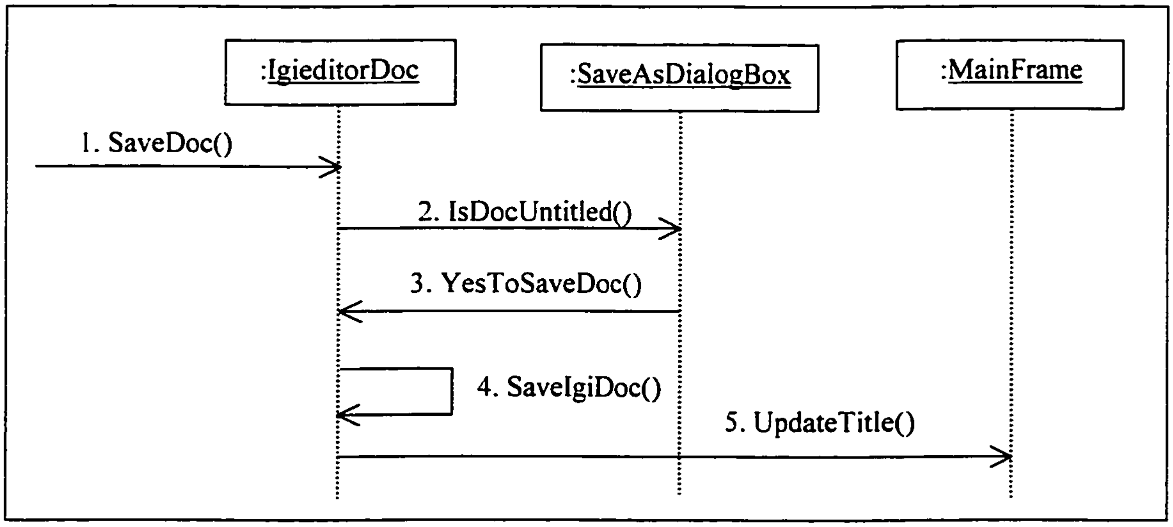


Figure 1-8 Sequence diagram for use case 3

1.3.2.4 Use case 4: User saves an active IGI document with a new name

1.3.2.4.1 Actor

An IGI Editor User.

1.3.2.4.2 Purpose

Actor saves an active IGI to another new IGI file.

1.3.2.4.3 Preconditions

1. An active IGI exists in the edit area.

2. The active IGI name is displayed on the title bar.

1.3.2.4.4 Typical Course of Events

Table 1-7 Typical course of events of use case 4

Actor Action	System Response
1. Select "Save As" menu item.	
	2. Pop up "Save As" window.
3. Select a folder and type a new IGI name. Click on "Save" button or type "Enter" key.	
	4. Create a new IGI file and save the active IGI to this new file.
	5. The title text changes to new IGI name.

Alternative Course

1. Line 3: Click on "Cancel" button to cancel "Save As" action.

1.3.2.4.5 Interaction Diagram for the main scenario

1.3.2.4.5.1 Collaboration diagram

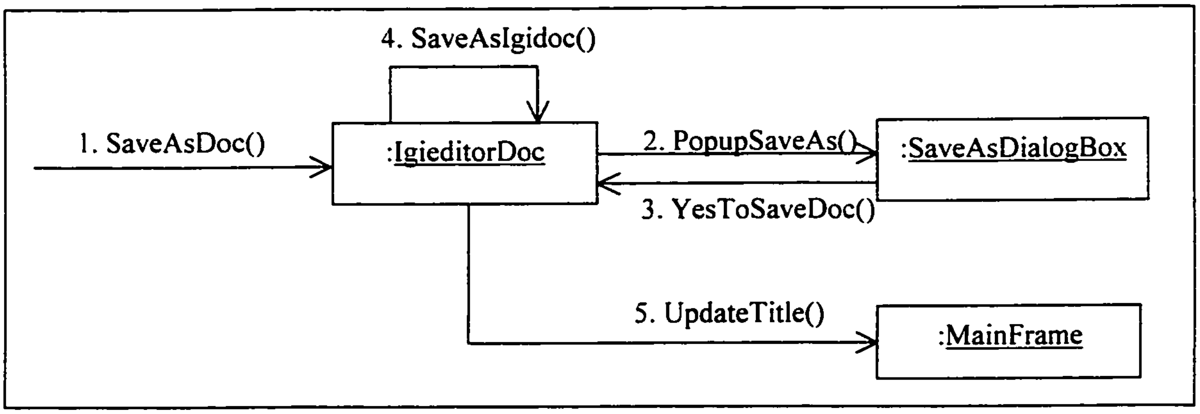


Figure 1-9 Collaboration diagram for use case 4

1.3.2.4.5.2 Sequence diagram

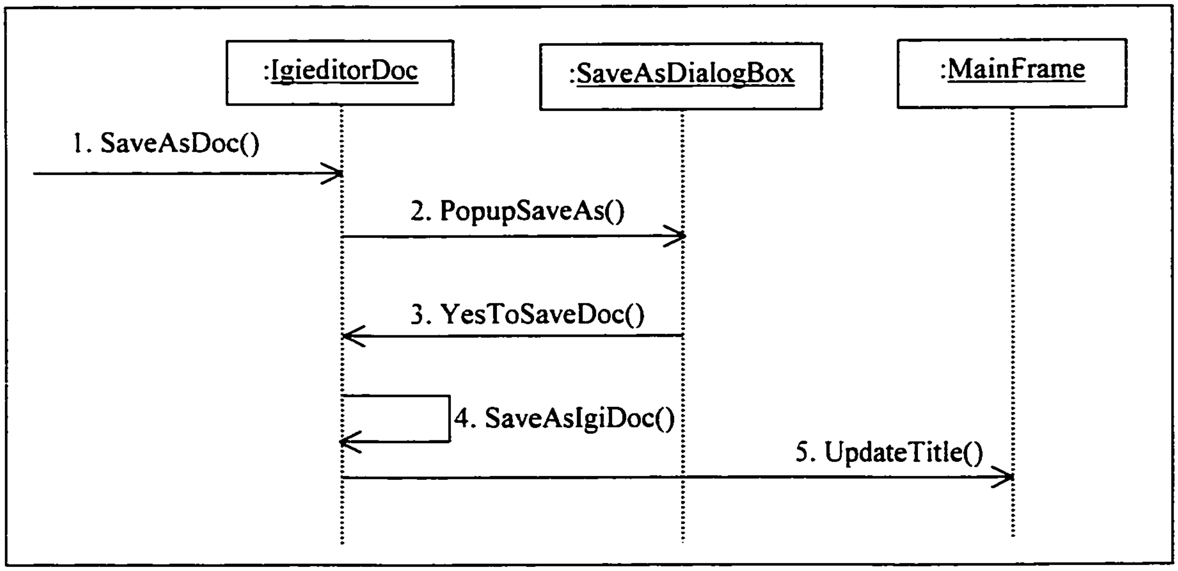


Figure 1-10 Sequence diagram for use case 4

1.3.2.5 Use case 5: User adds a primitive object to an IGI

1.3.2.5.1 Actor

An IGI Editor User.

1.3.2.5.2 Purpose

Actor selects a primitive element on the primitive palette and adds the primitive object in the edit area.

1.3.2.5.3 Preconditions

N/A.

1.3.2.5.4 Typical Course of Events

Table 1-8 Typical course of events of use case 5

Actor Action	System Response
1. Select a menu item from main menu "Primitives",	

Actor Action	System Response
or select a primitive from primitive palette.	
	2. Create a previous primitive list for undo purpose.
	3. Set the selected primitive as current primitive.
	4. Check the selected primitive palette button on the palette bar.
5. Select a desired position in the edit area.	
6. Drag the primitive object to the desired position.	
	7. Create a new primitive object.
	8. Set the new primitive object parameters according to the primitive object type, color, and position.
	9. Display the primitive object at the desired position.

Alternative Course

1. Line 3: if primitive Rectangle/Fill Rectangle is selected

   3.1. Click left mouse button on the desired position for the top left corner of the rectangle.

   3.2. Hold the left mouse button down and move the mouse to the desired position for the bottom right corner of the rectangle.

   3.3. Release left mouse button.

   3.4. Click left mouse button at another desired position for adding another rectangle, or click right mouse button to end adding rectangle action.

2. Line 3: if primitive Triangle/Fill Triangle is selected

   3.1. Click left mouse button on the desired position for the first point of the triangle.

   3.2. Release the left mouse button and move the mouse to the desired position for the second point of the triangle and click the left mouse button.

   3.3. Release the left mouse button and move the mouse to the desired position for the third point of the triangle and click the left mouse button.

   3.4. Release left mouse button.

   3.5. Click left mouse button at another desired position for adding another triangle, or click right mouse button to end adding triangle action.

3. Line 3: if primitive Ellipse/Fill Ellipse is selected

   3.1. Click left mouse button on the desired position for the top left corner of the ellipse.

   3.2. Hold the left mouse button down and move the mouse to the desired position for the bottom right corner of the ellipse.

   3.3. Release left mouse button.

   3.4. Click left mouse button at another desired position for adding another ellipse, or click right mouse button to end adding ellipse action.

4. Line 3: if primitive Text is selected

   3.1. A "Text Input" window pop up.

   3.2. Type the text in the edit box, and click OK button.

   3.3. Move mouse to the desired position

   3.4. Click left mouse button to add text at the position.

   3.5. Click left mouse button at another desired position for adding another text, or click right mouse button to end adding text action.

   Alternative Course

   - Line 3.2. Click on "Cancel" button to cancel adding text action.

5. Line 3: if primitive Tag is selected

   3.1. Move mouse to the desired position

   3.2. Click left mouse button to add tag at the position.

   3.3. Click right mouse button to end adding tag action.

   Alternative Course

   - Line 3.2. If a tag has existed already, a message window pops up with the message "Only add one tag name!". Click on "OK" button to kill the message window and end adding tag action.

   - Line 3.3. If click right mouse button, a message window pops up with the message "Only add one tag name!". Click on "OK" button to kill the message window and end adding tag action.

Note: The tag button on the palette is unselected after the message window disappears.

6. Line 3: if primitive process/logic/analog input/output stream is selected

    3.1. Move mouse to the desired position

    3.2. Click left mouse button to add a stream at the position.

    3.3. Click left mouse button at another desired position to add another stream.

    3.4. Click right mouse button on some where in the edit area to end adding stream action.

7. Line 3: if primitive line is selected

    3.1. Move mouse to the desired position

    3.2. Click left mouse button on the desired position for the start point of the line.

    3.3. Hold the left mouse button down and move the mouse to the desired position for the end point of the line.

    3.4. Release left mouse button.

    3.5. Click left mouse button at another desired position for adding another line. or click right mouse button to end adding line action.

8. Line 3: if primitive Arc/Pie is selected

    3.1. Click left mouse button on the desired position for the center point of the arc/pie.

    3.2. Release the left mouse button and move the mouse to the desired position for the start point of the arc/pie and click the left mouse button.

    3.3. Release the left mouse button and move the mouse to the desired position for the end point of the arc/pie and click the left mouse button.

    3.4. Release left mouse button.

    3.5. Click left mouse button at another desired position for adding another arc/pie, or click right mouse button to end adding arc/pie action.

1.3.2.5.5 Interaction Diagram for the main scenario

1.3.2.5.5.1 Collaboration diagram

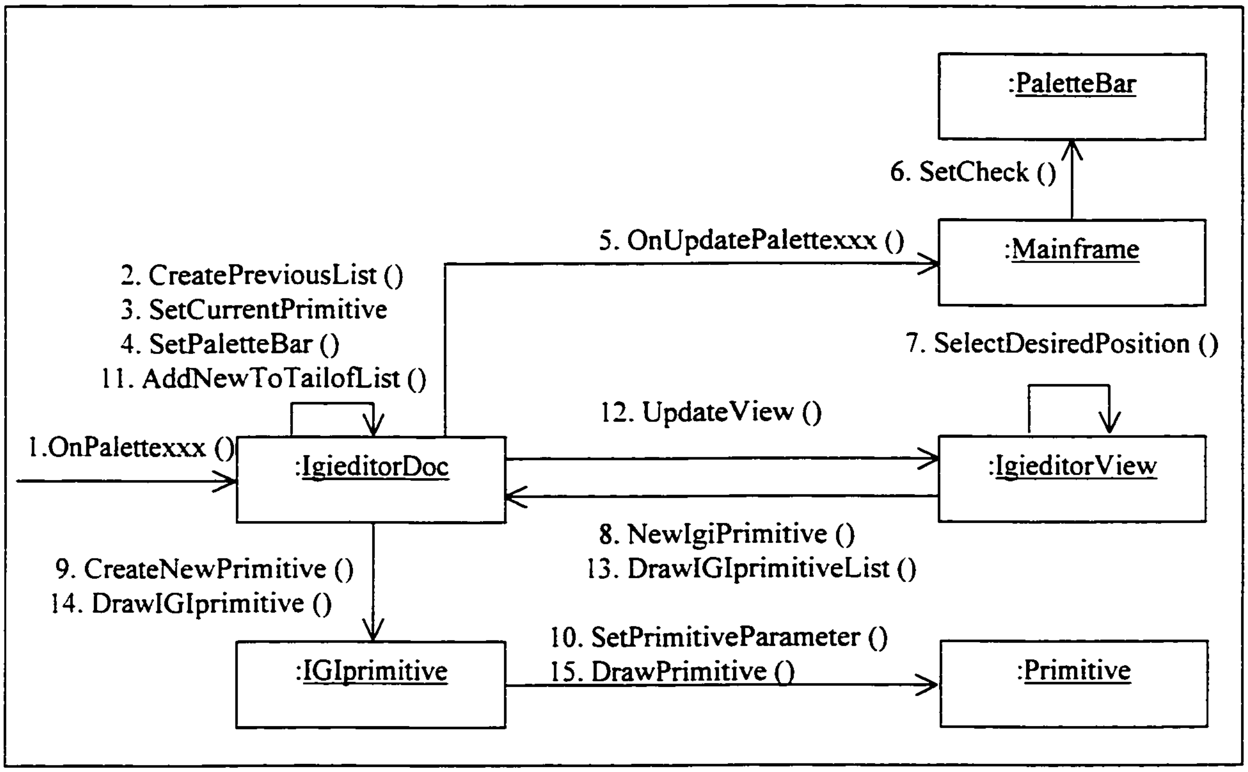


Figure 1-11 Collaboration diagram for use case 5

1.3.2.5.5.2 Sequence diagram

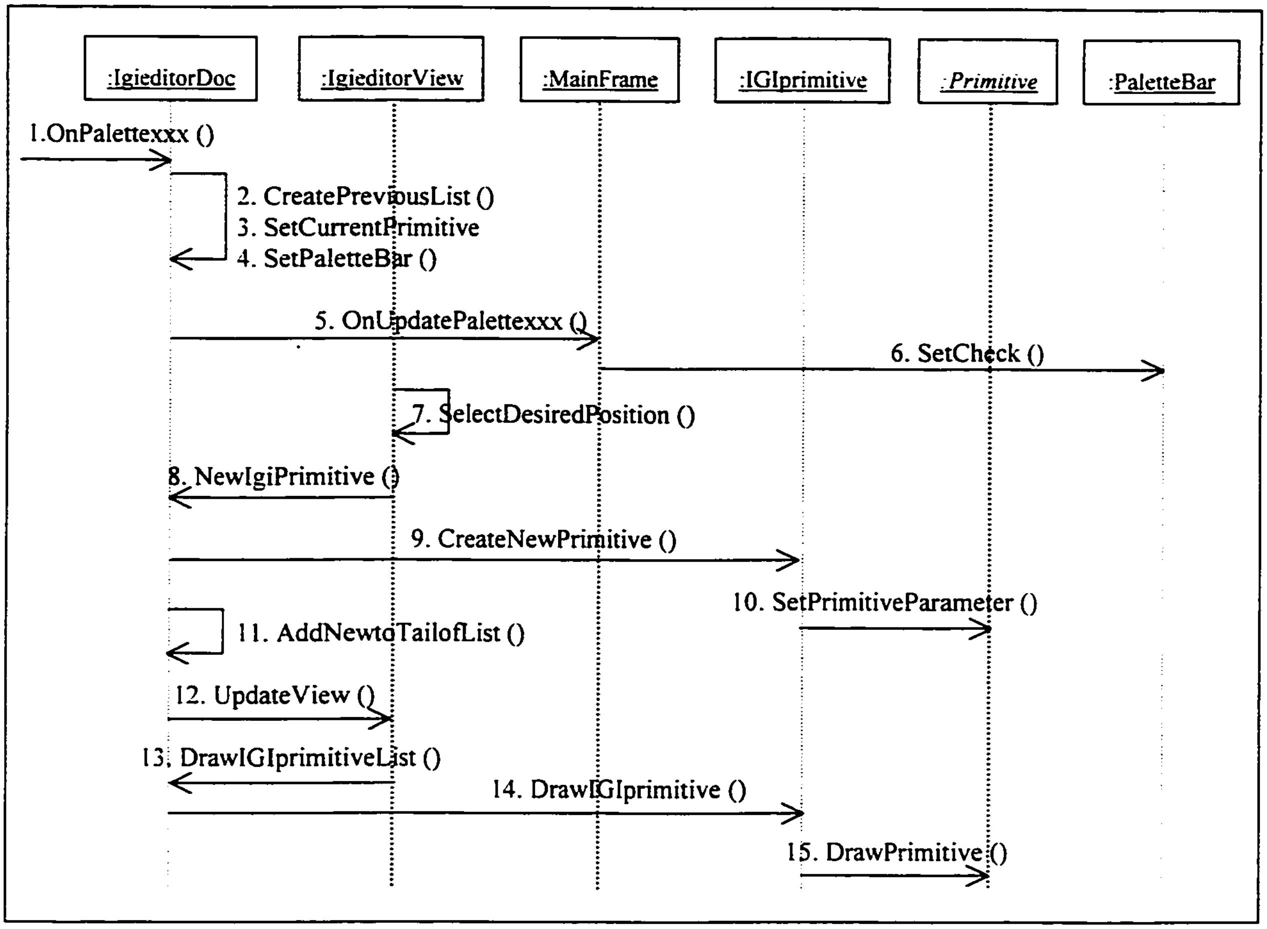


Figure 1-12 Sequence diagram for use case 5

1.3.2.6 Use case 6: User selects a primitive object

1.3.2.6.1 Actor

An IGI Editor User.

1.3.2.6.2 Purpose

Actor selects a primitive object for edit purpose. The primitive object edit actions include move, delete, copy, modify, and change color, push down/pop up. For stream, it can be rotated.

1.3.2.6.3 Preconditions

A primitive object exists in the edit area.

1.3.2.6.4 Typical Course of Events

Table 1-9 Typical course of events of use case 6

Actor Action	System Response
1. Move mouse over a primitive object.	
2. Click left mouse button.	
	3. Show a small square on the control points of the primitive object.
	4. The primitive object is ready for any edit action.

1.3.2.6.5 Interaction Diagram for the main scenario

1.3.2.6.5.1 Collaboration diagram

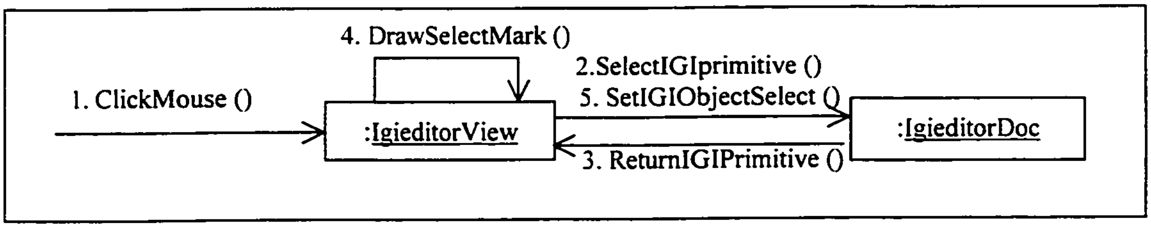


Figure 1-13 Collaboration diagram for use case 6

1.3.2.6.5.2 Sequence diagram

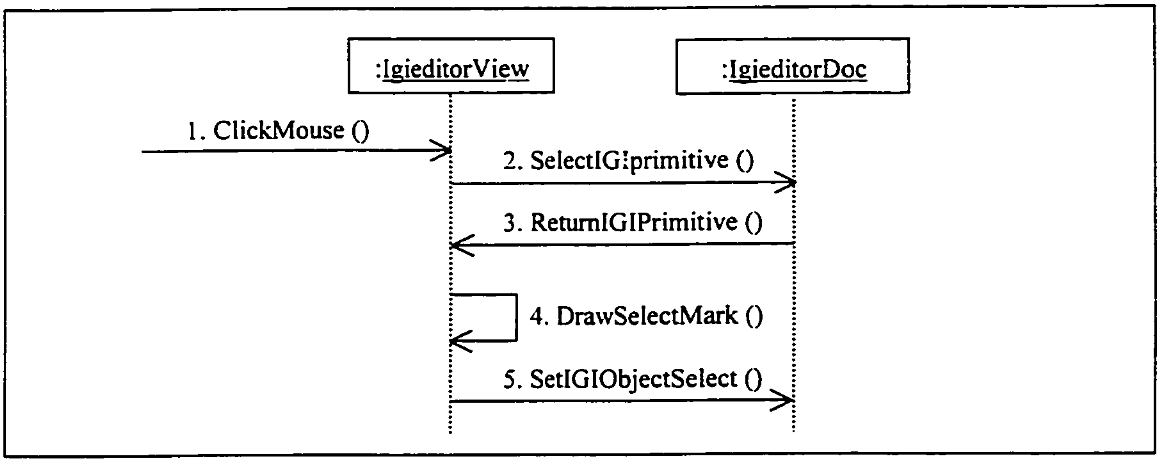


Figure 1-14 Sequence diagram for use case 6

1.3.2.7 Use case 7: User moves the selected primitive object(s)

1.3.2.7.1 Actor

An IGI Editor User.

1.3.2.7.2 Purpose

Actor moves the selected primitive object(s) from original position to another desired position.

1.3.2.7.3 Preconditions

At least a primitive object has been selected.

1.3.2.7.4 Typical Course of Events

Table 1-10 Typical course of events of use case 7

Actor Action	System Response
1. Move mouse over a selected primitive object.	
2. Click left mouse button.	
3. Hold left mouse button down.	
4. Move the mouse to the desired position.	
5. Release left mouse button.	
	6. Update IGI document.
	7. The primitive object is displayed on the desired position.

1.3.2.7.5 Interaction Diagram for the main scenario

1.3.2.7.5.1 Collaboration diagram

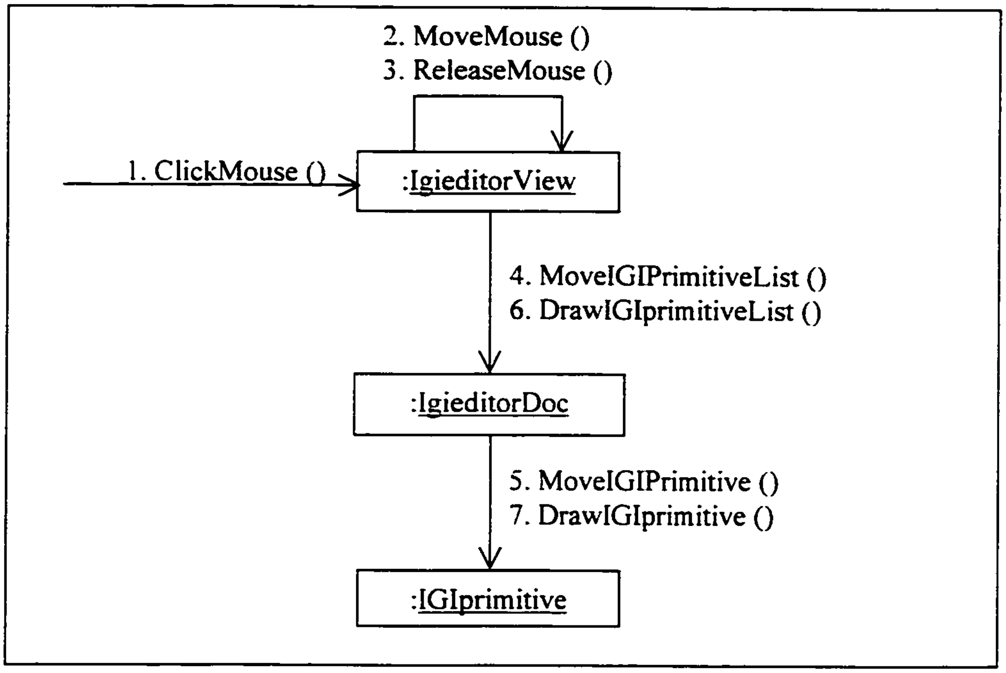


Figure 1-15 Collaboration diagram for use case 7

1.3.2.7.5.2 Sequence diagram

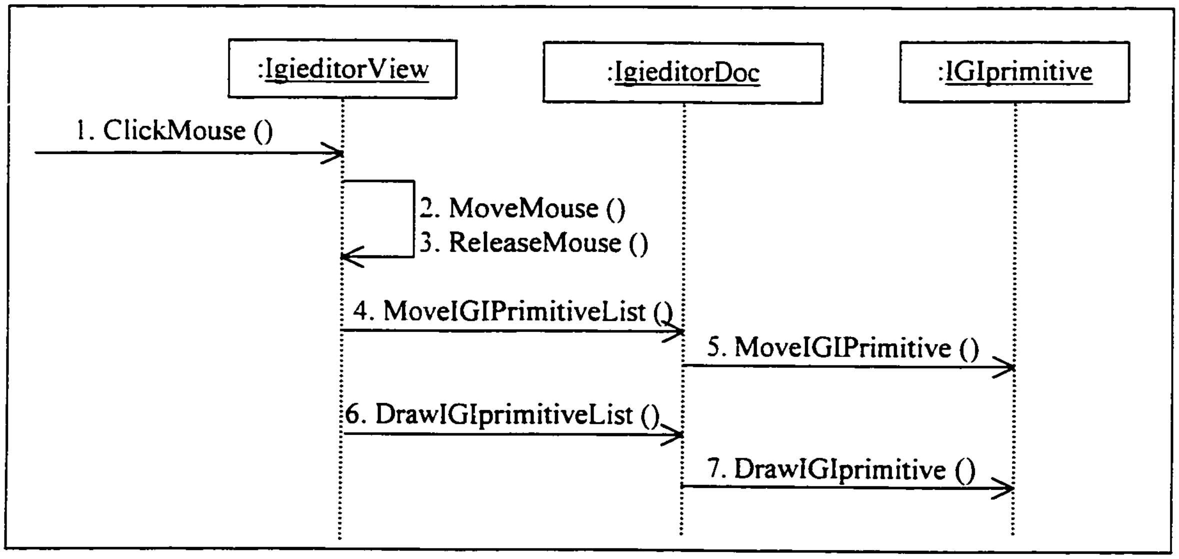


Figure 1-16 Sequence diagram for use case 7

1.3.2.8 Use case 8: User selects all primitive objects

1.3.2.8.1 Actor

An IGI Editor User.

1.3.2.8.2 Purpose

Actor selects all primitive objects in the edit area once.

1.3.2.8.3 PrecondiTions

Some primitive objects exist in the edit area.

1.3.2.8.4 Typical Course of Events

Table 1-11 Typical course of events of use case 8

Actor Action	System Response
1. Select "Select All" menu item, or click "Select All" toolbar button.	
	2. All primitive objects are selected. Display small squares on the control points of all primitive objects.

1.3.2.8.5 Interaction Diagram for the main scenario

1.3.2.8.5.1 Collaboration diagram

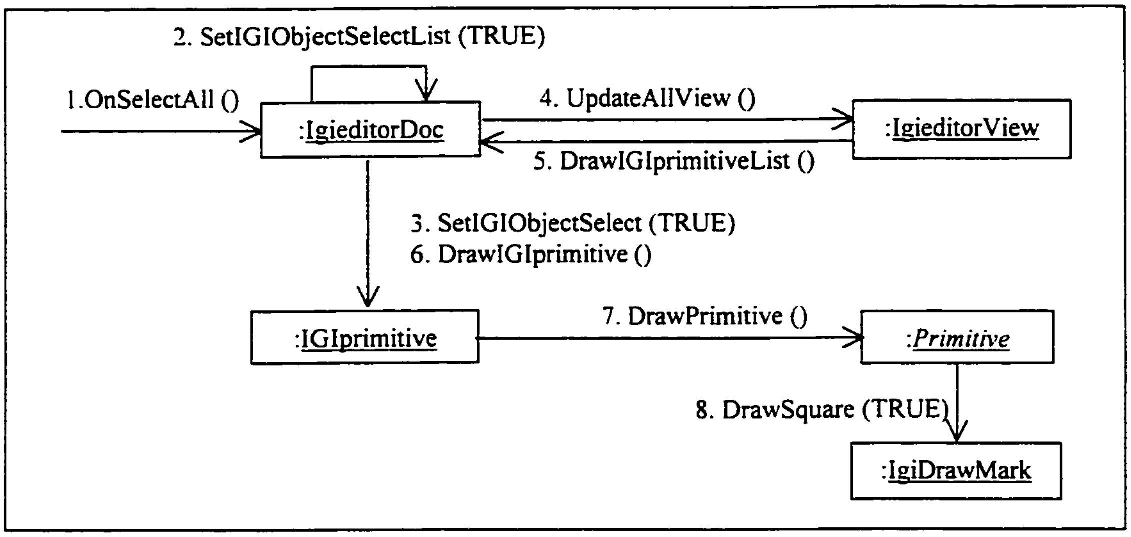


Figure 1-17 Collaboration diagram for use case 8

1.3.2.8.5.2 Sequence diagram

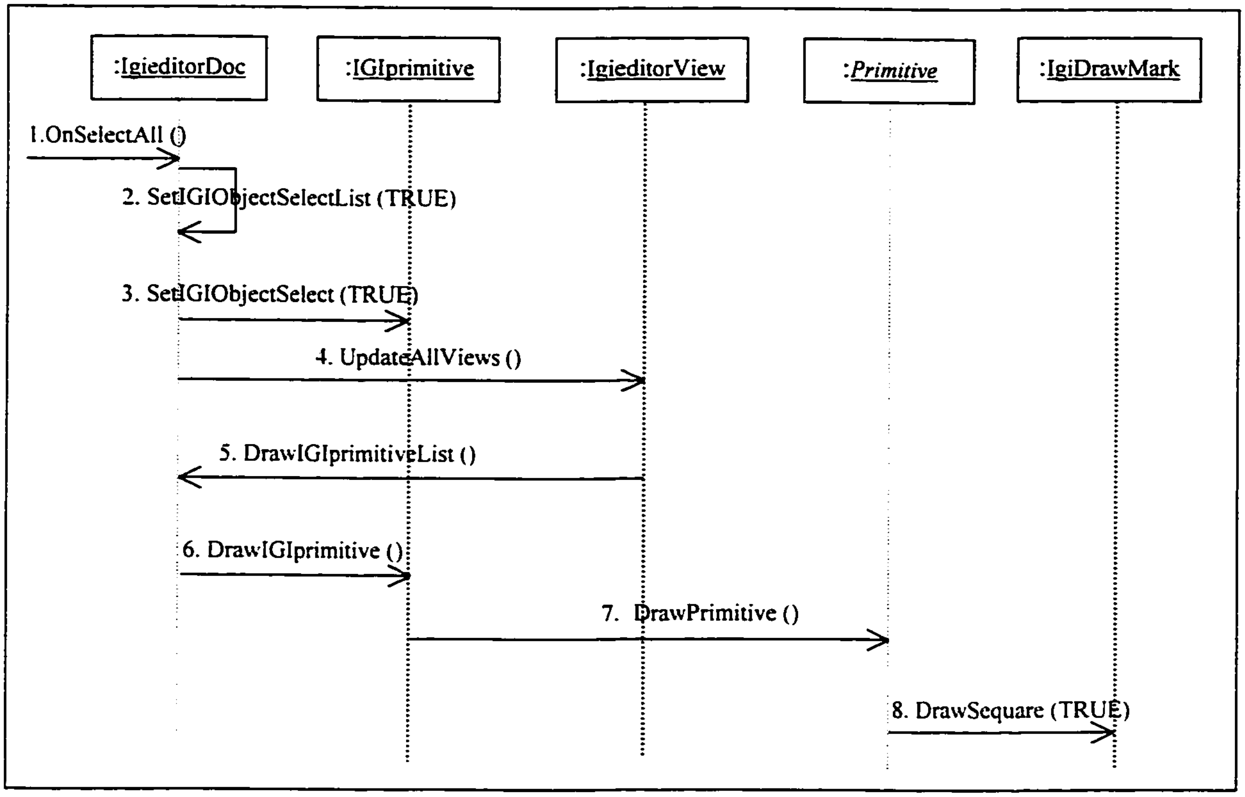


Figure 1-18 Sequence diagram for use case 8

1.3.2.9 Use case 9: User unselects all selected primitive object(s)

1.3.2.9.1 Actor

An IGI Editor User.

1.3.2.9.2 Purpose

Actor selects all primitive objects in the edit area once.

1.3.2.9.3 Preconditions

Some primitive objects have been selected.

1.3.2.9.4 Typical Course of Events

Table 1-12 Typical course of events of use case 9

Actor Action	System Response
1. Select "Unselect All" menu item, or click "Unselect All" toolbar button, or click left mouse button any where except on any primitive object in edit area.	
	2. All primitive objects are unselected. The small squares on the control points of the selected primitive objects are removed.

1.3.2.9.5 Interaction Diagram for the main scenario

1.3.2.9.5.1 Collaboration diagram

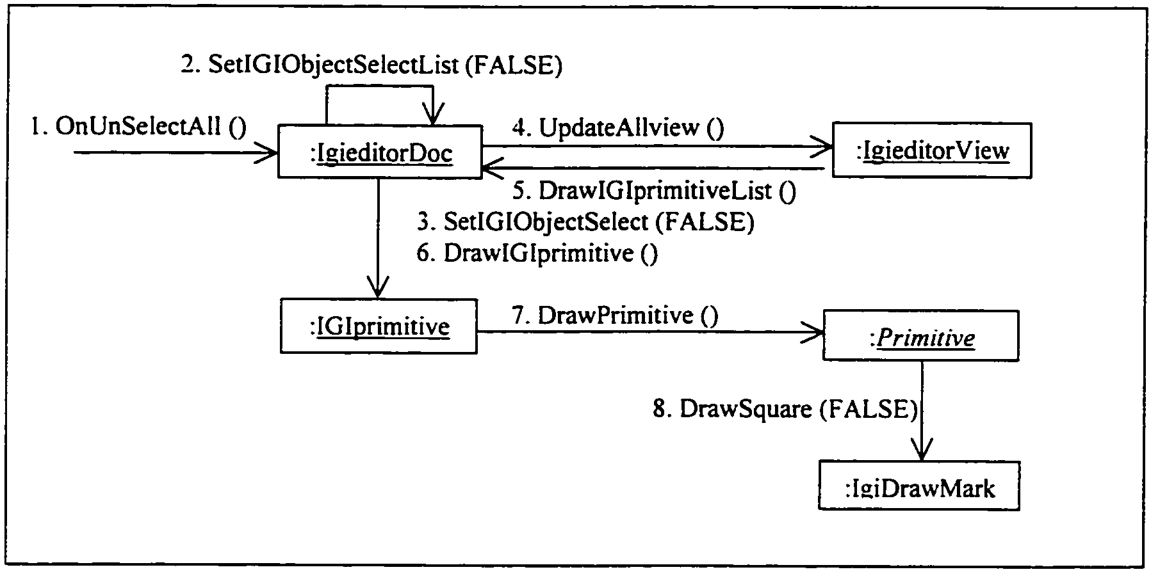


Figure 1-19 Collaboration diagram for use case 9

1.3.2.9.5.2 Sequence diagram

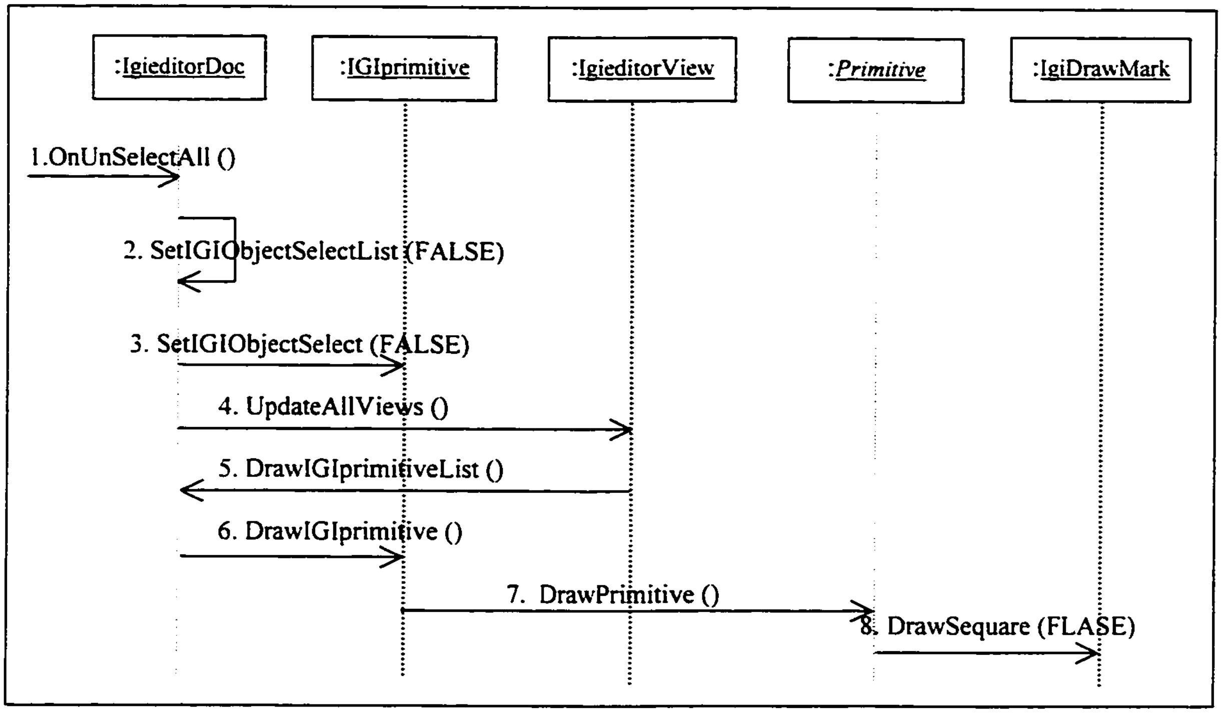


Figure 1-20 Sequence diagram for use case 9

1.3.2.10 Use case 10: User cuts any selected primitive objects

1.3.2.10.1 Actor

An IGI Editor User.

1.3.2.10.2 Purpose

Actor deletes all the selected primitive objects in the edit area.

1.3.2.10.3 Preconditions

Some primitive objects have been selected.

1.3.2.10.4 Typical Course of Events

Table 1-13 Typical course of events of use case 10

Actor Action	System Response
1. Select "Cut" menu item, or clicks "Cut" toolbar button, or presses "Delete" on keyboard.	
	2. Create a previous primitive object list.
	3. All the selected primitive objects are removed from the primitive object list.
	4. Redraw the primitive object list.

1.3.2.10.5 Interaction Diagram for the main scenario

1.3.2.10.5.1 Collaboration diagram

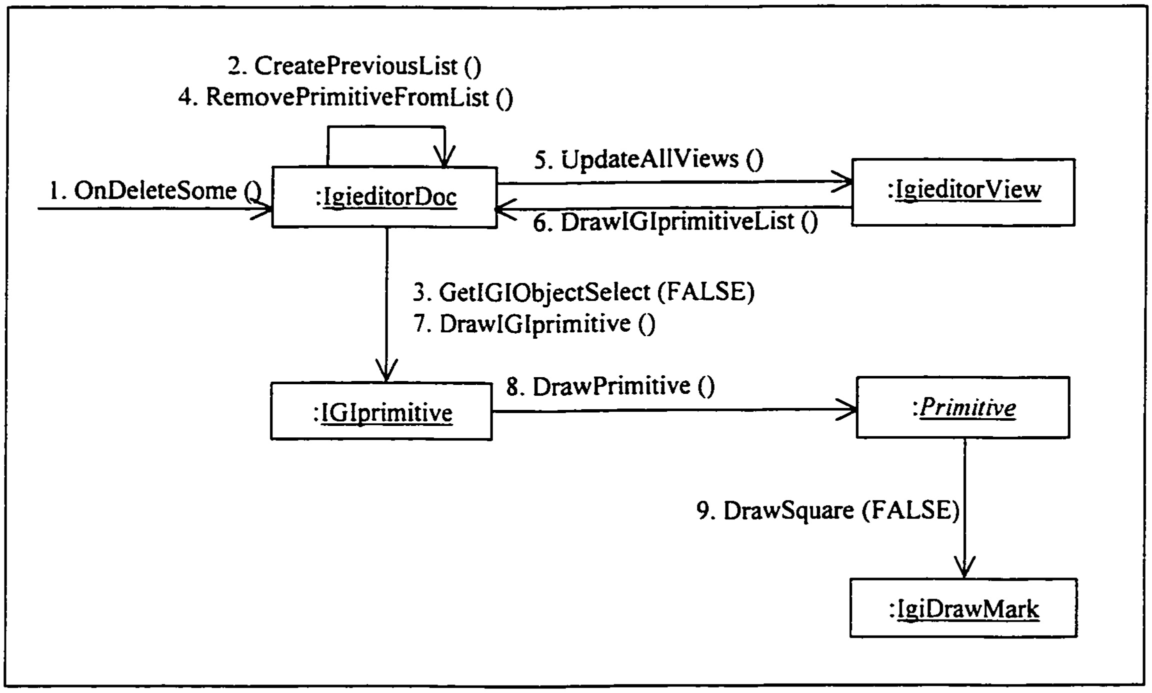


Figure 1-21 Collaboration diagram for use case 10

1.3.2.10.5.2 Sequence diagram

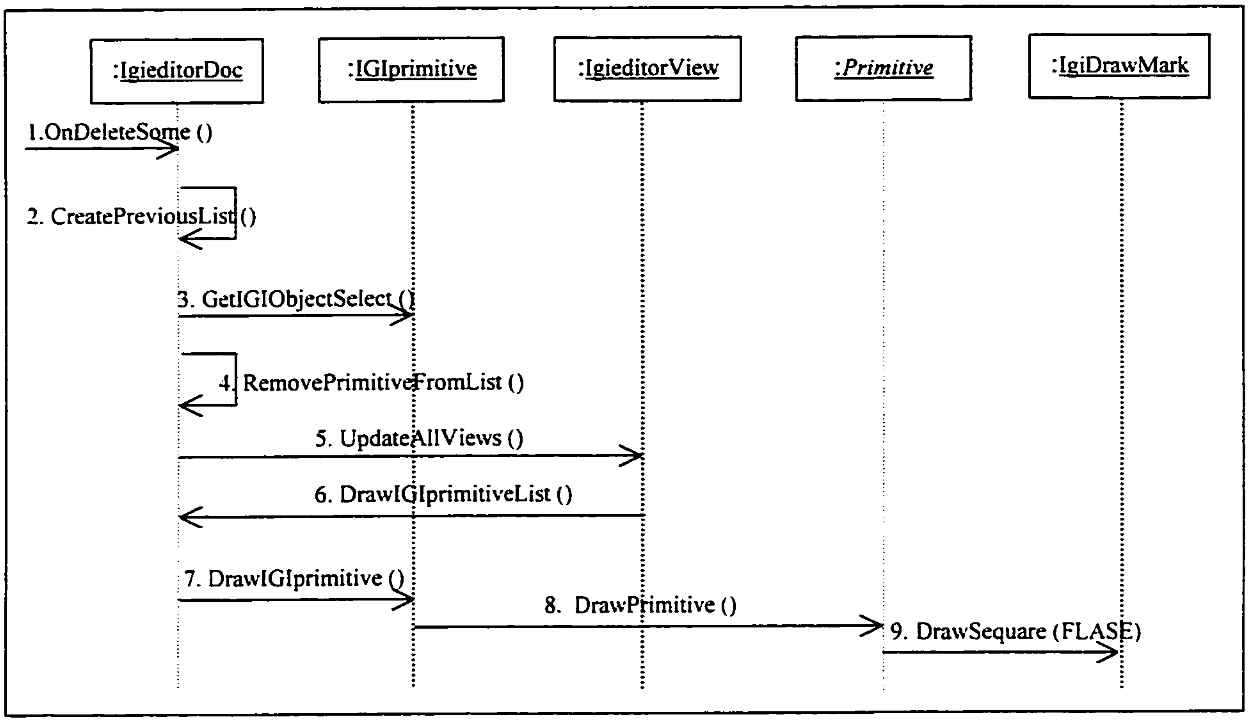


Figure 1-22 Sequence diagram for use case 10

1.3.2.11 Use case 11: User cuts all primitive objects

1.3.2.11.1 Actor

An IGI Editor User.

1.3.2.11.2 Purpose

Actor deletes all the primitive objects in the edit area.

1.3.2.11.3 Preconditions

Some primitive objects have been added in the edit area.

1.3.2.11.4 Typical Course of Events

Table 1-14 Typical course of events of use case 11

Actor Action	System Response
1. Select "Cut All" menu item, or clicks "Cut All" toolbar button.	
	2. Create a previous primitive object list.
	3. All the added primitive objects are deleted from the primitive object list.
	4. The edit area is refreshed.

1.3.2.11.5 Interaction Diagram for the main scenario

1.3.2.11.5.1 Collaboration diagram

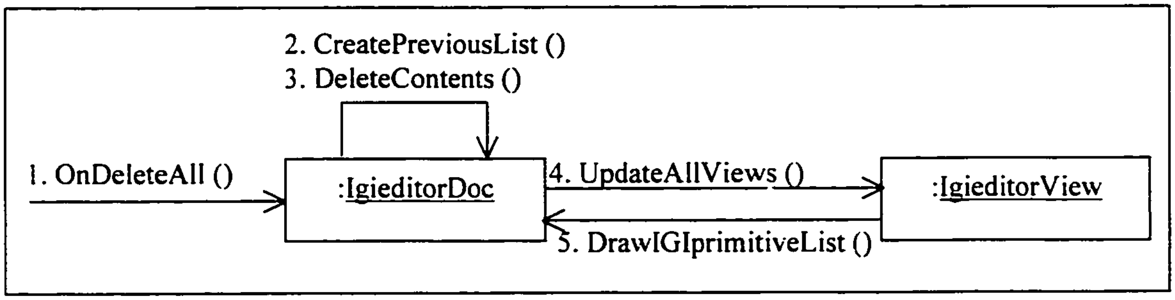


Figure 1-23 Collaboration diagram for use case 11

1.3.2.11.5.2 Sequence diagram

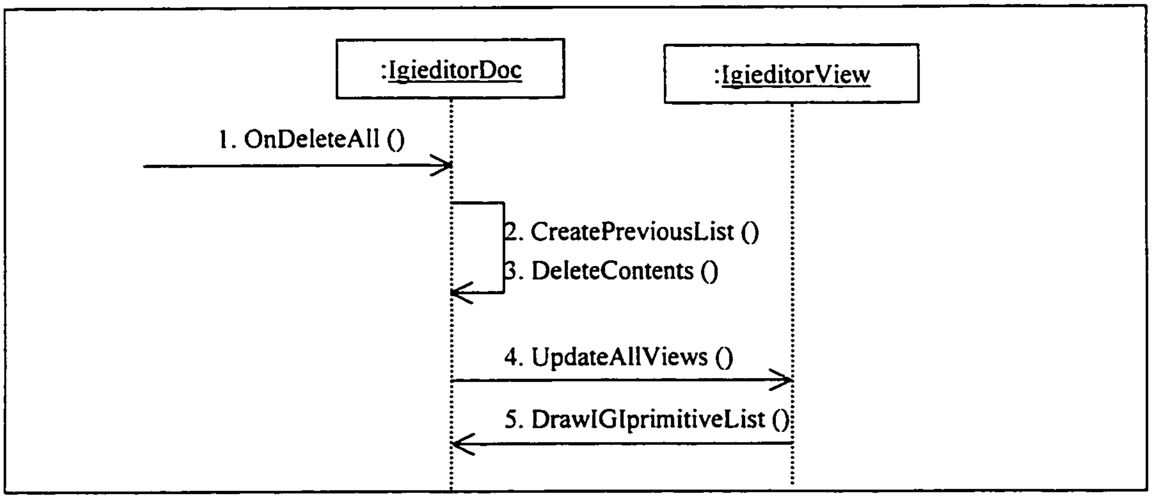


Figure 1-24 Sequence diagram for use case 11

1.3.2.12 Use case 12: User undoes the last edit action

1.3.2.12.1 Actor

An IGI Editor User.

1.3.2.12.2 Purpose

Actor cancels the last edit action.

1.3.2.12.3 Preconditions

An edit action is just completed.

1.3.2.12.4 Typical Course of Events

Table 1-15 Typical course of events of use case 12

Actor Action	System Response
1. Select "Undo" menu item, or clicks "Undo" toolbar button.	
	2. Check the IGI document is modified?
	3. Swap the current primitive object list with the previous primitive object list.
	4. Redraw the primitive object list on the edit area.

1.3.2.12.5 Interaction Diagram for the main scenario

1.3.2.12.5.1 Collaboration diagram

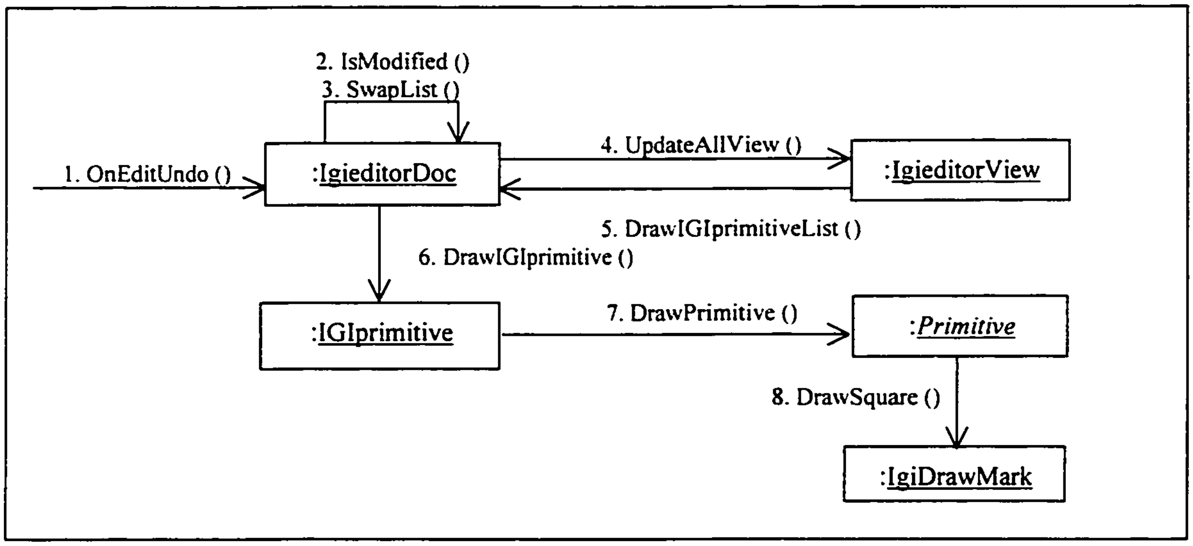


Figure 1-25 Collaboration diagram for use case 12

1.3.2.12.5.2 Sequence diagram

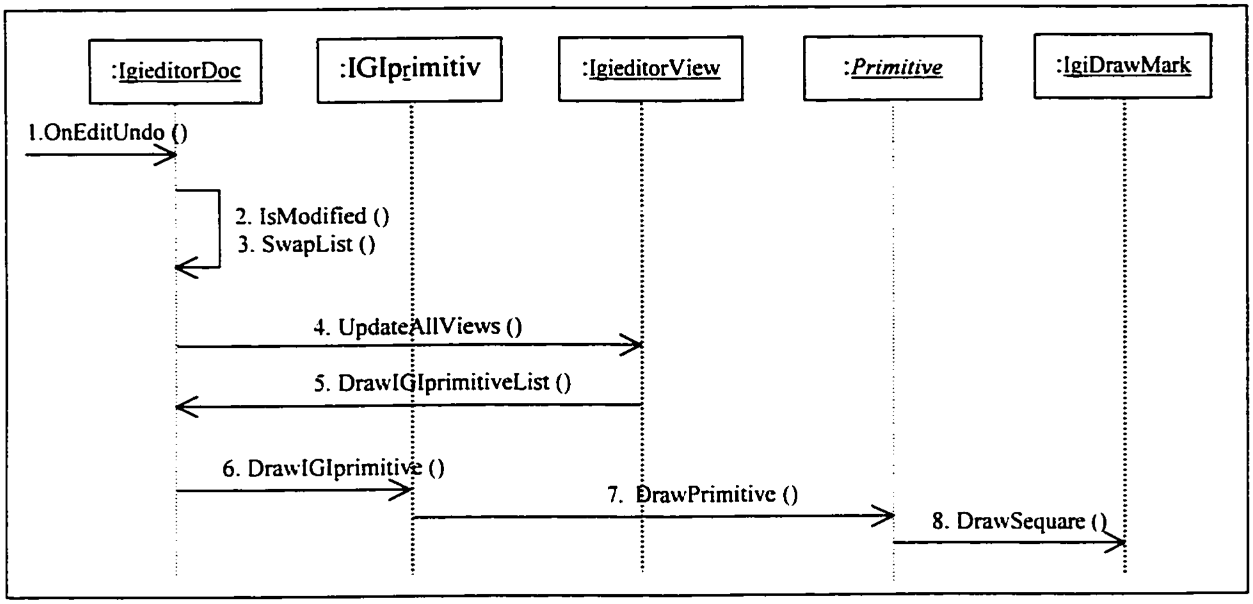


Figure 1-26 Sequence diagram for use case 12

1.3.2.13 Use case 13: User pushes primitive object down

1.3.2.13.1 Actor

An IGI Editor User.

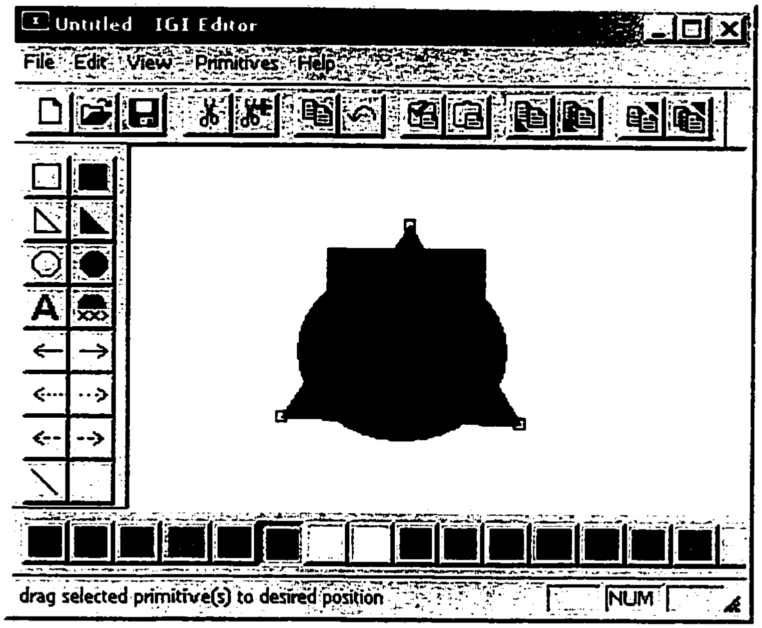
1.3.2.13.2 Purpsoe

Actor pushes a selected primitive object down to next level.

An icon is consisted of different primitive objects. The first added object is at lowest (bottom) level and the last added object is at highest (top) level. After adding the primitive objects, user can change the objects level with edit command "Push Down", "Push to Bottom", "Pop Up" and "Pop to Top".

Here is an example of pushing a selected object one level down in the edited icon.

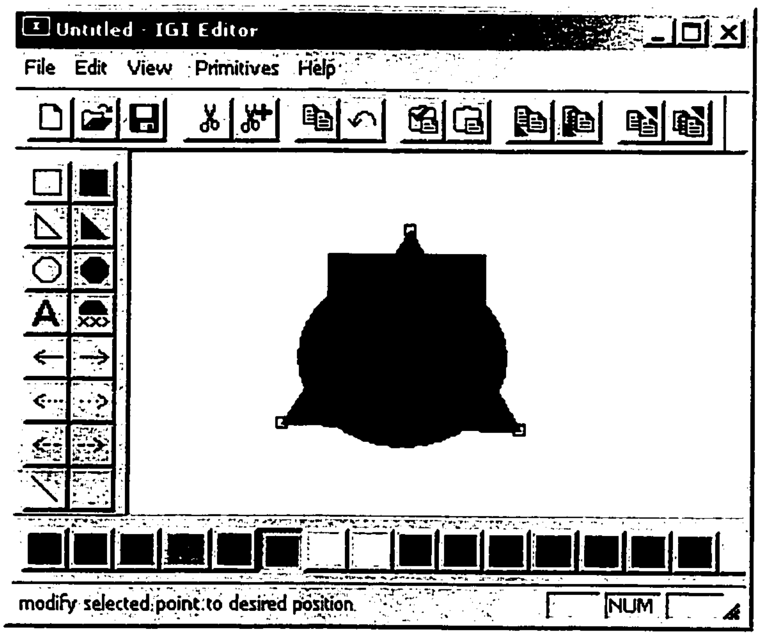
In figure 1-27, before user clicks a "Push Down" command button, a selected triangle is on the top level. A circle in red color is in the middle level. A rectangle in magenta color is in the bottom level.



drag selected primitive(s) to desired position

Figure 1-27 Before an object push down

In figure 1-28, after user clicks on the "Push Down" command button, the selected triangle has down to middle level and the circle object has changed to the top level.



modify selected point to desired position

Figure 1-28 After an object push down

1.3.2.13.3 Preconditions

A primitive object has been selected.

1.3.2.13.4 Typical Course of Events

Table 1-16 Typical course of events of use case 13

Actor Action	System Response
1. Select "Push Down" menu item, or clicks "Push Down" toolbar button.	
	2. Get the selected primitive object position in the primitive object list.
	3. Insert the selected primitive object before its previous object.
	4. Remove the selected primitive object at its current position
	5. Redraw the primitive object list.

Alternative Course

1. Line 2: If the selected primitive object is only primitive object in the edit area or the primitive object is at bottom level already, system does nothing.

1.3.2.13.5 Interaction Diagram for the main scenario

1.3.2.13.5.1 Collaboration diagram

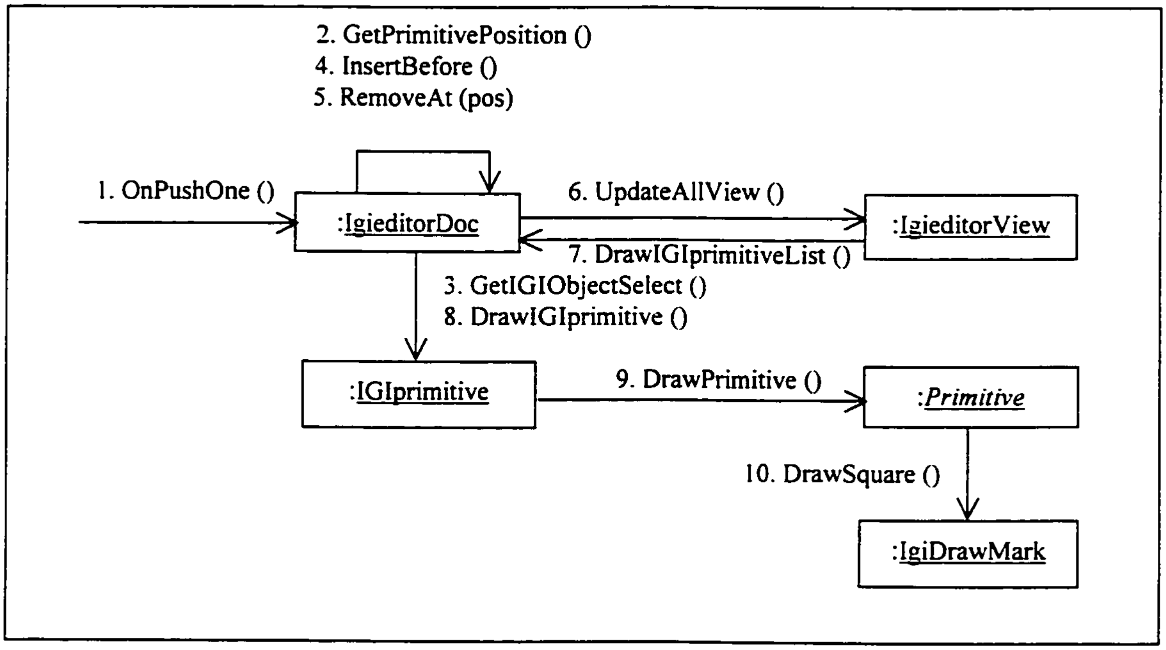


Figure 1-29 Collaboration diagram for use case 13

1.3.2.13.5.2 Sequence diagram

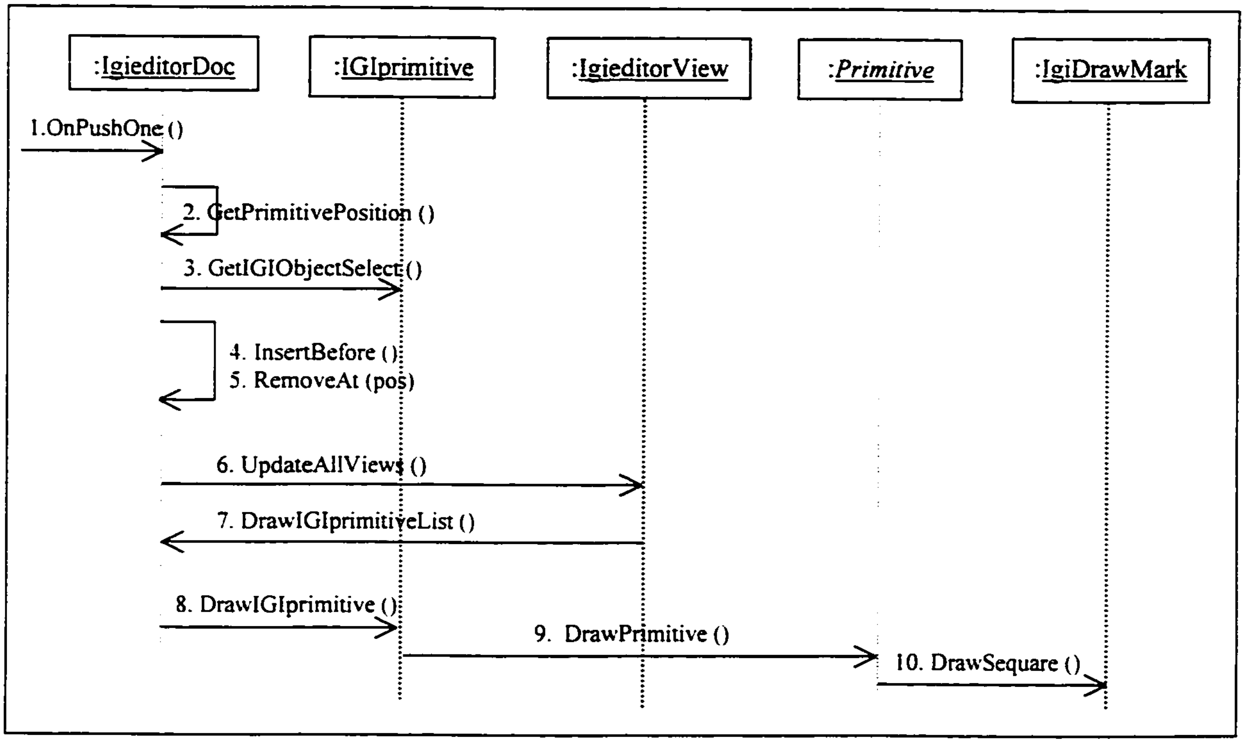


Figure 1-30 Sequence diagram for use case 13

1.3.2.14 Use case 14: User pushes primitive object down to bottom

1.3.2.14.1 Actor

An IGI Editor User.

1.3.2.14.2 Purpose

Actor pushes a selected primitive object down to bottom level in the edit area.

1.3.2.14.3 Preconditions

A primitive object has been selected.

1.3.2.14.4 Typical Course of Events

Table 1-17 Typical course of events of use case 14

Actor Action	System Response
1. Select "Push To Bottom" menu item, or clicks "Push To Bottom" toolbar button.	

Actor Action	System Response
	2. Get the selected primitive object position in the primitive object list.
	3. Add the selected primitive object at the head of primitive object list.
	4. Remove the selected primitive object at its current position
	5. Redraw the primitive object list.

Alternative Course

Line 2: If the selected primitive object is only primitive object in the edit area or the primitive object is at bottom level already, system does nothing.

1.3.2.14.5 Interaction Diagram for the main scenario

1.3.2.14.5.1 Collaboration diagram

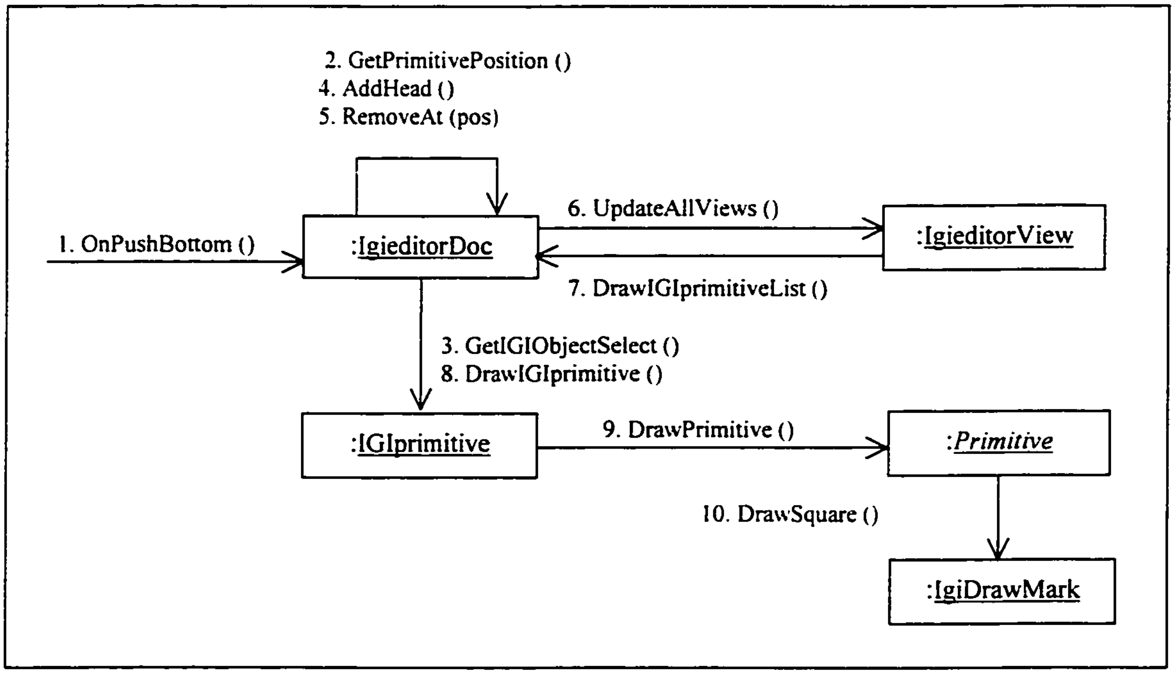


Figure 1-31 Collaboration diagram for use case 14

1.3.2.14.5.2 Sequence diagram

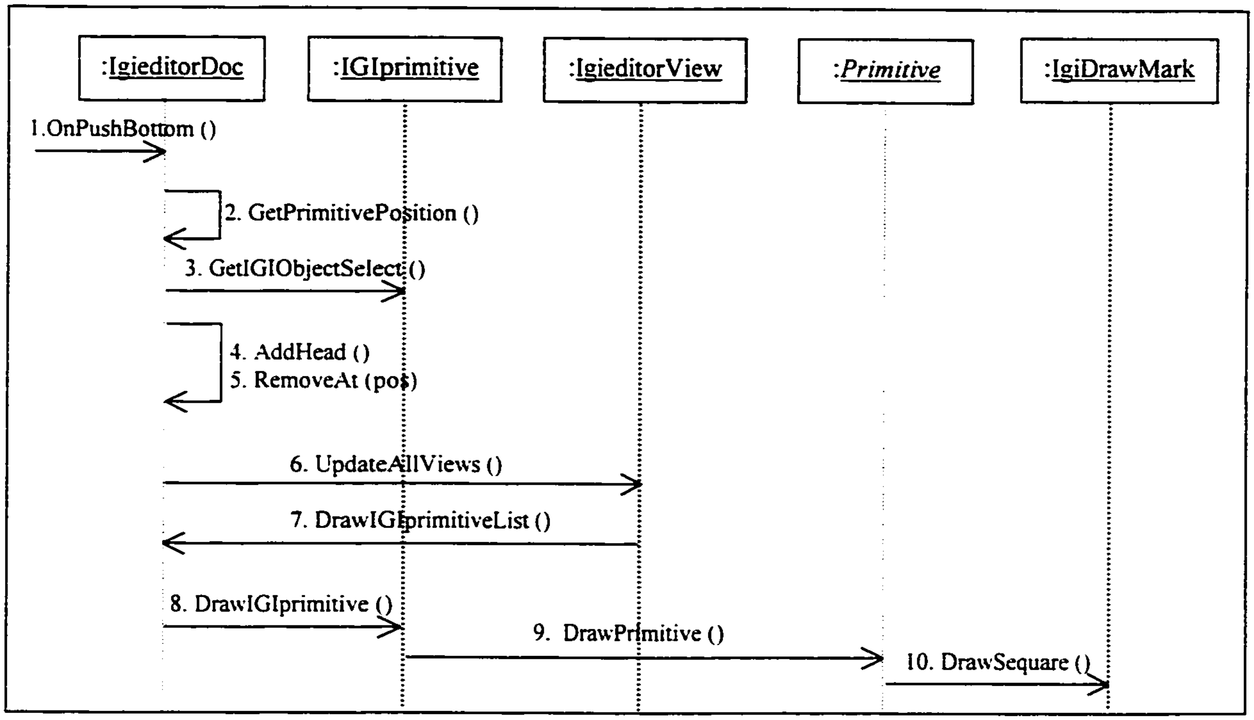


Figure 1-32 Sequence diagram for use case 14

1.3.2.15 Use case 15: User pops a primitive object up

1.3.2.15.1 Actor

An IGI Editor User.

1.3.2.15.2 Purpose

Actor pops a selected primitive object up to next level.

1.3.2.15.3 Preconditions

A primitive object has been selected.

1.3.2.15.4 Typical Course of Events

Table 1-18 Typical course of events of use case 15

Actor Action	System Response
1. Select "Pop Up" menu item, or clicks "Pop Up" toolbar button.	

Actor Action	System Response
	2. Get the selected primitive object position in the primitive object list.
	3. Insert the selected primitive object after its next primitive object.
	4. Remove the selected primitive object at its current position
	5. Redraw the primitive object list.

Alternative Course

Line 2: If the selected primitive object is only primitive object in the edit area or the primitive object is at top level already, system does nothing.

1.3.2.15.5 Interaction Diagram for the main scenario

1.3.2.15.5.1 Collaboration diagram

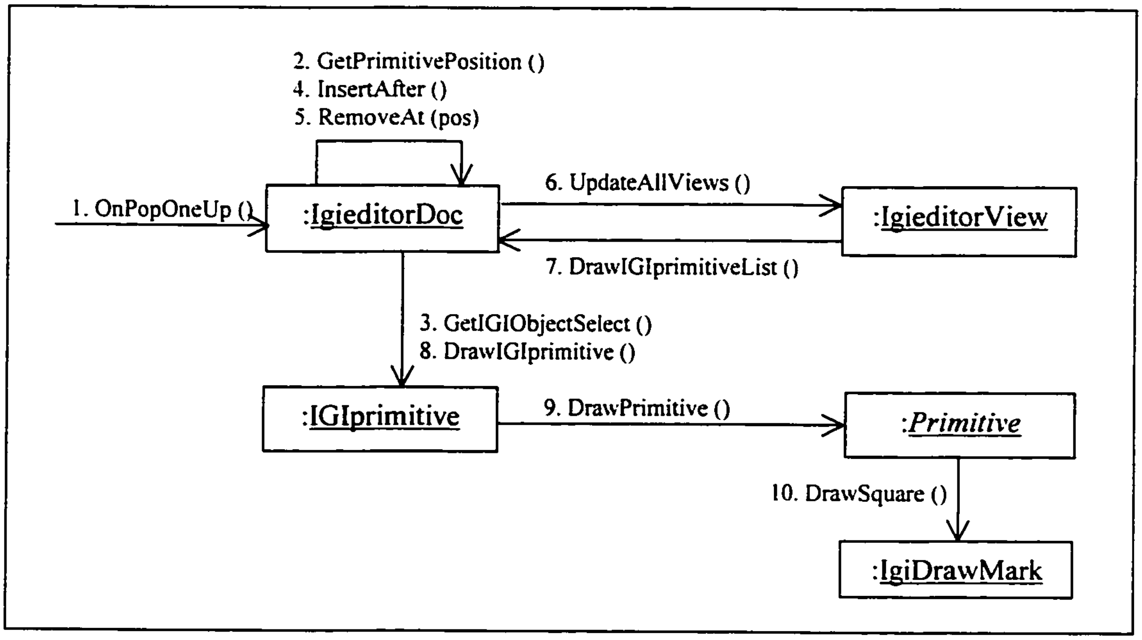


Figure 1-33 Collaboration diagram for use case 15

1.3.2.15.5.2 Sequence diagram

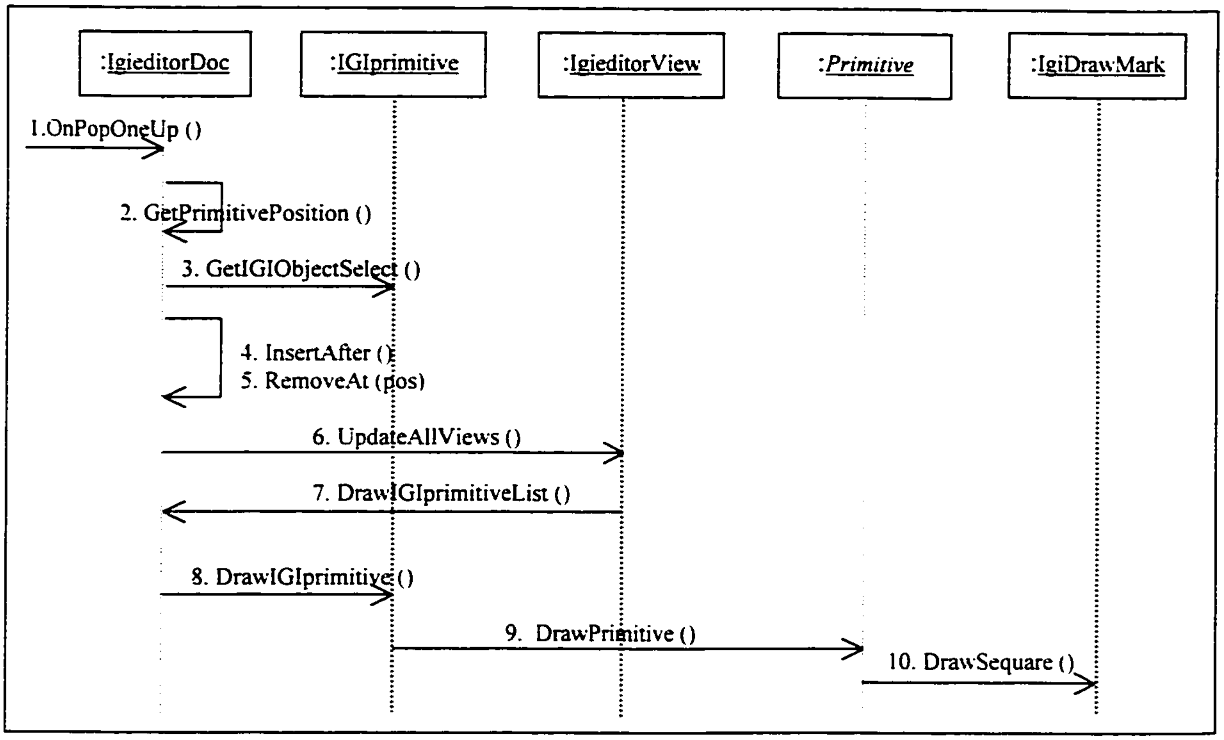


Figure 1-34 Sequence diagram for use case 15

1.3.2.16 Use case 16: User pops primitive object up to top

1.3.2.16.1 Actor

An IGI Editor User.

1.3.2.16.2 Purpose

Actor pops a selected primitive object up to top level in the edit area.

1.3.2.16.3 Preconditions

A primitive object has been selected.

1.3.2.16.4 Typical Course of Events

Table 1-19 Typical course of events of use case 16

Actor Action	System Response
1. Select "Pop To top" menu item, or clicks "Pop to Top" toolbar button.	

Actor Action	System Response
	2. Get the selected primitive object position in the primitive object list.
	3. Add the selected primitive object at the tail of the primitive object list.
	4. Remove the selected primitive object at its current position
	5. Redraw the primitive object list.

Alternative Course

Line 2: If the selected primitive object is only primitive object in the edit area or the primitive object is at top level already, system does nothing.

1.3.2.16.5 Interaction Diagram for the main scenario

1.3.2.16.5.1 Collaboration diagram

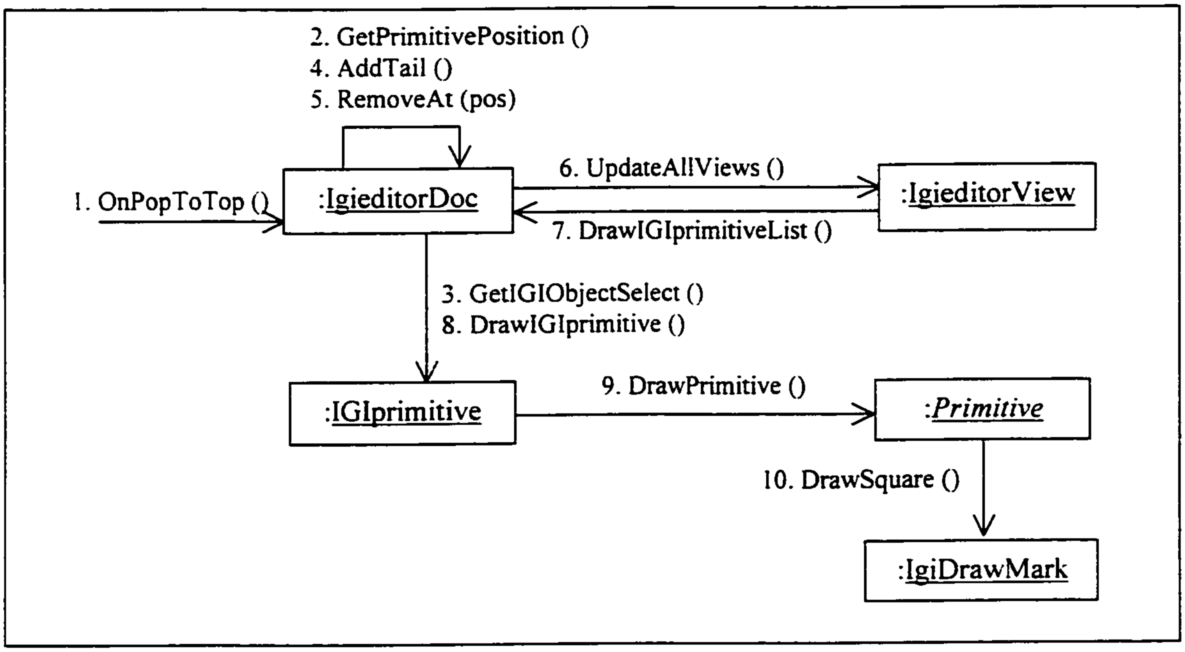


Figure 1-35 Collaboration diagram for use case 16

1.3.2.16.5.2 Sequence diagram

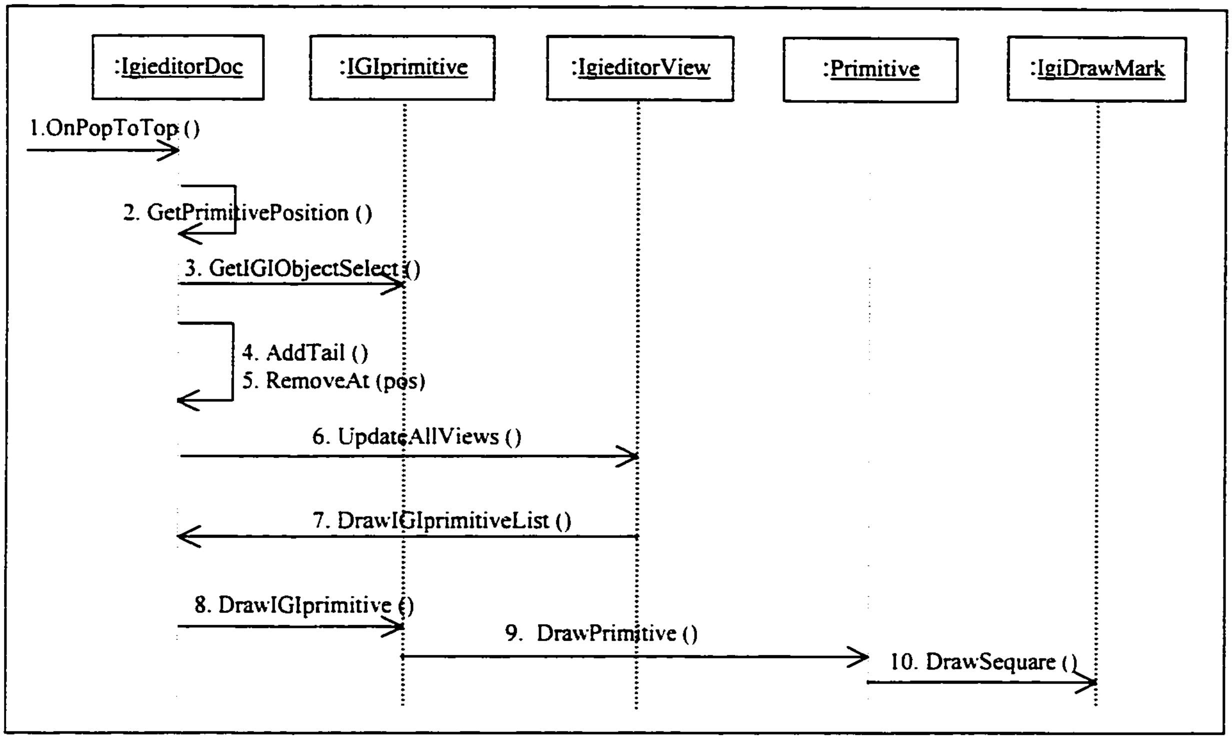


Figure 1-36 Sequence diagram for use case 16

1.3.2.17 Use case 17: User copies any selected primitive object(s)

1.3.2.17.1 Actor

An IGI Editor User.

1.3.2.17.2 Purpose

Actor makes a copy for all of the selected primitive objects.

1.3.2.17.3 Preconditions

Some primitive objects have been selected.

1.3.2.17.4 Typical Course of Events

Table 1-20 Typical course of events of use case 17

Actor Action	System Response
1. Select "Copy" menu item, or clicks "Copy" toolbar button	

Actor Action	System Response
	2. Create the previous primitive object list.
	3. Get the selected primitive object position in the primitive object list.
	4. Create a new primitive object.
	5. Copy the parameters of the selected primitive object to the new primitive object.
	6. Add the new primitive object to the tail of the primitive object list.
	7. Redraw the primitive object list.

Alternative Course

Line 2: If the selected primitive object is tag name, system does nothing.

1.3.2.17.5 Interaction Diagram for the main scenario

1.3.2.17.5.1 Collaboration diagram

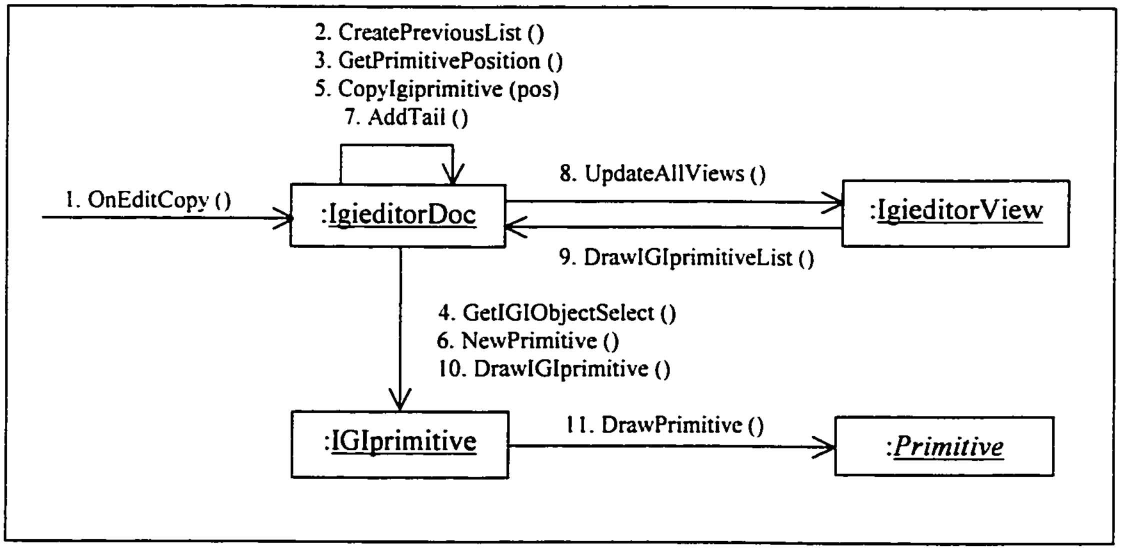


Figure 1-37 Collaboration diagram for use case 17

1.3.2.17.5.2 Sequence diagram

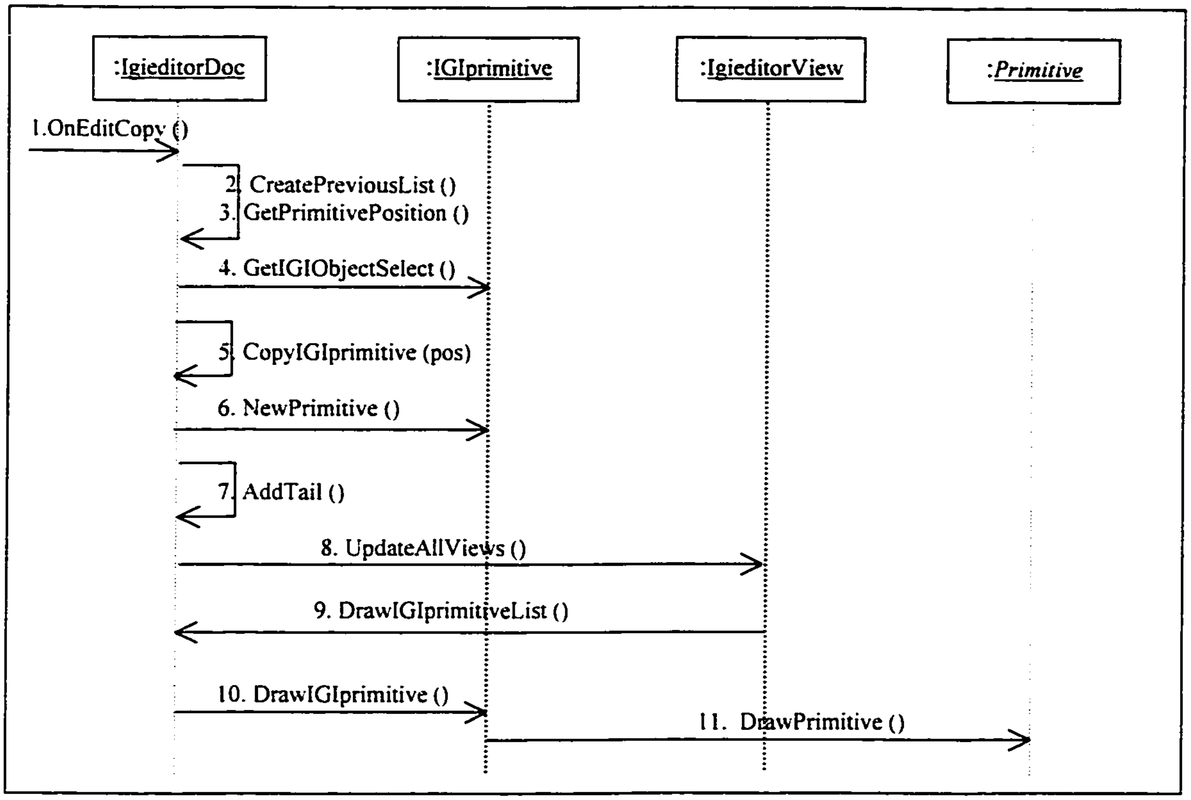


Figure 1-38 Sequence diagram for use case 17

1.3.2.18 Use case 18: User changes any selected primitive object(s) color

1.3.2.18.1 Actor

An IGI Editor User.

1.3.2.18.2 Pursose

Actor changes the display color for the selected primitive objects.

1.3.2.18.3 Preconditions

Some primitive objects have been selected.

1.3.2.18.4 Typical Course of Events

Table 1-21 Typical course of events of use case 18

Actor Action	System Response
1. Select one desired color on the color palette.	
	2. Create a previous primitive object list.
	3. Set current color to the selected color.
	4. Check the selected color button on the color bar.
	5. Update the selected primitive object color to the current color.
	6. Redraw the primitive object list.

1.3.2.18.5 Interaction Diagram for the main scenario

1.3.2.18.5.1 Collaboration diagram

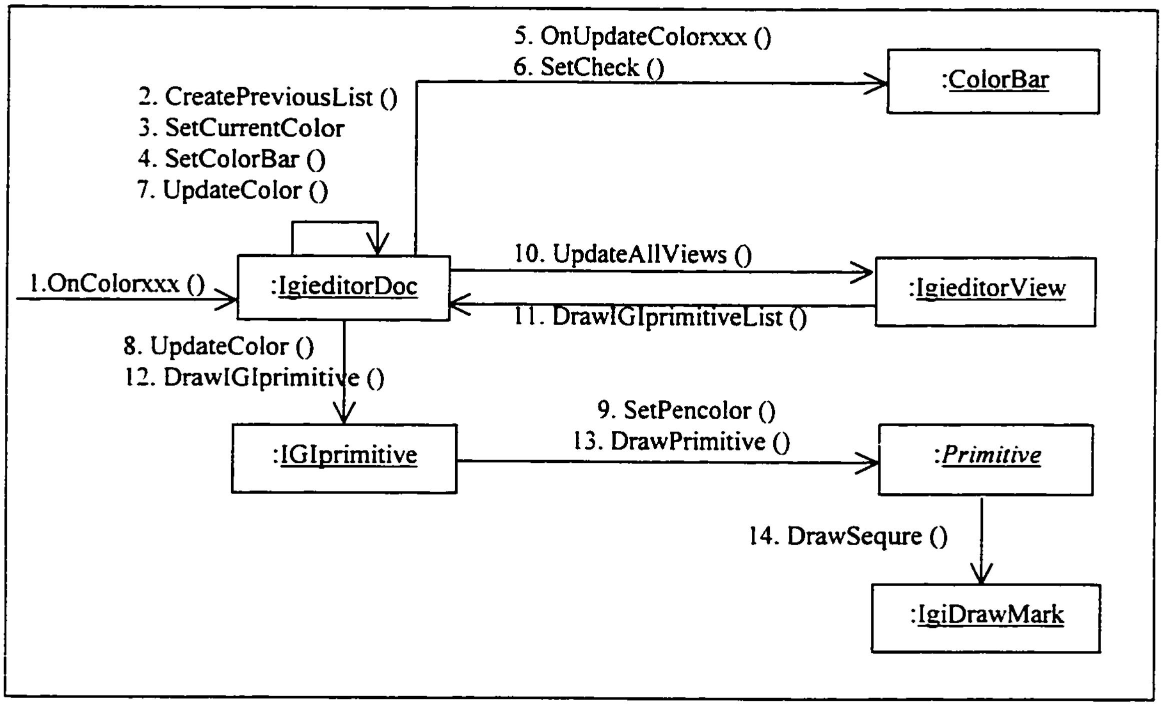


Figure 1-39 Collaboration diagram for use case 18

1.3.2.18.5.2 Sequence diagram

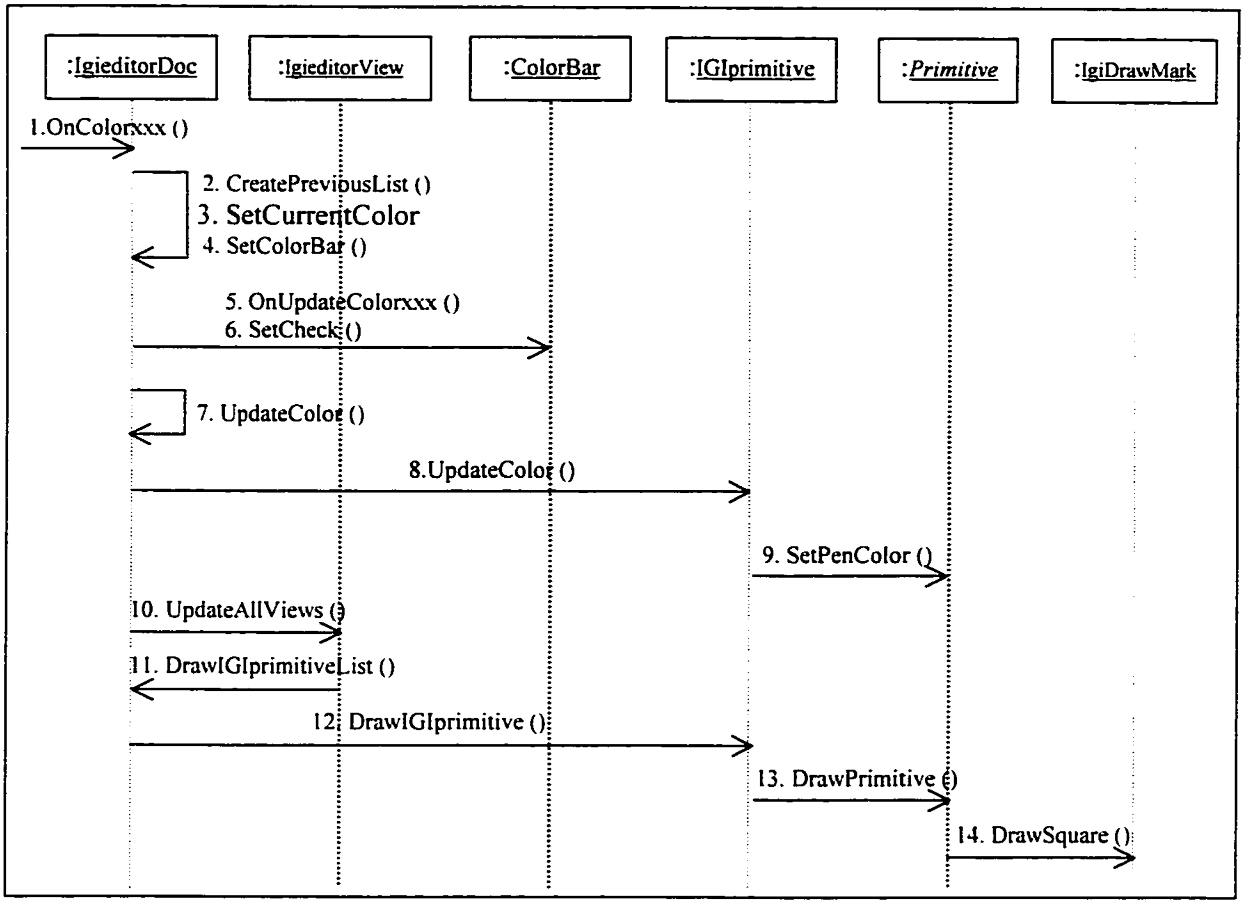


Figure 1-40 Sequence diagram for use case 18

1.3.2.19 Use case 19: User rotates a selected stream object

1.3.2.19.1 Actor

An IGI Editor User.

1.3.2.19.2 Pursose

Actor changes the orientation of a last selected stream object.

1.3.2.19.3 Preconditions

Selecting a stream object was the last edit action.

1.3.2.19.4 Typical Course of Events

Table 1-22 Typical course of events of use case 19

Actor Action	System Response
1. Move mouse to edit area.	
2. Click right mouse button.	
	3. Get the selected stream object's orientation.
	4. Create a previous primitive object list.
	5. Increase the selected stream object's orientation.
	6. Redraw the primitive object list.

1.3.2.19.5 Interaction Diagram for the main scenario

1.3.2.19.5.1 Collaboration diagram

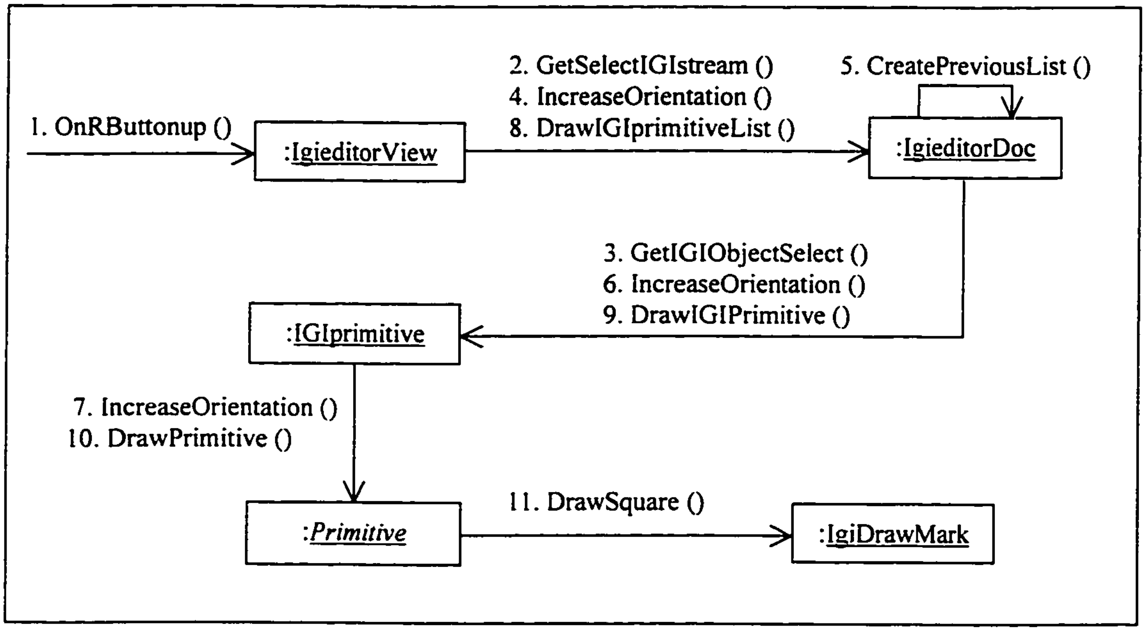


Figure 1-41 Collaboration diagram for use case 19

1.3.2.19.5.2 Sequence diagram

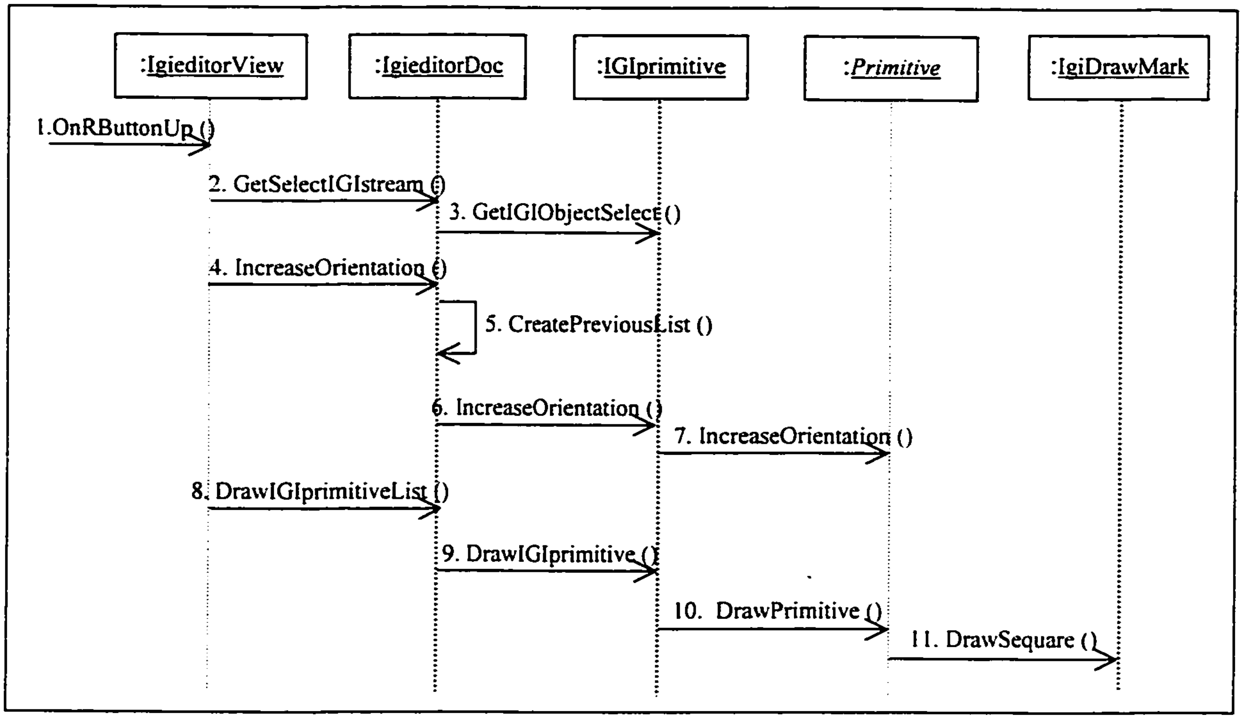


Figure 1-42 Sequence diagram for use case 19

1.3.2.20 Use case 20: User modifies a selected primitive object

1.3.2.20.1 Actor

An IGI Editor User.

1.3.2.20.2 Pursose

Actor modifies the size of a selected primitive object. Only the following primitive objects can be modified: Rectangle/Fill Rectangle, Ellipse/Fill Ellipse, Triangle/Fill Triangle, and Line.

1.3.2.20.3 Preconditions

A primitive object that is modifiable has been selected.

1.3.2.20.4 Typical Course of Events

Table 1-23 Typical course of events of use case 20

Actor Action	System Response
1. Move mouse to one of the select mark of the selected primitive object.	
2. Click left mouse button.	
3. Hold the left mouse button down and move the mouse to the desired position.	
4. Release the left mouse button	
	5. Create a previous primitive object list.
	6. Modify the primitive object's parameter.
	7. Redraw the primitive object list.

1.3.2.20.5 Interaction Diagram for the main scenario

1.3.2.20.5.1 Collaboration diagram

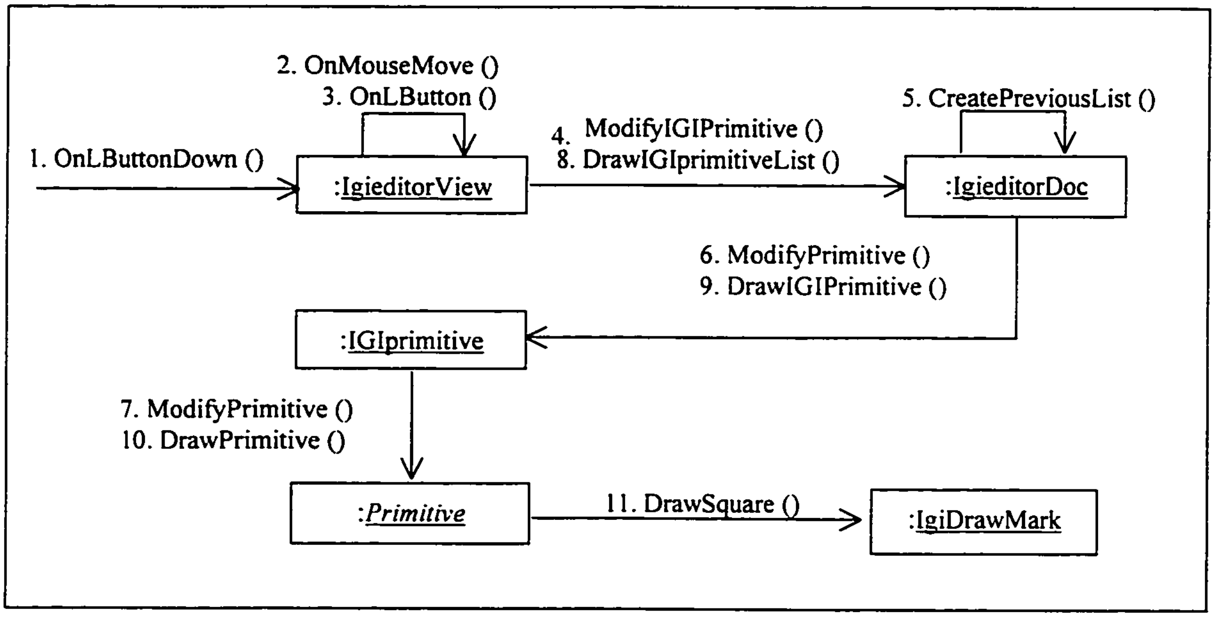


Figure 1-43 Collaboration diagram for use case 20

1.3.2.20.5.2 Sequence diagram

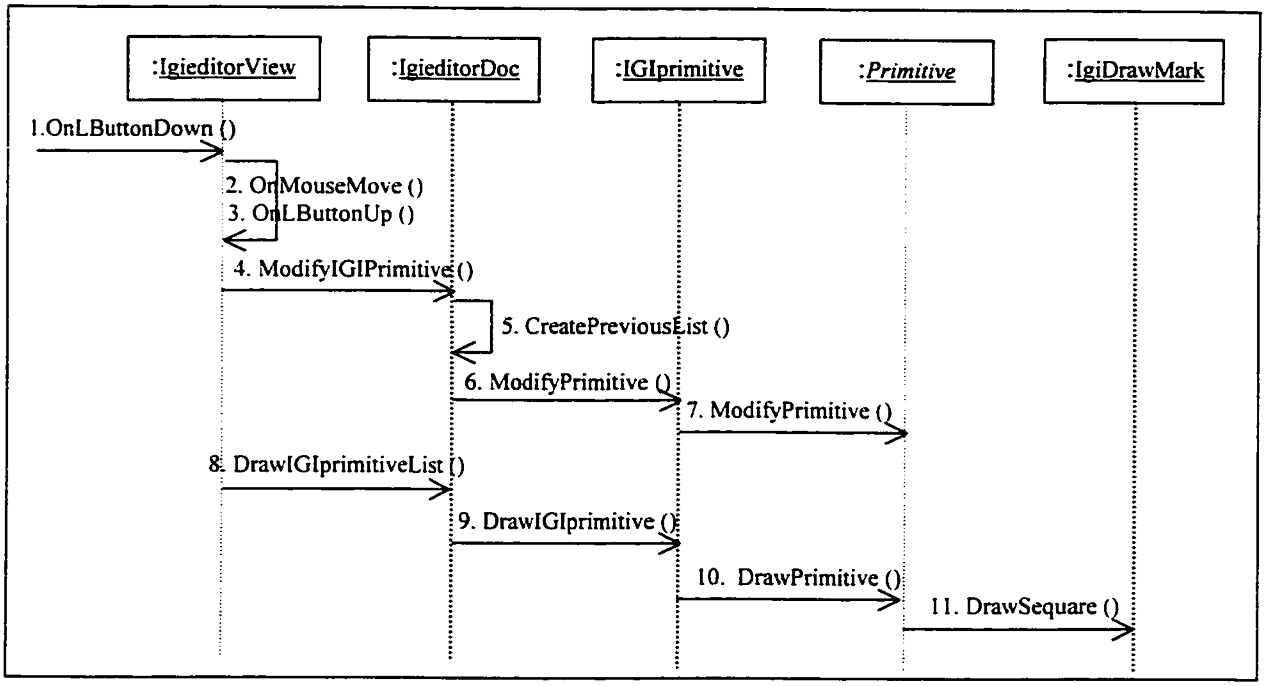


Figure 1-44 Sequence diagram for use case 20

1.3.2.21 Global Collaboration Diagram

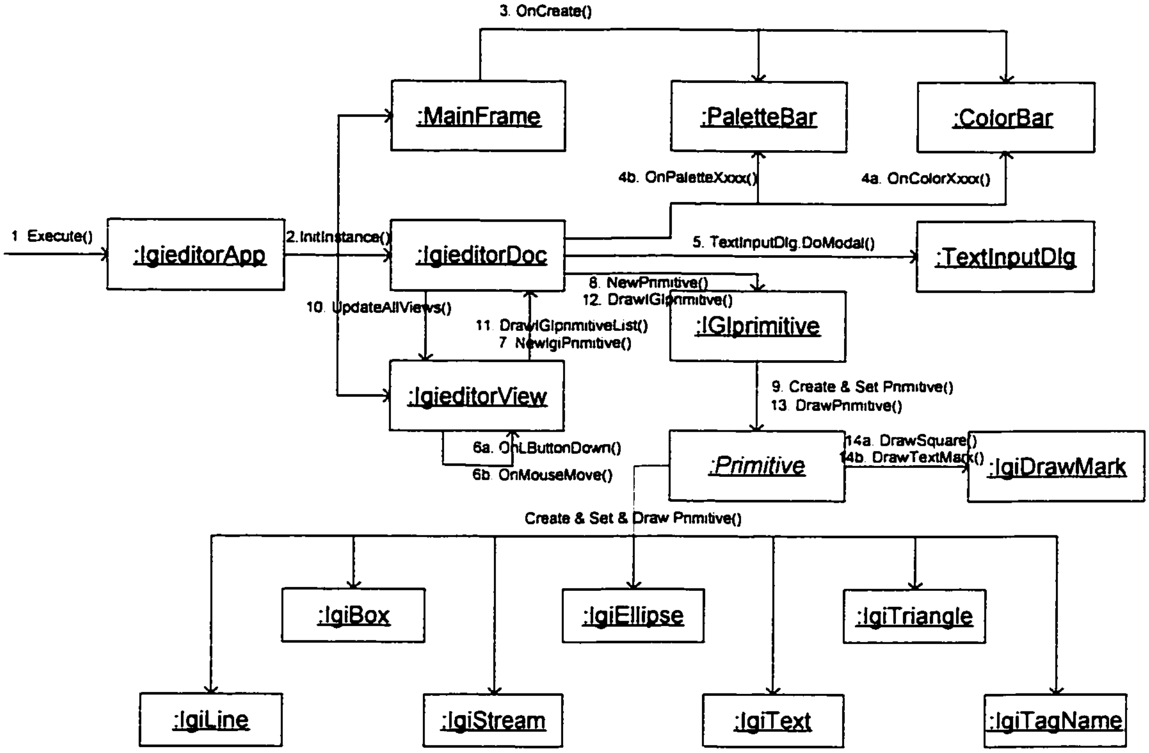


Figure 1-45 Collaboration Diagram

1.3.2.22 Class Diagram

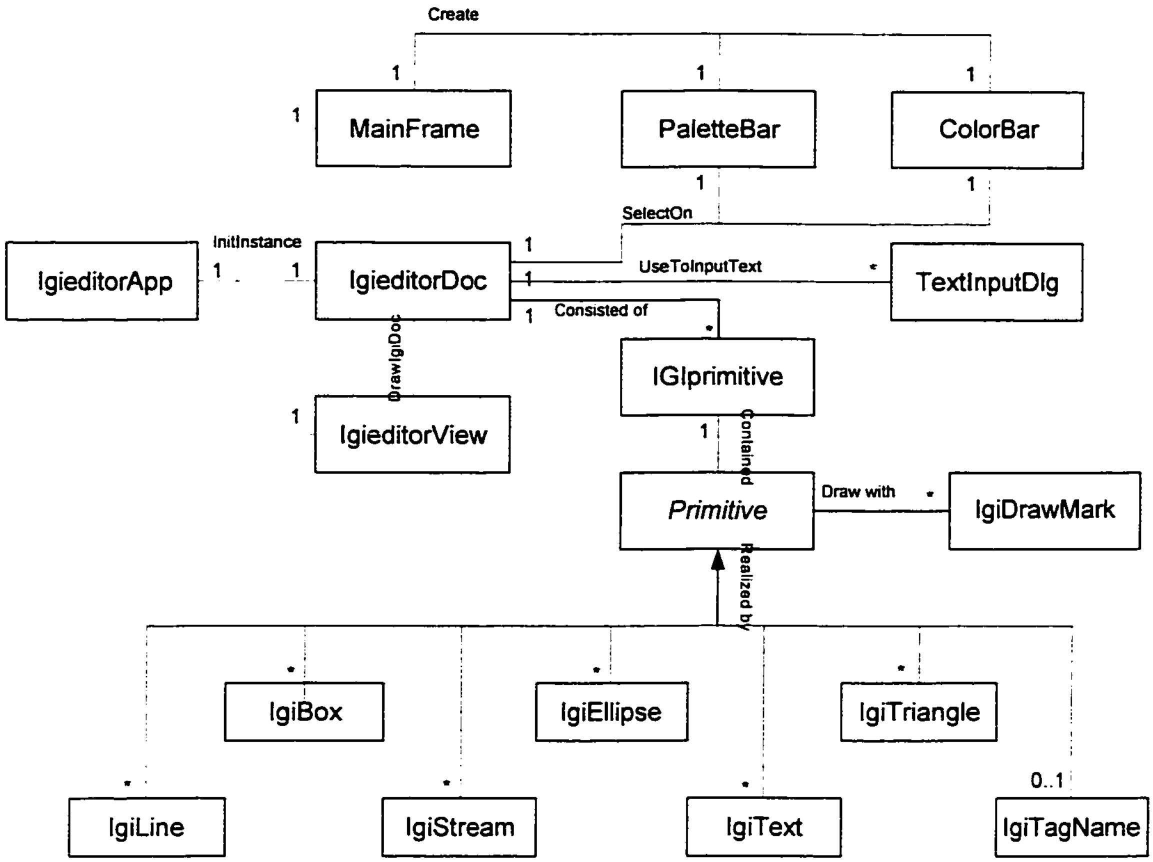


Figure 1-46 Domain Model

1.3.3 Performance Requirements

The software needs to communicate with hard disk to open or save an IGI document. Our goal is to open/save an IGI file from/to hard dish to/from main memory in less than 3 seconds, 98% of the time regardless of file size.

1.3.4 Design Constraints

For the file services, window support, graphic user interface, and graphic drawing, etc., we will use the Microsoft Foundation Class (MFC) Library. For the software design, we will follow the architecture of the MFC Application Framework that has been proven in many software environment and in many projects.

1.3.5 Software System Attributes

1.3.5.1 Reliability

The software allows saving the IGI document automatically in background (FUTURE GROWTH).

The software allows recover the IGI document in case of system shut down by accident (FUTURE GROWTH).

1.3.5.2 Availability

The software is available to any kind of engineers who are working on the simulation project.

1.3.5.3 Security

All usage requires user authentication (FUTURE GROWTH).

1.3.5.4 Maintainability

Using the MFC Library and the MFC application Framework to make the software easy to update and maintain.

1.3.5.5 Portability

N/A

1.3.6 Logical Database Requirements

N/A

1.3.7 Other Requirements

N/A

1.4 Definitions, Acronyms, and Abbreviations

Active IGI

An active IGI is an IGI document that has been loaded to IGI editor or an IGI is displayed on the screen.

Class diagram

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them.

Collaboration diagram

Collaboration diagrams illustrate object interactions in a graph or network format.

Control point

A control point is the position on that user can modify the size of a modifiable primitive object, such as Line, Rectangle, Triangle, and Ellipse. The control point is displayed as a small square called select mark when an object has been selected.

Edit action

An edit action is an operation on a primitive object. It can be one of the following operations: add/remove an object to IGI, move an object to another desired position, modify an object size for primitive Line/Rectangle/Triangle/Ellipse objects, copy any selected object, change an object position in the list, change object colour, rotate stream object orientation.

Icon

An icon is a simulated object. It includes the simulated structure parameters for the object and the graphic image to display on the screen.

IGI: Icon Graphic Image

An IGI is the graphical representation of the object as displayed to the user on the screen. An IGI includes the different kind of objects that is called the primitive of the IGI.

IGI Document

An IGI document is an IGI file that has been saved to a hard disk. That can be loaded to IGI Editor to display or modify.

Interaction Diagram

An interaction diagram illustrates the message interactions between instances (and classes) in the class model. The starting point for these interactions is the fulfillment of the post-conditions of the operation contracts.

Primitive

A primitive is basic object type in an IGI. An IGI can have the following primitives: Line, Rectangle, Triangle, Ellipse, Text, Tagname, and Stream. The different primitive has the different data structure. A primitive can have many primitive object that has same attributes but with different values.

Primitive object

A primitive object is an instance of a kind of primitive. It is the basic graphicl symbol that is displayed on the edit area.

Primitive object list

A primitive object list is a list of primitive objects adding to an IGI document. The objects can have different type of primitive. The position of the object in the list depends on the edit action. Usually, a last added object is added to the tail of the list. However, push/pop an object down/up can change the object position in the list.

Select mark

A select mark is displayed as a small square on the object control point when an object has been selected. It is displayed as box around the text for text object and tagname object. It shows an object state of selection. The select mark is disappeared if an object has been unselected. All the edit action can be done only after an object has been selected except of the adding a new object to the IGI.

Sequence diagram

Sequence diagrams illustrate interactions in a kind of fence format.

2. Software Design Descriptions of IGI Editor

2.1 Introduction

2.1.1 Purpose

The purpose of this design description is to provide a primary reference for code development. Therefore, it must contain all the information required by a programmer to write code.

2.1.2 Scope

The software detail design describes briefly the high level system structure, functionality, interactions with external systems, system issues, etc. It states the dominant design methodology and overview the architecture of the product briefly.

2.1.3 Document Overview

Section 2.2 of the Software Design Description describes the design methodology that will be used for this software design.

Section 2.3 of the document describes the architectural design issues. Section 2.3.1 provides a high level overview of the structural and functional decomposition of the system. Section 2.3.2 provides the top-level design view of the system and provides a basis for more detailed design work. It shows the main components and their relationship. Section 2.3.3 desribes each compoment from architecture in further detail. There is an overview diagram of the classes in the object model, followed by detailing attributes and interfaces of each class. Section 2.3.4 shows the activity diagram for each use case to show the workflow of the system. It records when, and in what order, events occur as the user interacts with the system. Section 2.3.5 shows a data flow diagram that records the dependencies between each function of the system, and the dataflows that are the Inputs/Outputs of the functions.

Section 2.4 lists the interface specification for each class of the system.

2.1.4 Definitions and Acronyms

Active diagram

An activity diagram describes the sequencing of activities, with support for both conditional and parallel behavior. An activity diagram is a variant of a state diagram in which most, if not all, the states are activity states.

Active IGI

An active IGI is an IGI document that has been loaded to IGI editor or an IGI is displayed on the screen.

Class diagram

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them.

Control point

A control point is the position on that user can change object size for the modifiable primitive object, such as Line, Rectangle, Triangle, and Ellipse. The control point is displayed as a small square called select mark when an object has been selected.

Data flow diagram

A data flow diagram records the dependencies between each function of the system, and illustrates the data flows that are the Inputs/Outputs of the functions. It contains the outlines of a model describling what the eventual software will do.

Edit action

An edit action is an operation on a primitive object. It can be one of the following operations: add/remove an object to IGI, move an object to another desired position, modify an object size for primitive Line/Rectangle/Triangle/Ellipse objects, copy any selected object, change an object position in the list, change object colour, rotate stream object orientation.

Existing IGI

An existing IGI is an IGI document file that is saved on the hard disk.

IGI: Icon Graphic Image

An IGI is the graphical representation of the object as displayed to the user on the screen. An IGI includes the different kind of objects that is called the primitive of the IGI.

IGI Document

An IGI document is an IGI file that has been saved to a hard disk. That can be loaded to IGI Editor to display or modify.

Primitive

A primitive is basic object type in an IGI. An IGI can have the following primitives: Line, Rectangle, Triangle, Ellipse, Text, Tagname, and Stream. The different primitive has the different data structure. A primitive can have many primitive object that has same attributes but with different values.

Primitive object

A primitive object is an instance of a kind of primitive. It is the basic graphicl symbol that is displayed on the edit area.

Primitive object list

A primitive object list is a list of primitive objects adding to an IGI document. The objects can have different type of primitive. The position of the object in the list depends on the edit action. Usually, a last added object is added to the tail of the list. However, push/pop an object down/up can change the object position in the list.

Select mark

A select mark is displayed as a small square on the object control point when an object has been selected. It is displayed as box around the text for text object and tagname object. It shows an object state of selection. The select mark is disappeared if an object has been unselected. All the edit action can be done only after an object has been selected except of the adding a new object to the IGI.

2.2 Design Methodology

Object-oriented design methodology is used in the software detail design. Object-oriented design centers on finding an appropriate set of classes and defining their contents and behavior. It involves determining the proper set of classes and then filling in the details of their implementation. Object-oriented design is fundamentally a three-step process: identifying the classes, characterizing them, and then defining the associated actions.

2.3 Architectural Design (AD)

2.3.1 Rational

This section summarizes the main issues and their selected solutions. It provides a high level overview of the structural and functional decomposition of the system. Focus is on how and why the system was decomposed in a particular way rather than on details of the particular components. It also includes the information on the major responsibilities and roles the system (or portions their of) must play.

2.3.1.1 Application Framework

The IGI Editor is a Windows Application. It will be run under Windows 2000/NT environment. According to the design constraints [1] in the SRS, we use the MFC application framework to develop the project. The MFC library application framework includes its own application structure – one that's been proven in many software environments and in many projects.

2.3.1.2 The Document – View Architecture

Typically, most of MFC library application contains application and frame classes plus two other classes that represent the "document" and "view". The document-view architecture is the core of the MFC application framework.

In simple terms, the document-view architecture separates data from the user's view of the data. In an MFC library application, documents and views are represented by instances of C++ classes.

The document base class code interacts with the File Open and File Save menu items; the derived document class does the actual reading and writing of the document object's data. (The application framework does most of the work of displaying the File Open and File Save dialog boxes and opening, closing, reading, and writing files). The view base class represents a window contained inside a frame window; the derived view class interacts with its associated document class and does the application's display and print I/O. The derived view class and its base classes handle Windows messages. The MFC library orchestrates all interactions among documents, views, frame windows, and the application object, mostly through virtual functions.

2.3.1.3 Single Document Interface

The MFC library supports two distinct application types: Single Document Interface (SDI) and Multiple Document Interface (MDI). An SDI application has, from the user's point view, only one window. If the application depends on disk-file "document," only one document can be loaded at a time. An MDI application has multiple child windows, each of which corresponds to an individual document. The MFC library application framework architecture ensures that most SDI applications can be upgraded easily to MDI applications.

For the IGI Editor, because it is a simple icon graphic editor and user only can edit one icon at a time, therefore, we choose the SDI as the IGI Edtor's application type.

2.3.1.4 Reading and Writing Document

In the MFC library, the process of saving and restoring objects is called serialization. The designated classes have a member function named *Serialize*. When the application framework calls *Serialize* for a particular object, the data for the object is either saved on disk or read from disk.

With the MFC library, objects of class *Cfile* represent disk files. A *Cfile* object encapsulates the binary file handle. The application framework uses this file handle for Win32 *ReadFile*, *WriteFile*, and *SetFilePointer* calls. If the application does no direct disk I/O but instead relies on the serialization process, we can avoid direct use of *Cfile* objects. Between the *Serialize* function and the *CFile* object is an archive object (of class *CArchive*).

The *CArchive* object object buffers data for the CFile object, and it maintains an internal flag that indicates whether the archive is storing (writing to disk) or loading (reading from disk). Only one active archive is associated with a file at any one time. The application framework takes care of constructing the CFile and CArchive objects, opening the disk file for the CFile object and associating the archive object with the file. All we have to do (in our *Serialize* function) is load data from or store data in the archive object. The application framework calls the document's *Serialize* function during the File Open and File Save processes.

The Figure 2-1 illustrates the relationships among the various classes, and Figure 2-2 illustrates the object relationships. Figure 2-3 shows the serialization process.

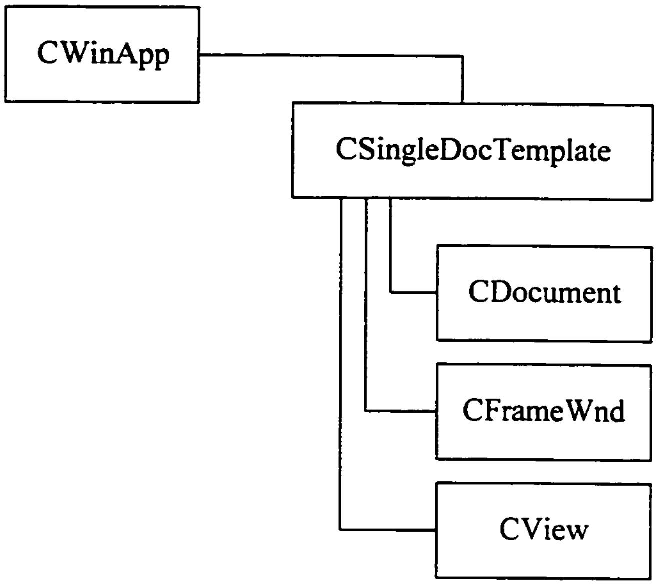


Figure 2-1 Class relationships

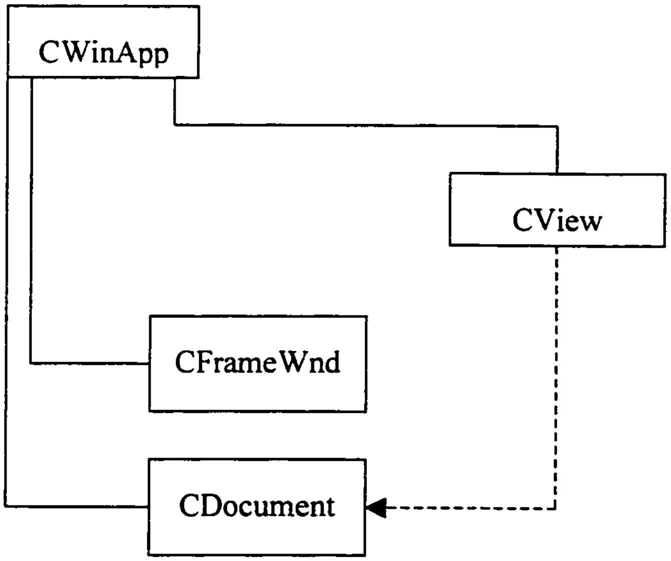


Figure 2-2 Object relationships

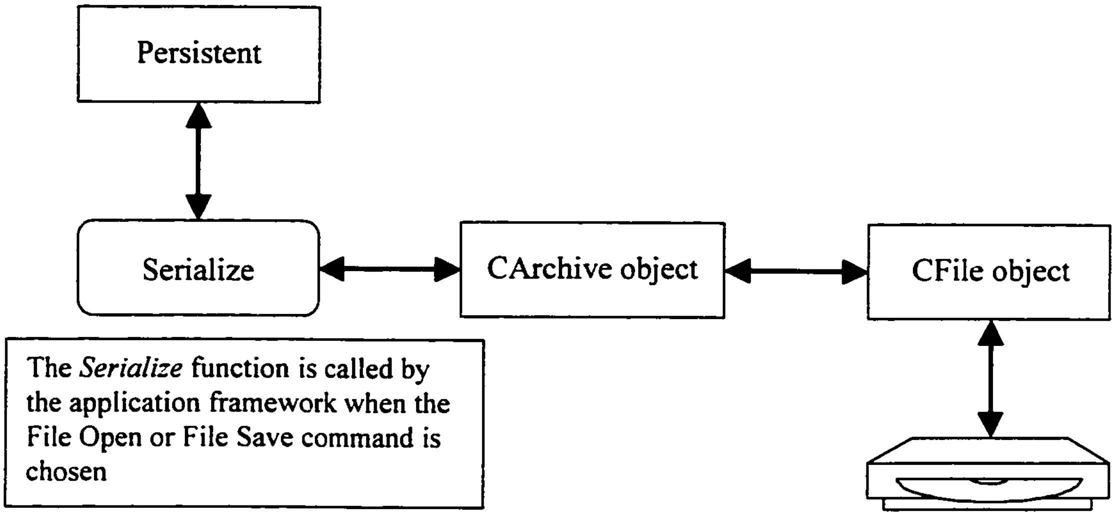

Persistent
Serialize
CArchive object
CFile object
The *Serialize* function is called by the application framework when the File Open or File Save command is chosen

Figure 2-3 The serialization process

For more information about the MFC library application framework and the document-view architecture, please see the reference book "Programming Microsoft Visual C++" fifth edition by David J. Krugliski, George Shepherd, and Scot Wingo.

2.3.1.5 IGI Document Data Structure

In the document-view architecture, the document object holds the data and the view object displays the data and allows editing. For the IGI Editor, one icon graphic image is saved to the hard disk as an igi document. It consists of a serial of primitive objects. We use a link list to represent the igi document data. When user selects a primitive and adds it to the igi document, one object of the primitive is added to the tail of the list. If user deletes a primitive object, the object is removed from the link list. When user saves the igi document, each primitive object's properties in the link list is written to the file with the *Serialize* function described before. Figure 4 shows the igi document data structure that is saved to the hard disk as an igi file.

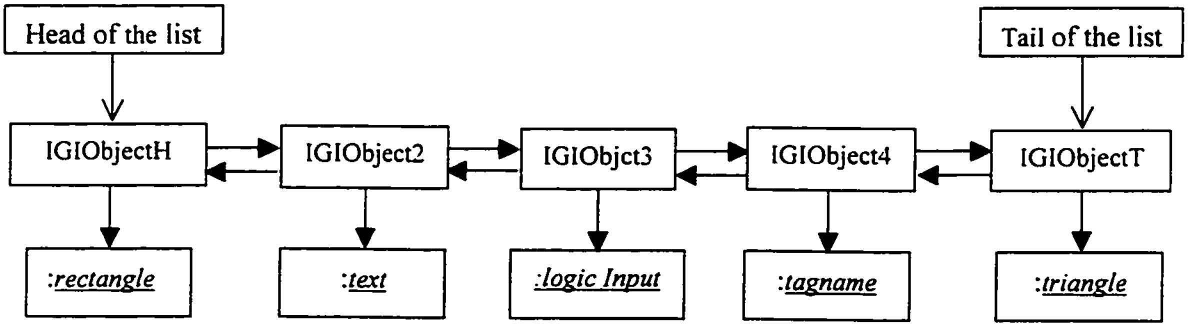


Figure 2-4 IGI document data structure

2.3.2 Architecture

The architecture provides the top-level design view of a system and provides a basis for more detailed design work. A software system is a set of communicating entities that collaborate to perform a task. The architecture shows these design entities, their relationships and the relationship to the actors in the system.

With the MFC library application framework and the document-view architecture, the IGI Editor application program contains the following main class: *CIgieditorApp*, *CMainFrame*, *CIgieditorDoc*, *CIgieditorView*.

The ***CIgieditorApp*** class – An object of class CIgieditorApp represents an application. The program defines a single global CIgieditorApp object, *theApp*. The CWinApp base class determines most of *theApp*'s behavior.

Application startup – When the user starts the application, Windows calls the application framework's built-in *WinMain* function, and *WinMain* looks for a globally constructed application object of a class derived from *CWinApp*.

The ***CMainFrame*** class – An object of class *CMainFrame* represents the application's main frame window. When the constructor calls the *Create* member function of the base class *CFrameWnd*, Windows creates the actual window structure and the application framework links it to the C++ object. The *ShowWindow* and *UpdateWindow* functions, also member functions of the base class, must be called in order to display the window.

Application shutdown – The user shuts down the application by closing the mainframe window. This action initiates a sequence of events, which ends with the destruction of the *CMainFrame* object, the exit from *Run*, the exit from *WinMain*, and the destruction of the *CIgieditorApp* object.

The ***CIgieditorDoc*** class – An object of class *CigieditorDoc* holds the data of the application. The different objects of the primitive classes represent the application data. The primitive classes include the following classes: *CIgiBox*, *CIgiEllipse*, *CIgiLine*, *CIgiTriangle*, *CIgiStream*, *CIgiText*, *CIgiTagName*. The most of property and interface for these classes are same and they can be derived from their parent class *CPrimitive*. When user adds/removes a primitive object to an icon, an object of the primitive class will be added/removed to/from the primitive link list. Because the structures of the primitive class are different, we use class *CIGIprimitive* as an adapter of these primitive classes. An object of class *CIGIprimitive* is a node of the link list. In the class *CIGIprimitive*, a *PrimitivePointer* is used to point to an object of a primitive class according to the type of a primitive.

The ***CIgieditorView*** class – An object of class *CigieditorView* displays the data of the application. It has a pointer of *CIGIprimitive* that points to the currently selected primitive object and allows editing the object.

Figure 2-5 shows the main classes and their relationship in the IGI Editor program.

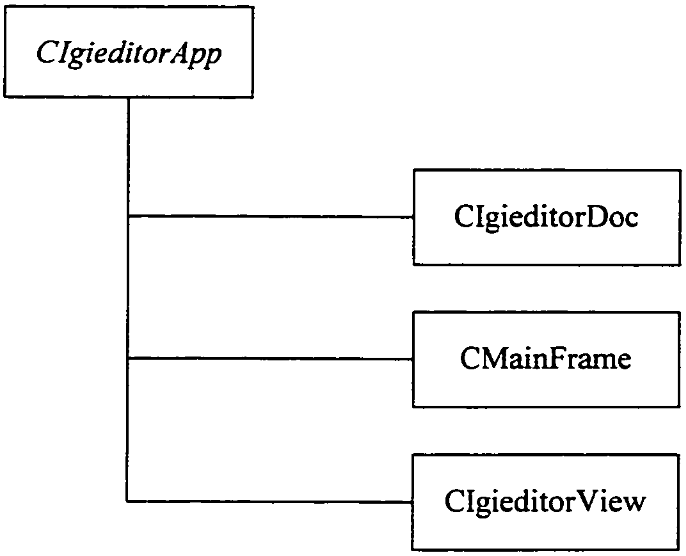


Figure 2-5 Main Class and Relationship Diagram

The architectural design follows the Model/View/Controller (MVC) architectural pattern. MVC consists of three kinds of ojects. The Model is the application object, the View is its screen presentation, and the controller defines the way the user interface reacts to user input. MVC decouples views and models by establishing a subscribe/notify protocol between them. A view must ensure that its appearance reflects the state of the model. Whenever the model's data changes, the model notifies views that depend on it. In response, each view gets an opportunity to update itself.

In this project, the object of class *CIGIprimitive* is a model object. It responds to message

such as add, remove, move and modify primitive objects and reveal various aspects of its internal data by a *PrimitivePointer* to point to an object of a primitive class according to the type of a primitive.

The object of class *CigieditorView* is the view object in the MVC pattern. It provides the way of presenting the data to the user.

The object of class *CigieditorDoc* has the role of controller object in the MVC pattern. It corrdinates the actions of the model and the view by sending appropriate messages to them. It passes the command to *CIGIprimitive* to update the state of primitive objects and then send UpdateView to *CigieditorView* to display the new view.

The figure 2-6 shows the objects relationshop in the MVC framework.

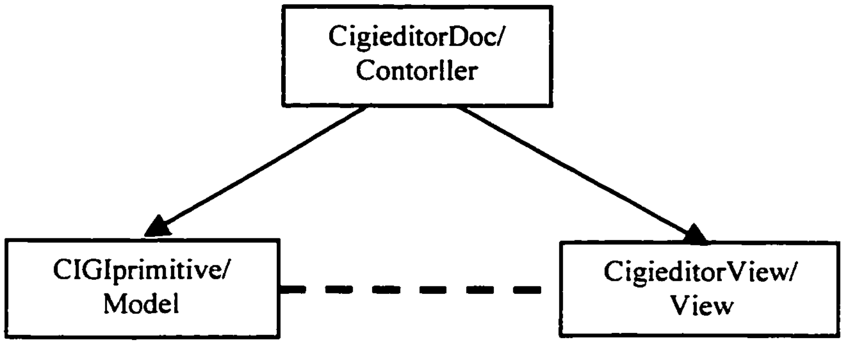


Figure 2-6 The Model-View-Controller Design pattern

⟶ Controller sends messages to Model/View

- - - - - - - Model notifies View

2.3.3 Object Model

Figure 2-7 shows the class diagram for the IGI Editor program.

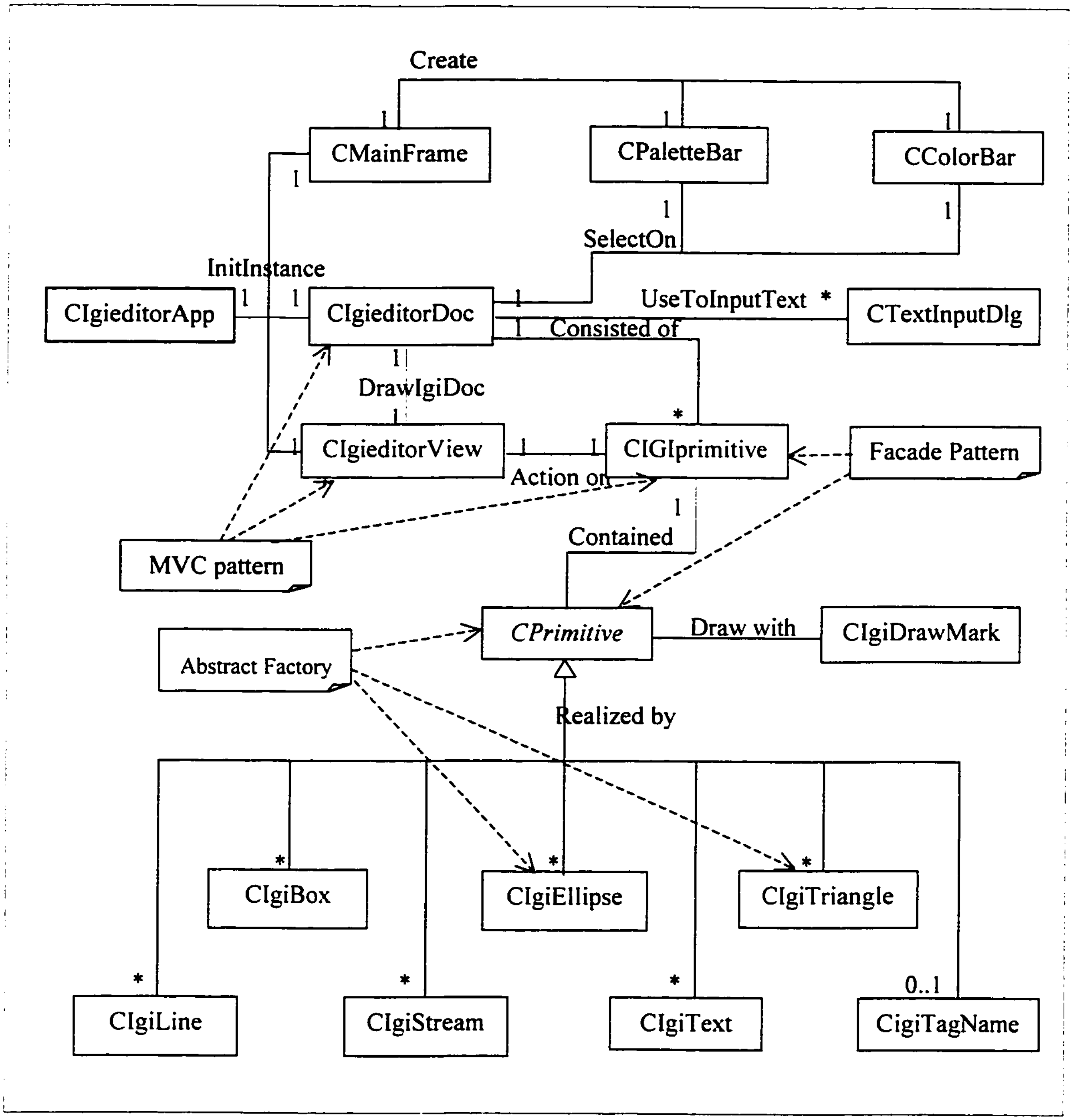


Figure 2-7 Main Class and Relationship Diagram

Except the MVC pattern, a few other design patterns are used in the design.

Command Pattern: it encapsulates a request as an object, thereby letting parameterize clients with different request, queue or log requests, and support undoable operations.

Facade Pattern: it provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.

Abstract Factory Pattern: it provides an interface for creating families of related or dependent objects without specifying their concrete classes.

An object of class *CPaletterBar and CColorBar* is created when the function CMainFrame::OnCreate() is called. From the object of class *CPaletterBar and CColorBar,* user can select the type of primitive and color that will be added and displayed on the edit area.

An object of class *CTextInputDlg* is created when user select a text button on the primitive palette to add a new text. The text input dialog window is displayed to allow user to type a new text string. The class *CTextInputDlg* is used by the function CIgieditorDoc::OnPaletteText().

An object of class *CIgiDrawMark* is created when an object of the primitive class is selected. It is used to draw an object selected mark and is called by the function DrawPrimitive() in each primitive class.

An object of class *CToolDlg* is created to configure toolbar setting. User can set the display of toolbars, tool tips and the primitive palette columns. It is called by the function CMainFrame::OnToolsToolbar().

The following tables show the detailing attributes and interfaces of each class.

2.3.3.1 Class CMainFrame

Table 2-1 Attributes and interfaces of class CMainFrame

Class CMainFrame

Attribute:
 	
public:
        BOOL m_bToolTips;
        BOOL m_bColor;
 	
protected:
        CToolBar   m_wndMainBar;

Class CMainFrame

CPaletteBar m_wndPaletteBar;
CColorBar m_wndColorBar;

Method():
 	
public:
int PreTranslateMessage(MSG* pMsg);
 	
protected:
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
afx_msg void OnToolsToolbar();

2.3.3.2 Class CigieditorView

Table 2-2 Attributes and interfaces of class CigieditorView

Class CIgieditorView

Method():
 	
public:
CIgieditorDoc* GetDocument();
virtual void OnDraw(CDC* pDC); // overridden to draw this view
virtual int PreTranslateMessage(MSG* pMsg);
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
 	
protected:
afx_msg void OnCaptureChanged(CWnd *pWnd);
afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
afx_msg void OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
afx_msg void OnMove(int x, int y);
afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
afx_msg void OnMouseMove(UINT nFlags, CPoint point);
afx_msg BOOL OnEraseBkgnd(CDC* pDC);
 	
private:
BOOL MovePrimitive(CPoint point);

Class CIgieditorView

void ModifyPrimitive(CPoint point);
void InvertEllipse (CDC* pDC, CPoint ptFrom, CPoint ptTo);
void InvertBox (CDC* pDC, CPoint ptFrom, CPoint ptTo);
void InvertLine (CDC* pDC, CPoint ptFrom, CPoint ptTo);

2.3.3.3 Class CigieditorDoc

Table 2-3 Attributes and interfaces of class CigieditorDoc

Class CIgieditorDoc

Attribute:
 	
public:
CTypedPtrList<CObList,CIGIprimitive*> m_primitiveList;
BOOL m_bColor;
 	
protected:
CTypedPtrList<CObList,CIGIprimitive*> m_primitivePrevList;
int m_iColorPalette[16];
int m_iPrimitivePalette[16];
int m_iSelectOrder;
int m_iCountSelect;
int m_iTagNameFlag;
BOOL m_bFillMode;
EditModeIndex m_iEditMode;
PalettePrimitive m_iCurrentPrimitive;
COLORREF m_cCurrentColor;
CString m_cTextInput;

Method():
 	
public:
void OnKeyDelete();
void InvertIgiPrimitive(CDC* pDC, CIGIprimitive* p, int dx, int dy);
CIGIprimitive* GetSelectIGIstream();
void IncreaseOrientation(CIGIprimitive* p);
int GetNewStreamNumber(int iType, int iDir);
CString GetTextInput();
void SetTagNameFlag(int tagNameFlag);
int GetTagNameFlag();
void SetSelectOrder(int selectOrder);
void ModifyIGIPrimitive(CIGIprimitive* p, int dx, int dy, int iIndex);

Class CIgieditorDoc

```
        void SetEditMode(EditModeIndex editMode);
        void MoveIGIPrimitiveList(int dx, int dy);
        void SetIGIObjectSelectList(BOOL select);
        void DrawIGIprimitiveList(CDC *pDC);
        void UnSetPaletteBar();

        int GetSelectOrder();
        BOOL GetFillMode();

        CIGIprimitive* SelectIGIprimitive(CPoint point);
        CIGIprimitive* NewIgiPrimitive(PalettePrimitive item, int x[6]);
        PalettePrimitive GetCurrentPrimitive();
        EditModeIndex GetEditMode();
        COLORREF GetCurrentColor();

        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);

protected:
        CIGIprimitive* CopyIGIprimitive(CIGIprimitive* pIGIprimitive, int dx);
        void SwapList();
        void CreatePreviousList();
        void SetPaletteBar(int index);
        void SetColorBar(int index);
        void InitDocument();
        void UpdateColor();
        void DeleteContents();

        afx_msg void OnSelectAll();
        afx_msg void OnUnSelectAll();
        afx_msg void OnDeleteAll();
        afx_msg void OnDeleteSome();
        afx_msg void OnEditCopy();
        afx_msg void OnEditUndo();
        afx_msg void OnPushOne();
        afx_msg void OnPushBottom();
        afx_msg void OnPopOneUp();
        afx_msg void OnPopToTop();

        afx_msg void OnColorBlack();
        afx_msg void OnColorYellow();
        afx_msg void OnColorRed();
```

Class CIgieditorDoc

```
afx_msg void OnColorBlue();
afx_msg void OnColorGreen();
afx_msg void OnColorCyan();
afx_msg void OnColorMagenta();
afx_msg void OnColorWhite();
afx_msg void OnColorDarkblue();
afx_msg void OnColorDarkgreen();
afx_msg void OnColorDarkcyan();
afx_msg void OnColorDarkred();
afx_msg void OnColorDarkmagenta();
afx_msg void OnColorDarkyellow();
afx_msg void OnColorDarkgray();
afx_msg void OnColorLightgray();

afx_msg void OnPaletteRectangle();
afx_msg void OnPaletteFillRectangle();
afx_msg void OnPaletteTriangle();
afx_msg void OnPaletteFillTriangle();
afx_msg void OnPaletteEllipse();
afx_msg void OnPaletteFillEllipse();
afx_msg void OnPaletteText();
afx_msg void OnPaletteTag();
afx_msg void OnPaletteProcessIn();
afx_msg void OnPaletteProcessOut();
afx_msg void OnPaletteLogicIn();
afx_msg void OnPaletteLogicOut();
afx_msg void OnPaletteAnalogIn();
afx_msg void OnPaletteAnalogOut();
afx_msg void OnPaletteLine();
afx_msg void OnPaletteIndicator();

afx_msg void OnUpdateSelectAll(CCmdUI* pCmdUI);
afx_msg void OnUpdateUnSelectAll(CCmdUI* pCmdUI);
afx_msg void OnUpdateDeleteAll(CCmdUI* pCmdUI);
afx_msg void OnUpdateDeleteSome(CCmdUI* pCmdUI);
afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
afx_msg void OnUpdateEditUndo(CCmdUI* pCmdUI);
afx_msg void OnUpdatePushOne(CCmdUI* pCmdUI);
afx_msg void OnUpdatePushBottom(CCmdUI* pCmdUI);
afx_msg void OnUpdatePopOneUp(CCmdUI* pCmdUI);
afx_msg void OnUpdatePopToTop(CCmdUI* pCmdUI);

afx_msg void OnUpdateColorBlack(CCmdUI* pCmdUI);
```

Class CIgieditorDoc

afx_msg void OnUpdateColorYellow(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorRed(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorBlue(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorGreen(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorCyan(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorMagenta(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorWhite(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorDarkblue(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorDarkgreen(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorDarkcyan(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorDarkred(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorDarkmagenta(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorDarkyellow(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorDarkgray(CCmdUI* pCmdUI);
afx_msg void OnUpdateColorLightgray(CCmdUI* pCmdUI);
 	
afx_msg void OnUpdatePaletteRectangle(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteFillRectangle(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteTriangle(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteFillTriangle(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteEllipse(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteFillEllipse(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteText(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteTag(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteProcessIn(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteProcessOut(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteLogicIn(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteLogicOut(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteAnalogIn(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteAnalogOut(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteLine(CCmdUI* pCmdUI);
afx_msg void OnUpdatePaletteIndicator(CCmdUI* pCmdUI);

2.3.3.4 Class CIGIprimitive

Table 2-4 Attributes and interfaces of class CIGIprimitive

Class CIGIprimitive
Attribute: private:

Class CIGIprimitive

```
        ObjectPointer m_pObject;
        int m_iVar[6];
        BOOL m_bFill;
        COLORREF m_cPenColor;
        PrimitiveType m_iType;
        CString m_cTextInput;
```

```
Method():

public:
        int GetPrimitiveType();
        void SetFillMode(BOOL bFillMode);
        int GetStreamDir();
        int GetStreamType();
        void IncreaseOrientation();
        void SetStreamNumber(int iNoStream);
        int GetStreamNumber(int iType. int iDir);
        BOOL IsLookForStream(int iType, int iDir);
        CString GetIGItextinput();
        void SetIGItextinput(CString cTextInput);
        BOOL GetFillMode();
        COLORREF GetPenColor();
        PalettePrimitive GetPrimitiveItem();
        void GetParameter(int* x, int dx);
        BOOL GetIGIObjectSelect();
        void UpdateColor(COLORREF currentColor);
        void MoveIGIPrimitive(int dx, int dy);
        void SetIGIObjectSelect(BOOL select);
        void InvertIgiPrimitive(CDC* pDC, int dx, int dy);
        void ModifyPrimitive(int dx, int dy, int index);
        int GetMarkIndex();
        BOOL GetSelectMark();
        BOOL SelectIGIprimitive(CPoint point);
        void SetPenColor(COLORREF currentColor);
        void NewPrimitive();
        void SetParameter(int x[6]);
        void SetPrimitiveType(PalettePrimitive item);
        BOOL DrawIGIprimitive(CDC* pDC);
        CPoint GetModifyPoint(int iMarkIndex, int iFromIndex);

        virtual void Serialize(CArchive& ar);
```

2.3.3.5 Class CPrimitive

Table 2-5 Attributes and interfaces of class CPrimitive

Class *CPrimitive*

Attribute:
 	
protected:
int m_iSquareSize;
COLORREF m_cPenColor;
BOOL m_bObjectSelect;
BOOL m_bSelectMark;
int m_iMarkIndex;

Method():
 	
public:
virtual void DrawPrimitive(CDC* pDC);
virtual void Serialize(CArchive &ar);
virtual BOOL SelectPrimitive (int x, int y);
virtual void ModifyPrimitive (int dx, int dy, int index);
virtual void MovePrimitive (int DX, int DY);
virtual void InvertIgiPrimitive(CDC* pDC, int dx, int dy);
 	
virtual void GetParameter(int* x, int dx);
virtual void SetPenColor(COLORREF currentColor);
virtual void SetObjectSelect(BOOL select);
virtual BOOL GetObjectSelect();
virtual COLORREF GetPenColor();

2.3.3.6 Class CIgiBox

Table 2-6 Attributes and interfaces of class CIgiBox

Class CIgiBox

Attribute:
 	
private:
POINT m_end;
POINT m_start;
BOOL m_bFill;

Method(): public:

Class CIgiBox

CIgiBox();
CIgiBox(int x1, int y1. int x2, int y2);
 	
int GetMarkIndex();
BOOL GetSelectMark();
void GetParameter(int* x, int dx);
CPoint GetModifyPoint(int iMarkIndex);
void SetFillMode(BOOL bFillMode);
BOOL GetFillMode();
void SetPenColor(COLORREF currentColor);
void SetObjectSelect(BOOL select);
BOOL GetObjectSelect();
COLORREF GetPenColor();
BOOL SelectPrimitive (int x, int y);
 	
void Serialize(CArchive& ar);
void InvertIgiPrimitive(CDC* pDC. int dx, int dy);
void MovePrimitive (int dx, int dy);
void DrawPrimitive(CDC* pDC);
void ModifyPrimitive (int dx, int dy, int index);

2.3.3.7 Class CIgiEllipse

Table 2-7 Attributes and interfaces of class CIgiEllipse

Class CIgiEllipse

Attribute:
 	
private:
POINT m_lowright;
POINT m_topleft;
BOOL m_bFill;

Method():
 	
public:
CIgiEllipse();
CIgiEllipse(int x1, int y1, int x2, int y2);
 	
int GetMarkIndex();
BOOL GetSelectMark();

Class CIgiEllipse

void GetParameter(int *x, int dx);
CPoint GetModifyPoint(int iMarkIndex);
void SetFillMode(BOOL bFillMode);
BOOL GetFillMode();
void SetPenColor(COLORREF currentColor);
void SetObjectSelect(BOOL select);
BOOL GetObjectSelect();
COLORREF GetPenColor();
void Serialize(CArchive &ar);
void InvertIgiPrimitive(CDC* pDC, int dx, int dy);
void MovePrimitive (int dx, int dy);
void DrawPrimitive(CDC* pDC);
void ModifyPrimitive (int dx, int dy, int index);
BOOL SelectPrimitive (int x, int y);

2.3.3.8 Class CIgiLine

Table 2-8 Attributes and interfaces of class CIgiLine

Class CIgiLine

Attribute:
 	
private:
POINT m_start;
POINT m_end;

Method():
 	
public:
CIgiLine();
CIgiLine(int x1, int y1, int x2, int y2);
int GetMarkIndex();
BOOL GetSelectMark();
void GetParameter(int* x, int dx);
CPoint GetModifyPoint(int iMarkIndex);
void SetPenColor(COLORREF currentColor);
void SetObjectSelect(BOOL select);
BOOL GetObjectSelect();
COLORREF GetPenColor();
void Serialize(CArchive &ar);
void InvertIgiPrimitive(CDC* pDC, int dx, int dy);
void MovePrimitive (int dx, int dy);

Class CIgiLine
void DrawPrimitive(CDC* pDC); void ModifyPrimitive (int dx, int dy, int index); BOOL SelectPrimitive (int x, int y);

2.3.3.9 Class CIgiTriangle

Table 2-9 Attributes and interfaces of class CIgiTriangle

Class CIgiTriangle

Attribute:
 	
private:
BOOL m_bFill;
POINT m_first;
POINT m_second;
POINT m_third;

Method():
 	
public:
CIgiTriangle();
CIgiTriangle(int x1, int y1, int x2, int y2, int x3, int y3);
int GetMarkIndex();
BOOL GetSelectMark();
void GetParameter(int* x, int dx);
CPoint GetModifyPoint(int iMarkIndex, int iFromIndex);
void SetFillMode(BOOL bFillMode);
BOOL GetFillMode();
void SetPenColor(COLORREF currentColor);
void SetObjectSelect(BOOL select);
BOOL GetObjectSelect();
COLORREF GetPenColor();
void Serialize(CArchive &ar);
void InvertIgiPrimitive(CDC* pDC, int dx, int dy); //, int Off);
void MovePrimitive (int dx, int dy);
void DrawPrimitive(CDC* pDC);
void ModifyPrimitive (int dx, int dy, int index);
BOOL SelectPrimitive (int x, int y);
 	
private:
BOOL SelectOnLine(int x, int y, POINT start, POINT end);

2.3.3.10 Class CIgiStream

Table 2-10 Attributes and interfaces of class CIgiStream

Class CIgiStream

Attribute:
 	
private:
int StreamArrow;
int StreamLength;
int m_iNoStream;
int m_iOrientation;
int m_iDir;
int m_iType;
POINT m_start;

Method():
public:
CIgiStream();
CIgiStream(int x, int y, int iType, int iDir, int iOrientatio, int iNoStream);
void GetParameter(int* x, int dx);
int GetStreamOrientation();
int GetStreamDir();
int GetStreamType();
void SetStreamNumber(int iNoStream);
int GetStreamNumber();
void SetObjectSelect(BOOL select);
BOOL GetObjectSelect();
void IncreaseOrientation();
void Serialize(CArchive &ar);
void InvertIgiPrimitive(CDC* pDC, int dx, int dy);
void MovePrimitive (int dx, int dy);
void DrawPrimitive(CDC* pDC);
BOOL SelectPrimitive (int x, int y);

2.3.3.11 Class CIgiText

Table 2-11 Attributes and interfaces of class CIgiText

Class CIgiText
Attribute: private: int m_iMarkHigh;

Class CIgiText

int m_iMarkLen;
POINT m_start;
CString m_cString;

Method():
public:
CIgiText();
CIgiText(int x, int y, int len, int high);
CString GetTextInput();
void SetTextInput(CString cTextInput);
void GetParameter(int* x, int dx);
void SetPenColor(COLORREF currentColor);
void SetObjectSelect(BOOL select);
BOOL GetObjectSelect();
COLORREF GetPenColor();
void Serialize(CArchive &ar);
void InvertIgiPrimitive(CDC* pDC, int dx, int dy);
void MovePrimitive (int DX, int DY);
void DrawPrimitive(CDC* pDC);
BOOL SelectPrimitive (int x, int y);

2.3.3.12 Class CIgiTagName

Table 2-12 Attributes and interfaces of class CIgiTagName

Class CIgiTagName

Attribute:
 	
private:
int m_iMarkHigh;
int m_iMarkLen;
POINT m_start;
CString m_cString;

Method():
 	
public:
CIgiTagName();
CIgiTagName(int x, int y, int len, int high);
void Serialize(CArchive &ar);
void InvertIgiPrimitive(CDC* pDC, int dx, int dy);
void MovePrimitive (int dx, int dy);
void DrawPrimitive(CDC* pDC);

Class CIgiTagName

BOOL SelectPrimitive (int x, int y);
void GetParameter(int* x, int dx);
BOOL GetObjectSelect();
COLORREF GetPenColor();
void SetObjectSelect(BOOL select);
void SetPenColor(COLORREF currentColor);

2.3.3.13 Class CIgiDrawMark

Table 2-13 Attributes and interfaces of class CIgiDrawMark

Class CIgiDrawMark

Attribute:
 	
private:
int m_iSquareSize;
int m_iY2;
int m_iX2;
int m_iY1;
int m_iX1;
CDC* m_pDC;

Method():
 	
public:
CIgiDrawMark();
void DrawMarkSetup(CDC *p, int X1, int Y1, int X2, int Y2);
void DrawSquare(BOOL yes);
void DrawTextMark(BOOL yes);

2.3.3.14 Class CTextInputDlg

Table 2-14 Attributes and interfaces of class CTextInputDlg

Class CTextInputDlg

Attribute:
 	
public:
CEdit m_TextInput;
CString m_cTextInput;

Class CTextInputDlg

Method():
 	
protected:
void OnOK();
BOOL OnInitDialog();
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
CTextInputDlg(CWnd* pParent = NULL); // standard constructor

2.3.3.15 Class CPaletteBar

Table 2-15 Attributes and interfaces of class CPaletteBar

Class CPaletteBar

Attribute:
 	
protected:
BOOL m_bRectangle;
UINT m_nColumns;

Method():
 	
public:
CPaletteBar();
UINT GetColumns();
void SetColumns(UINT nColumns);

2.3.3.16 Class CColorBar

Table 2-16 Attributes and interfaces of class CColorBar

Class CColorBar
Attribute: protected: UINT m_nColorColumns;

Method():
 	
public:
CColorBar();
UINT GetColumns();

Class CColorBar
void SetColumns(UINT nColorColumns);

2.3.3.17 Class CToolDlg

Table 2-17 Attributes and interfaces of class CToolDlg

Class CToolDlg

Attribute:
 	
public:
int m_nToolTips;
int m_nColumns;
int m_nColor;
BOOL m_bPalette;
BOOL m_bMain;
BOOL m_bColorbar;

Method():
 	
public:
CToolDlg(CWnd* pParent = NULL); // standard constructor
 	
protected:
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

2.3.4 Dynamic Model

This section illustrates the interesting events and states of an object, and the behavior of an object in reaction to an event for each use case.

The following active diagrams show the workflow of the system. It records when, and in what order, events occur as the user interacts with the system.

2.3.4.1 User creates a new IGI document

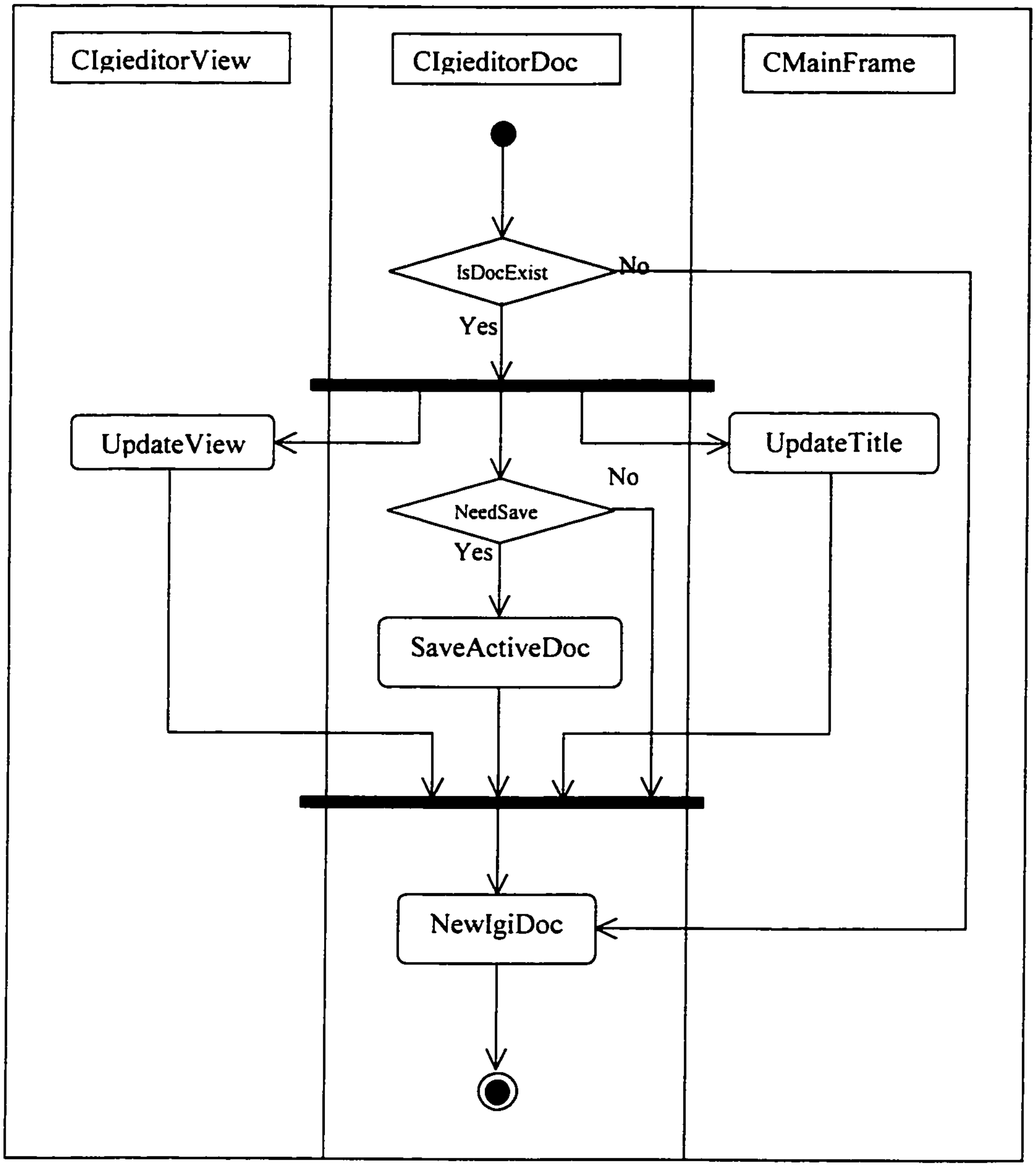


Figure 2-8 Active diagram of creating a new IGI document

2.3.4.2 User opens an existing IGI document

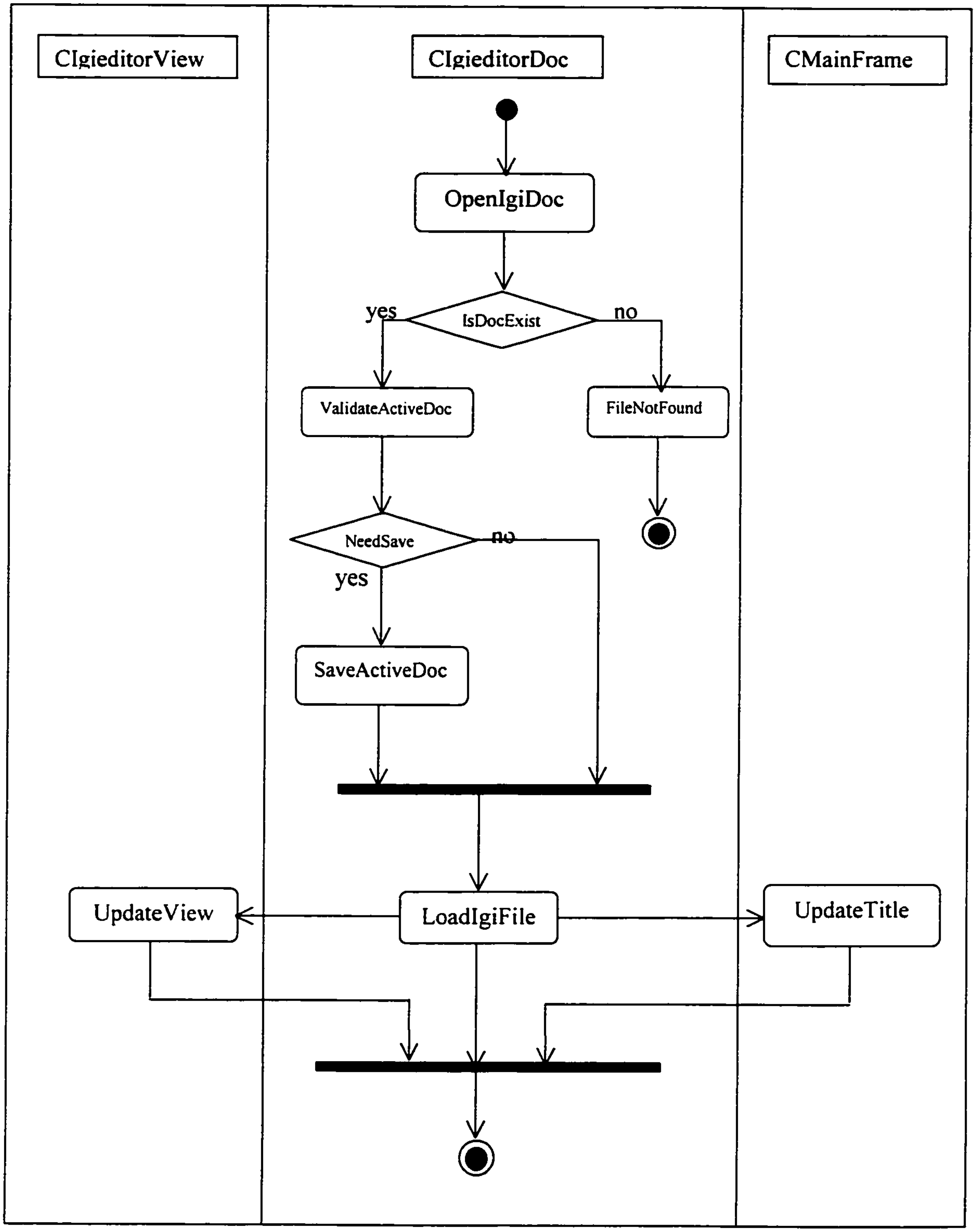


Figure 2-9 Active diagram of opening an existing IGI document

2.3.4.3 User saves an active IGI document

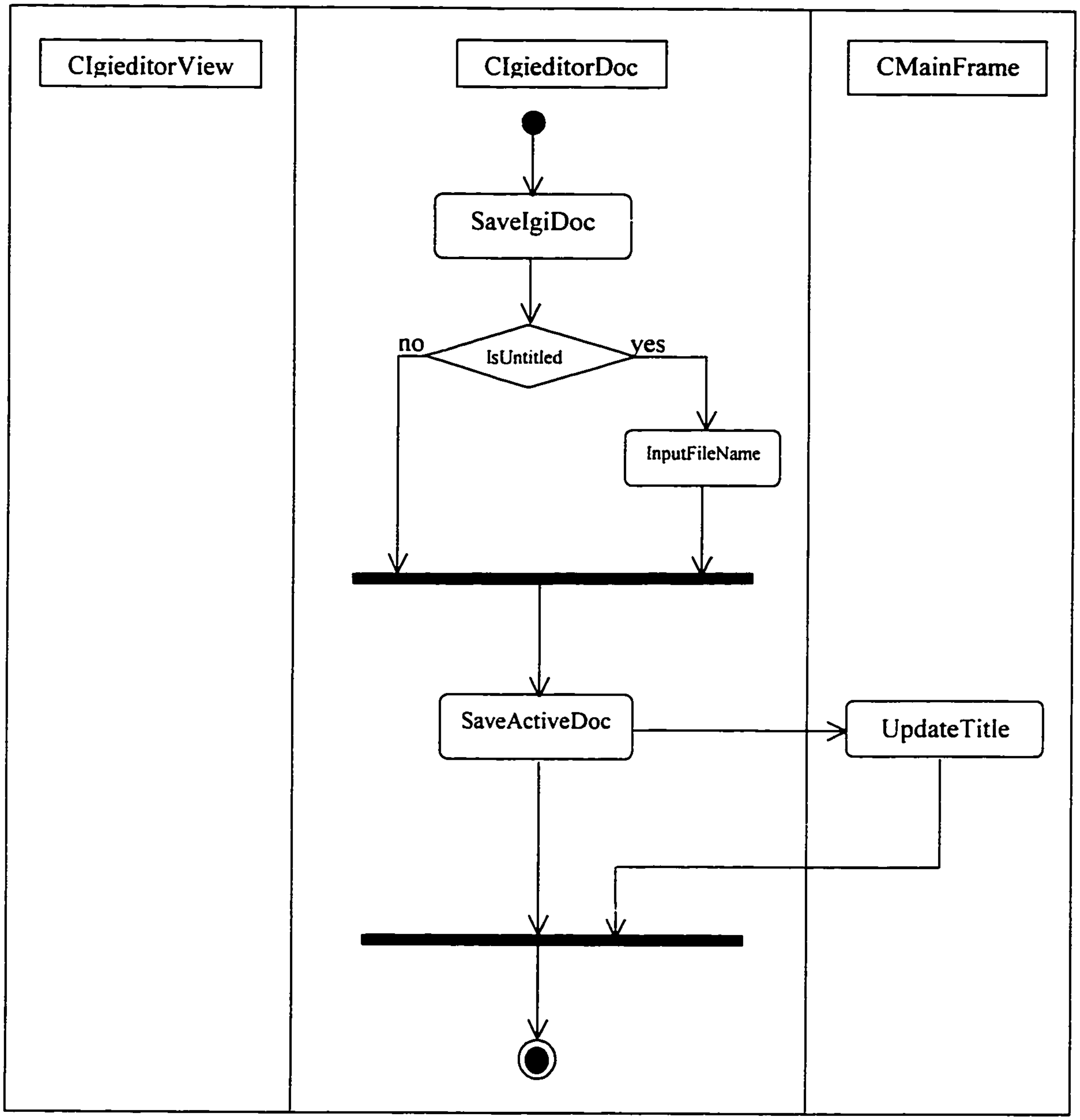


Figure 2-10 Active diagram of saving an active IGI document

2.3.4.4 User saves an active IGI document with a new name

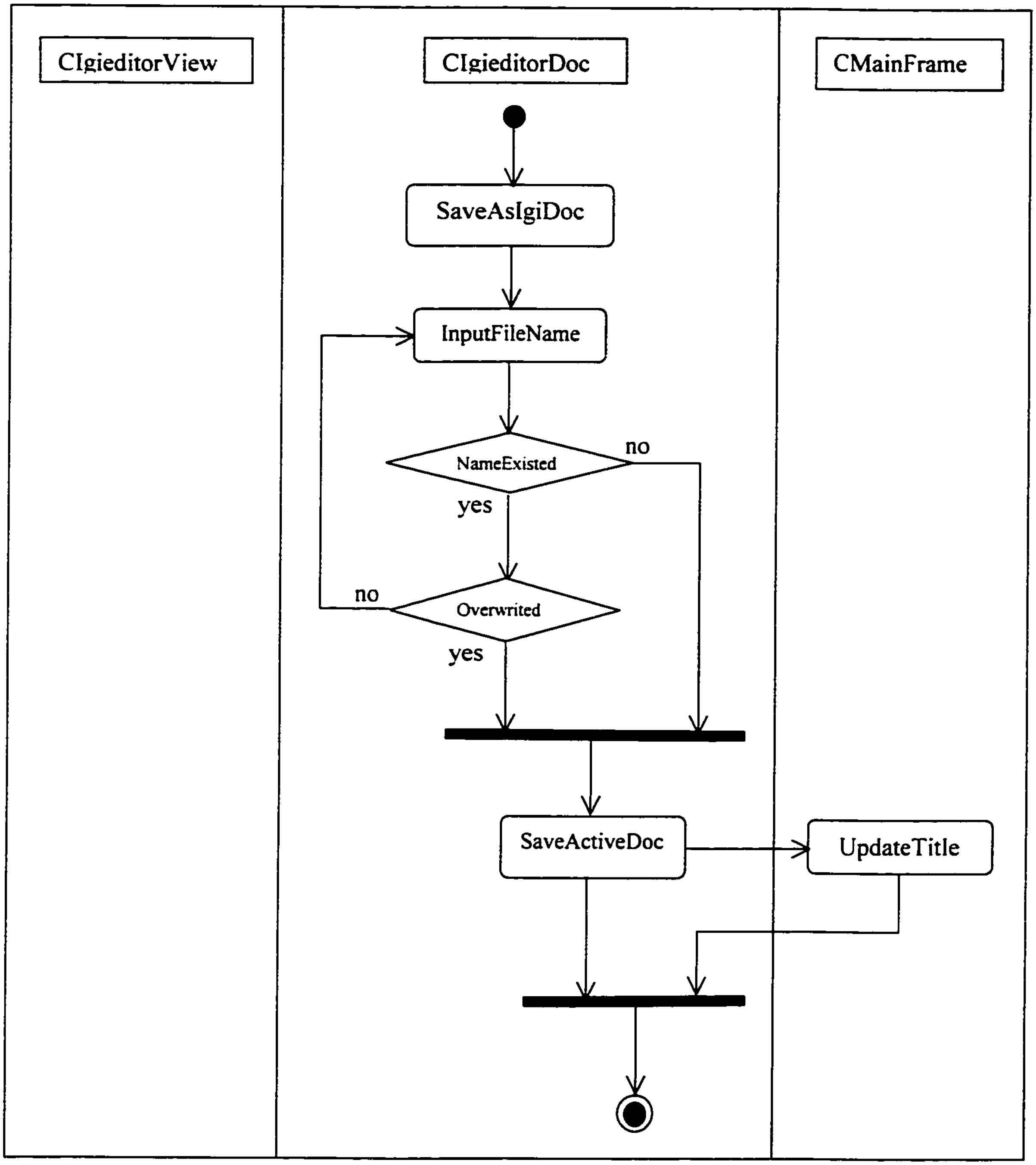


Figure 2-11 Active diagram of saving an active IGI document with a new name

2.3.4.5 User adds a primitive to an IGI

2.3.4.5.1 Adding a primitive Rectangle/Ellipse object to an IGI

In this active diagram, a primitive object includes Rectangle/Fill Rectangle and Ellipse/Fill Ellipse object.

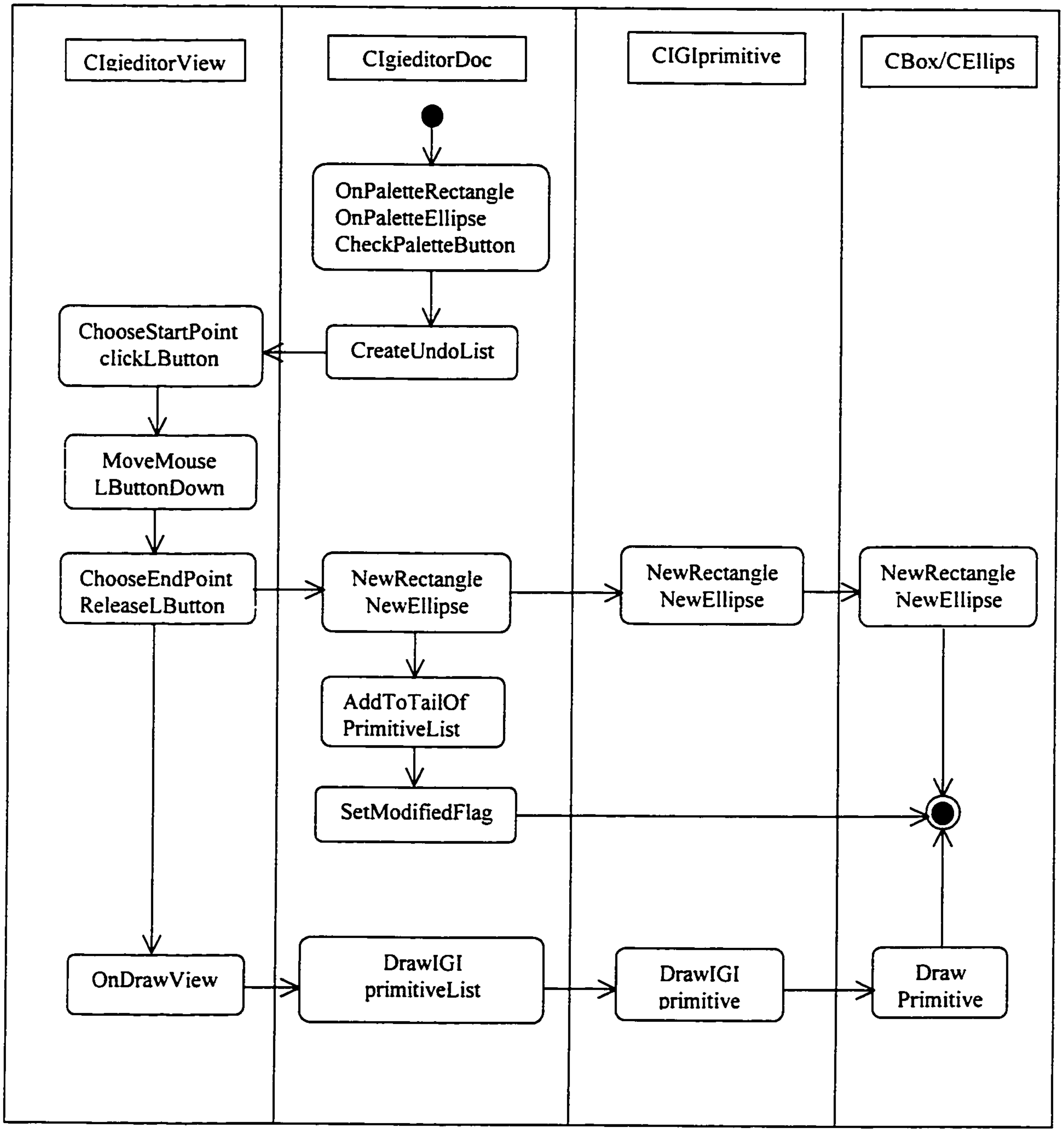


Figure 2-12 Active diagram of adding a Rectangle/Ellipse object

2.3.4.5.2 Adding a primitive Line object to an IGI

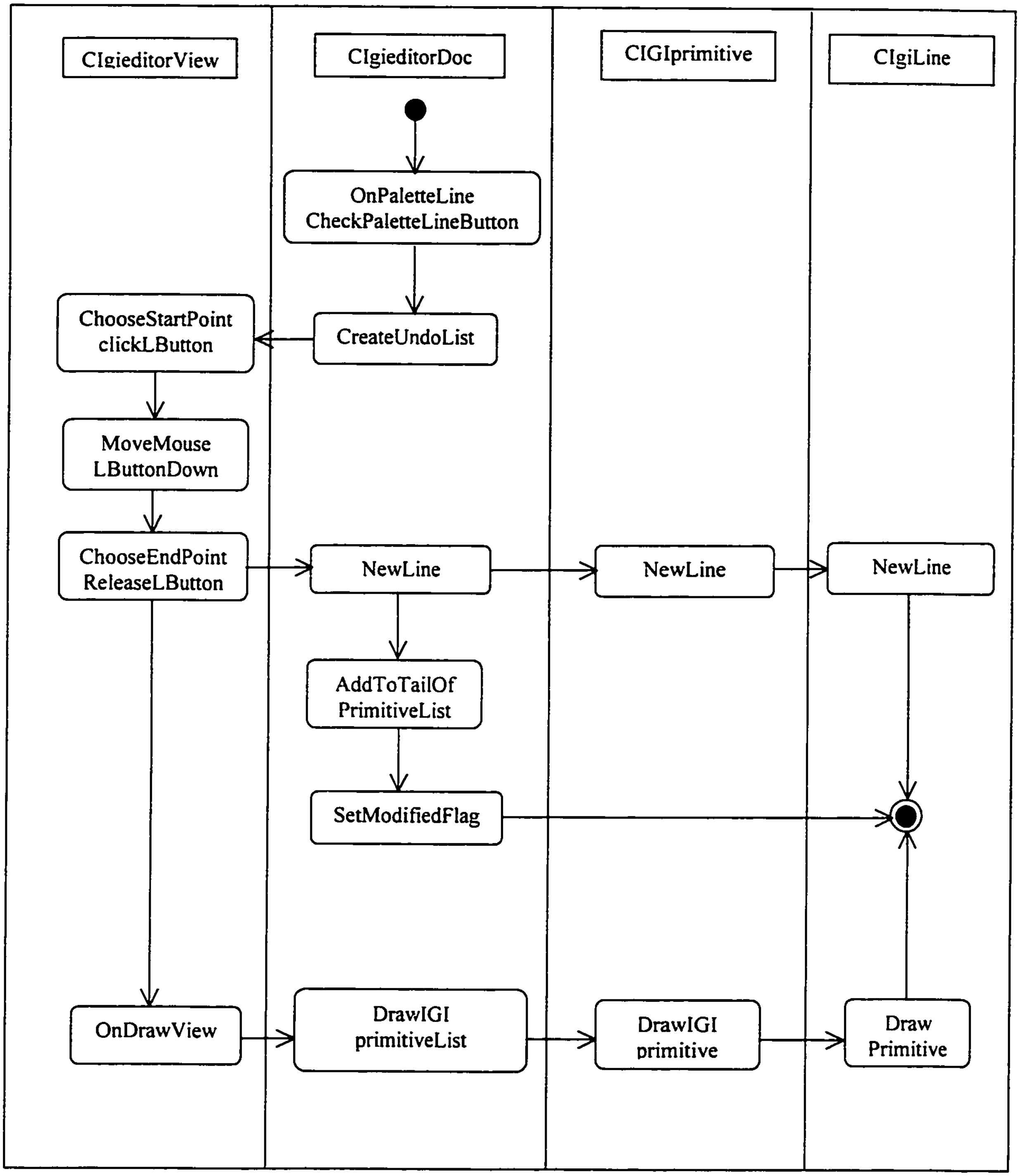


Figure 2-13 Active Diagram of adding a Line object

2.3.4.5.3 Adding a primitive Triangle object to an IGI

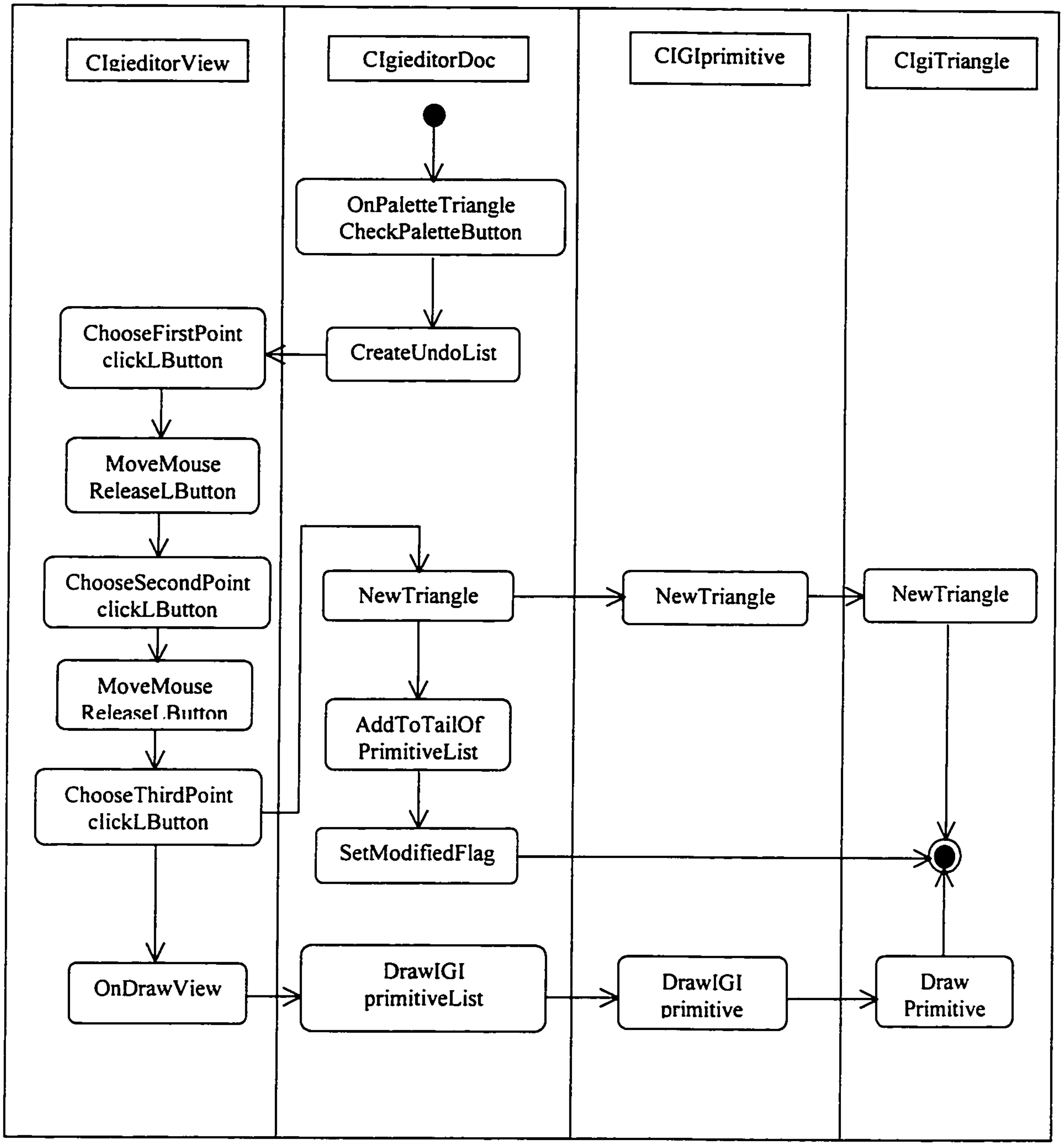


Figure 2-14 Active Diagram of adding a Triangle object

2.3.4.5.4 Adding a primitive Text object to an IGI

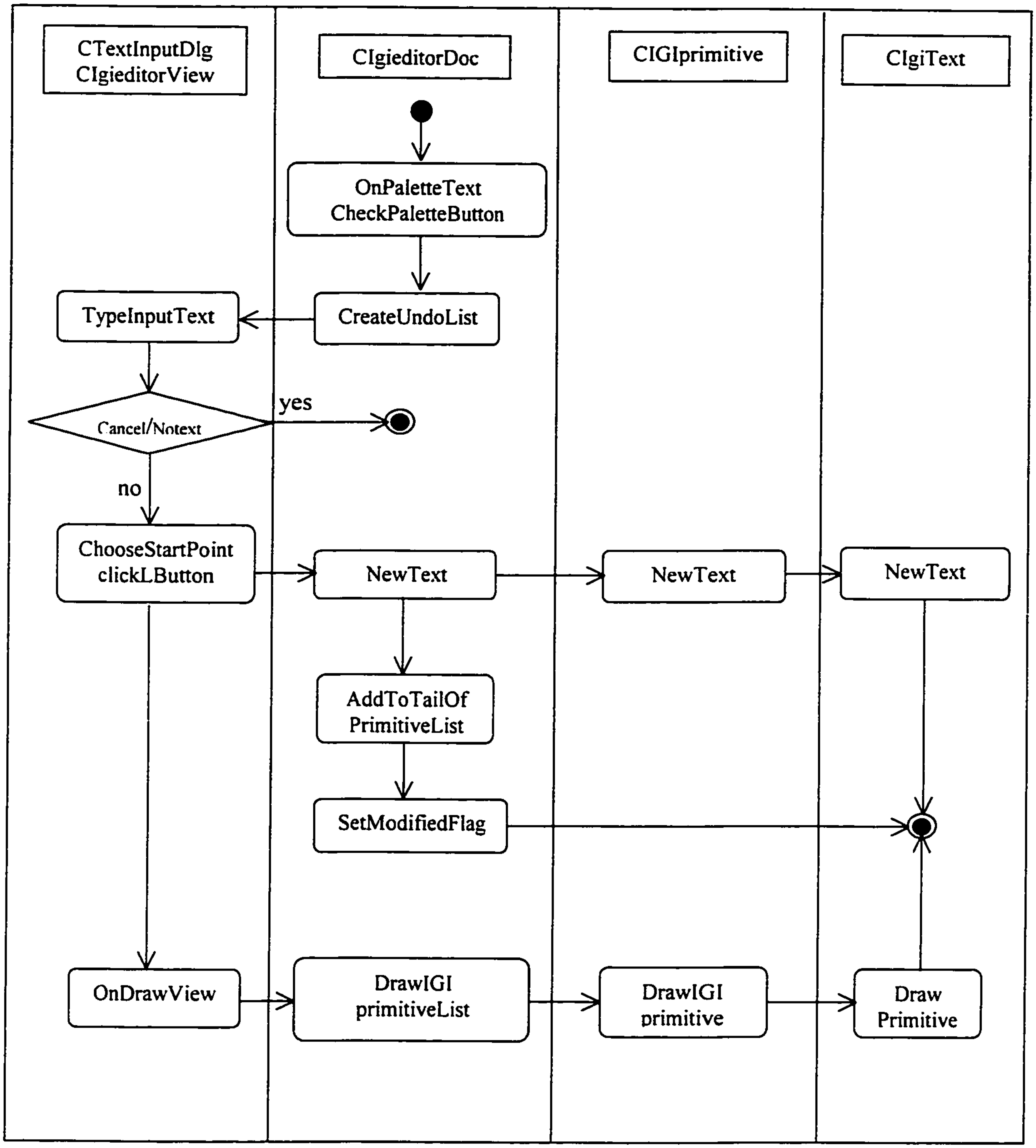


Figure 2-15 Active Diagram of adding a Text object

2.3.4.5.5 Adding a primitive TagName object to an IGI

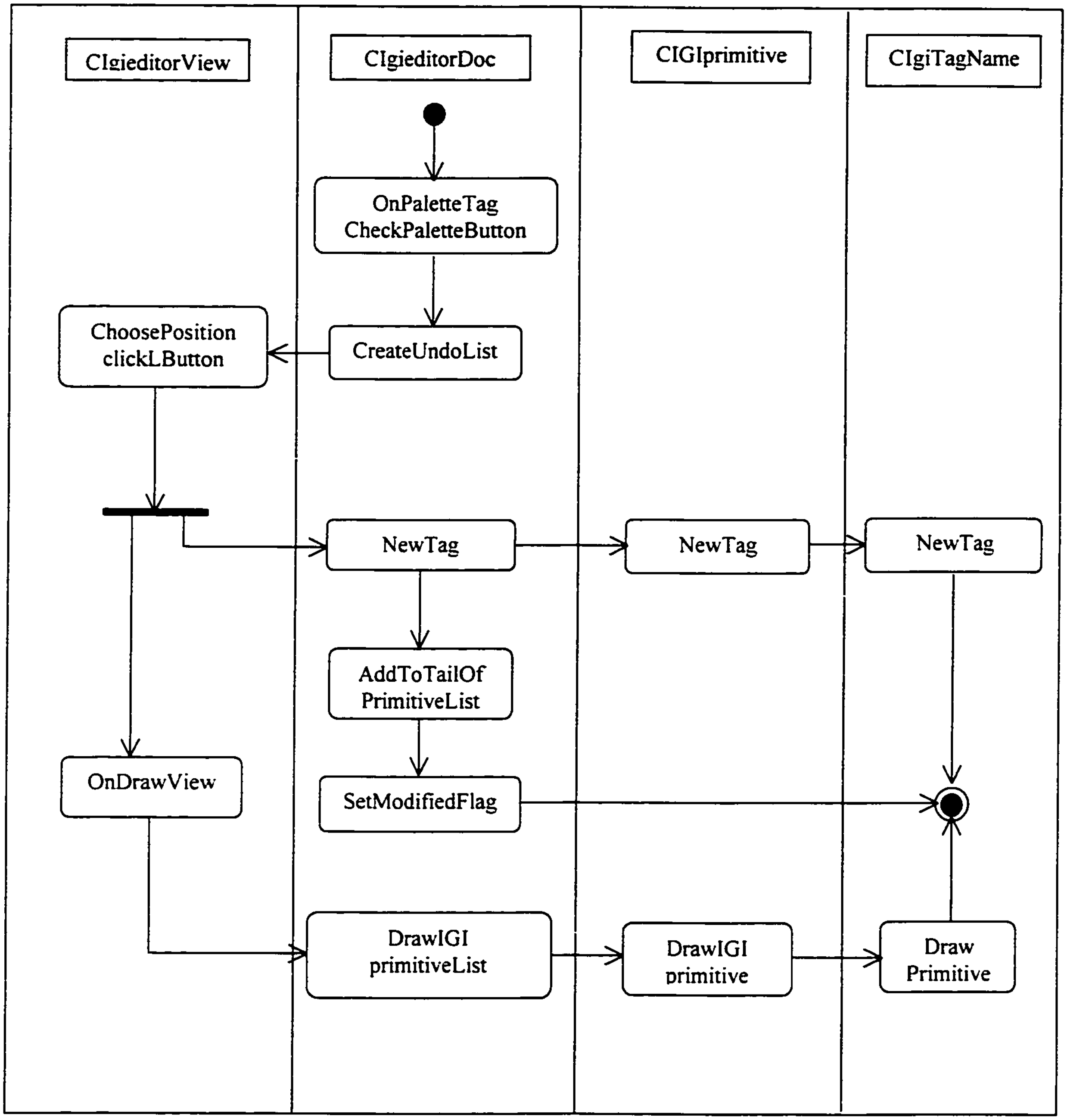


Figure 2-16 Active Diagram of adding a TagName object

2.3.4.5.6 Adding a primitive Stream object to an IGI

In this active diagram, a stream object includes process/logic/analog input/output stream object.

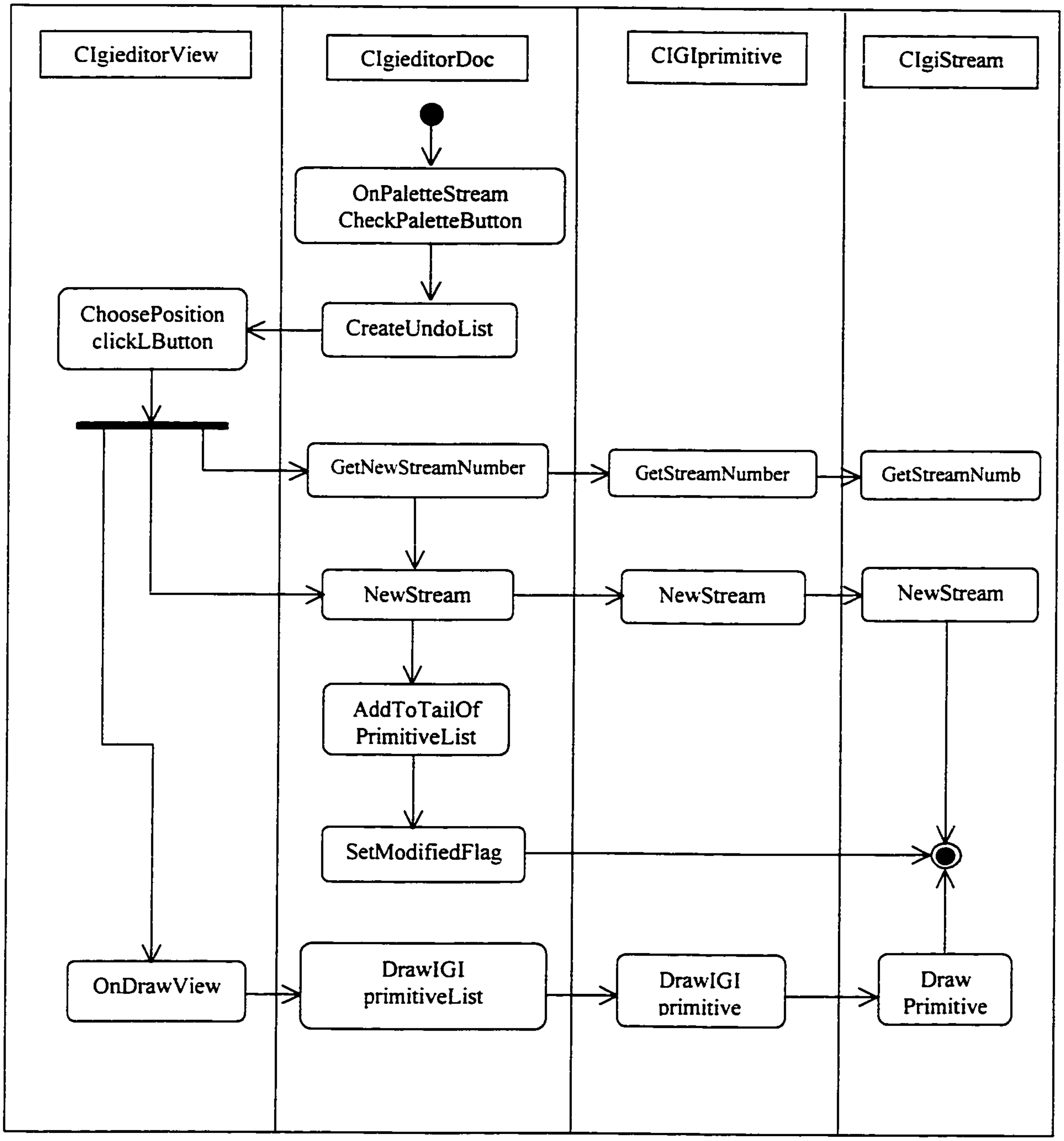


Figure 2-17 Active Diagram of adding a Stream object

2.3.4.6 User selects a primitive object

If a selected primitive object is a text or tagname, the activity that follows DrawPrimitive will be DrawTextMark that will draw a box around text. For other selected primitive objects, the activity DrawSquare will draw a small square on the control point of the object.

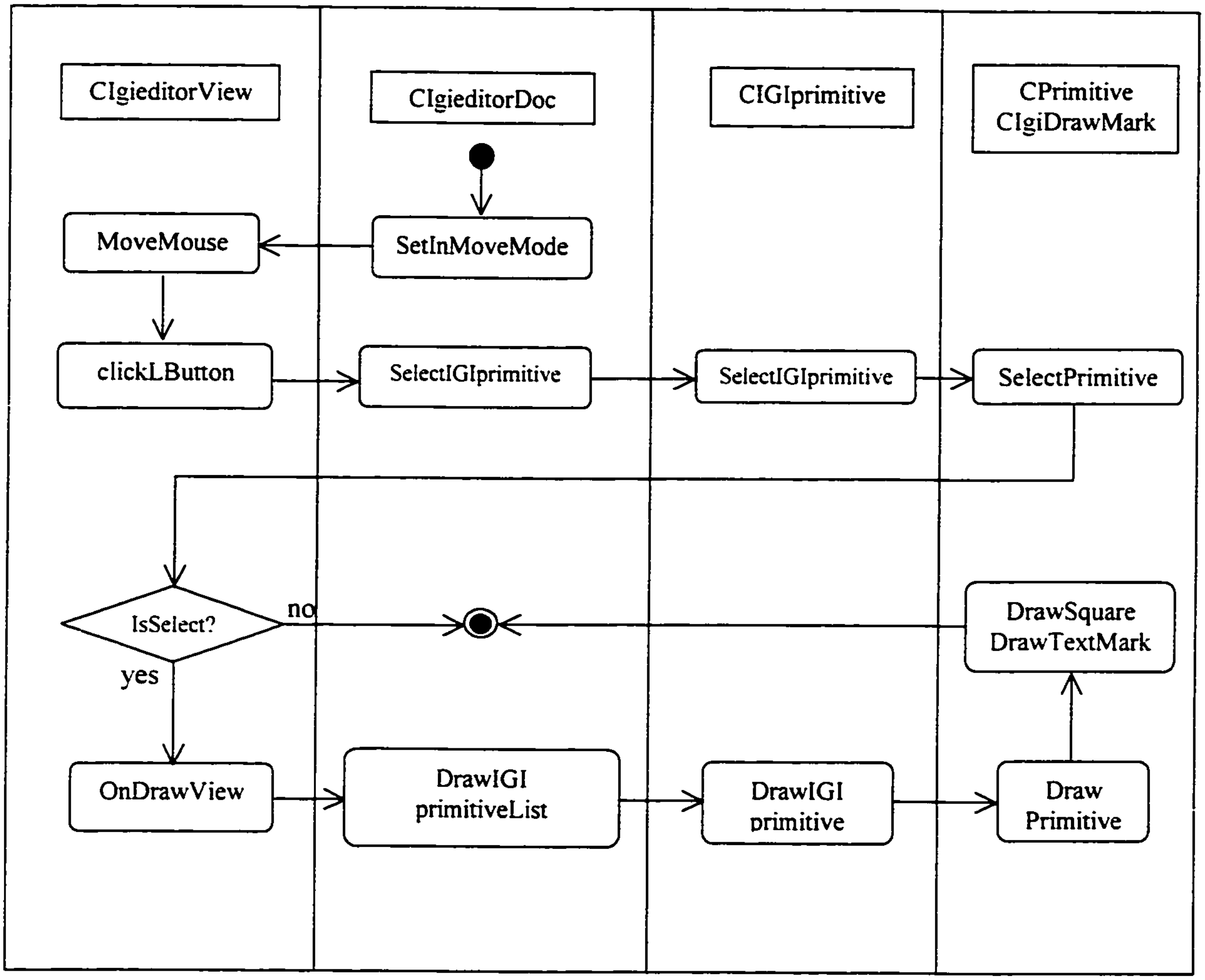


Figure 2-18 Active Diagram of selecting a primitive object

2.3.4.7 User moves the selected primitive(s)

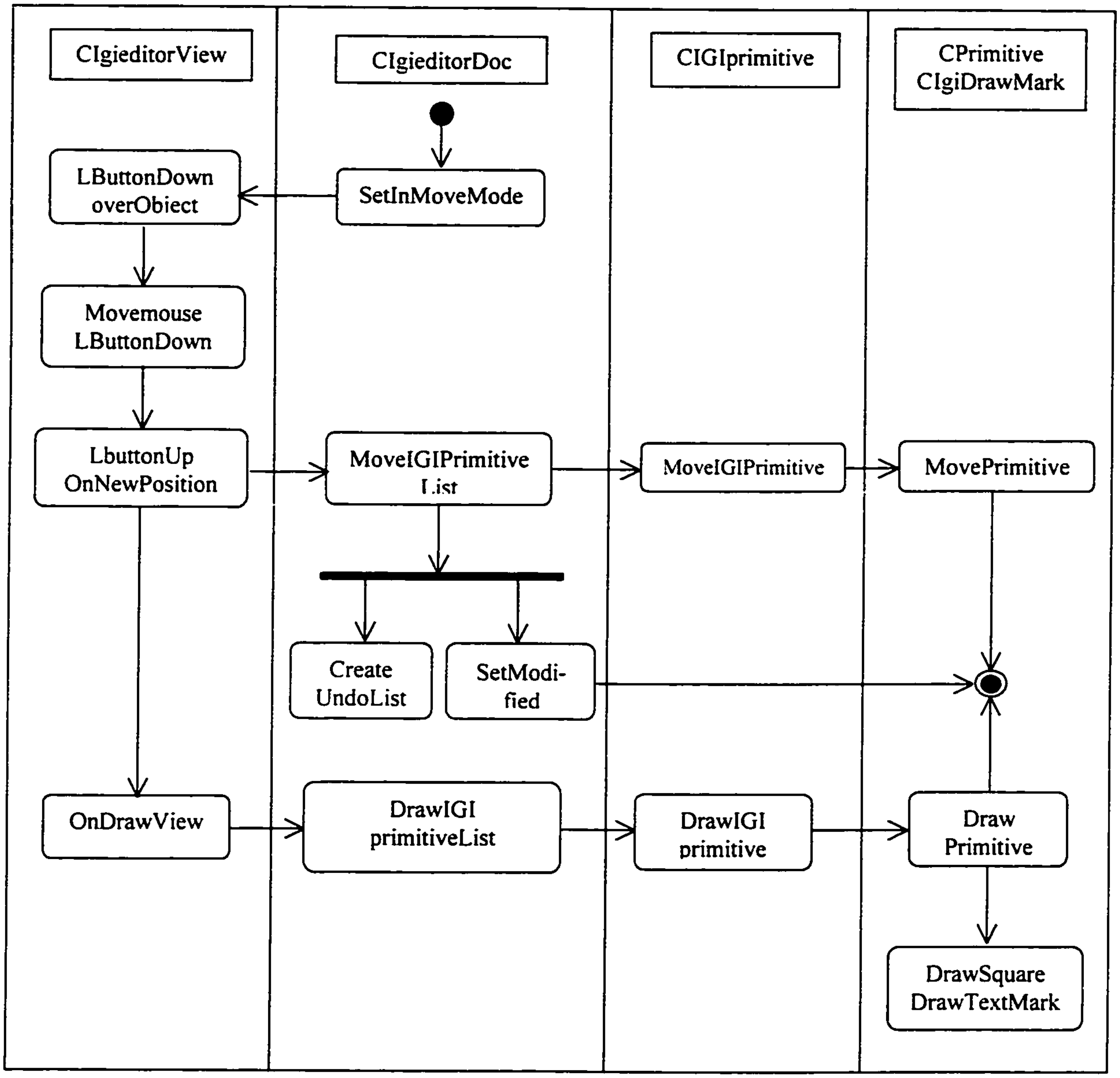


Figure 2-19 Active Diagram of moving the selected primitive(s)

2.3.4.8 User selects all primitive objects

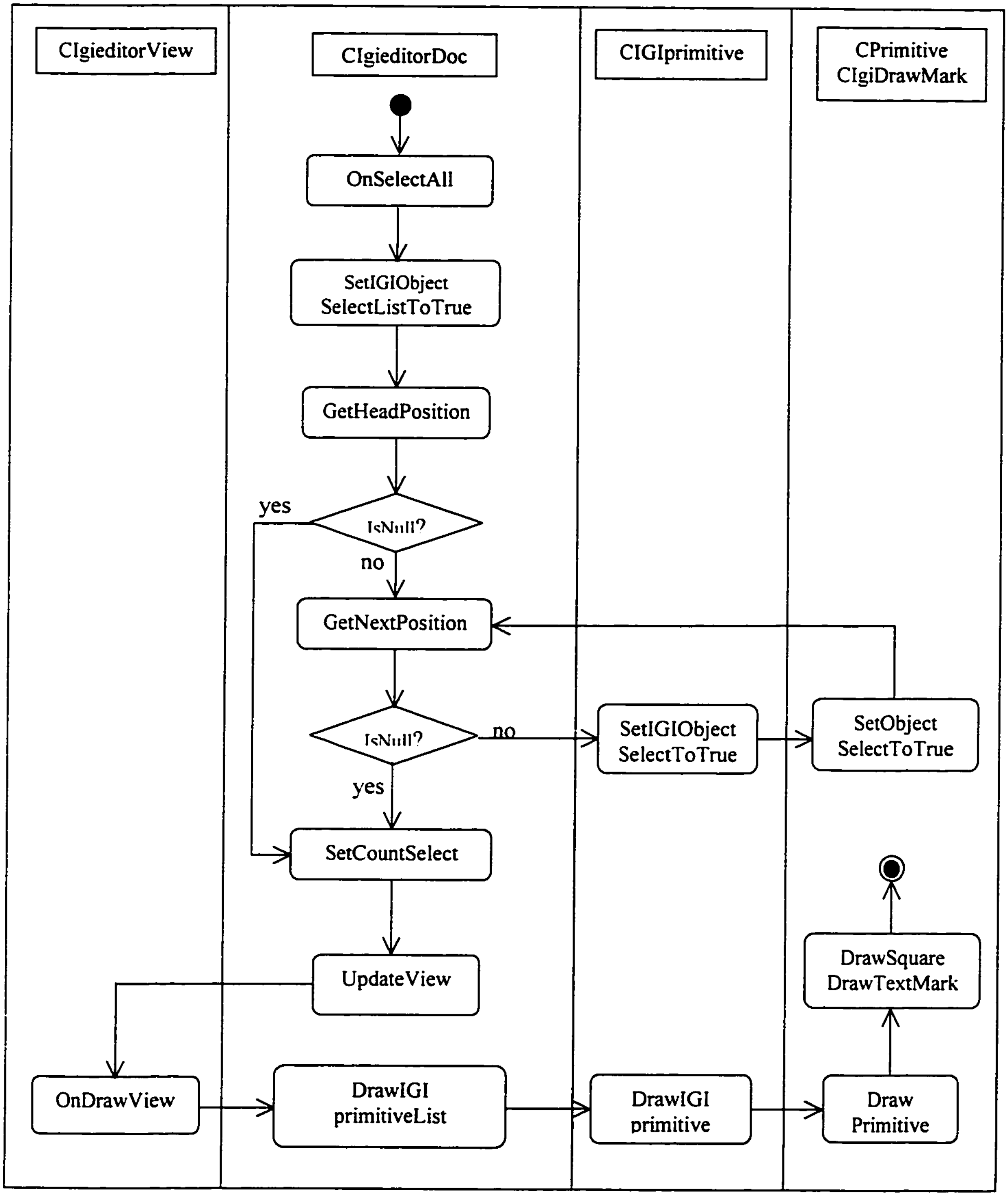


Figure 2-20 Active Diagram of selecting all primitive objects

2.3.4.9 User unselects all selected primitive object(s)

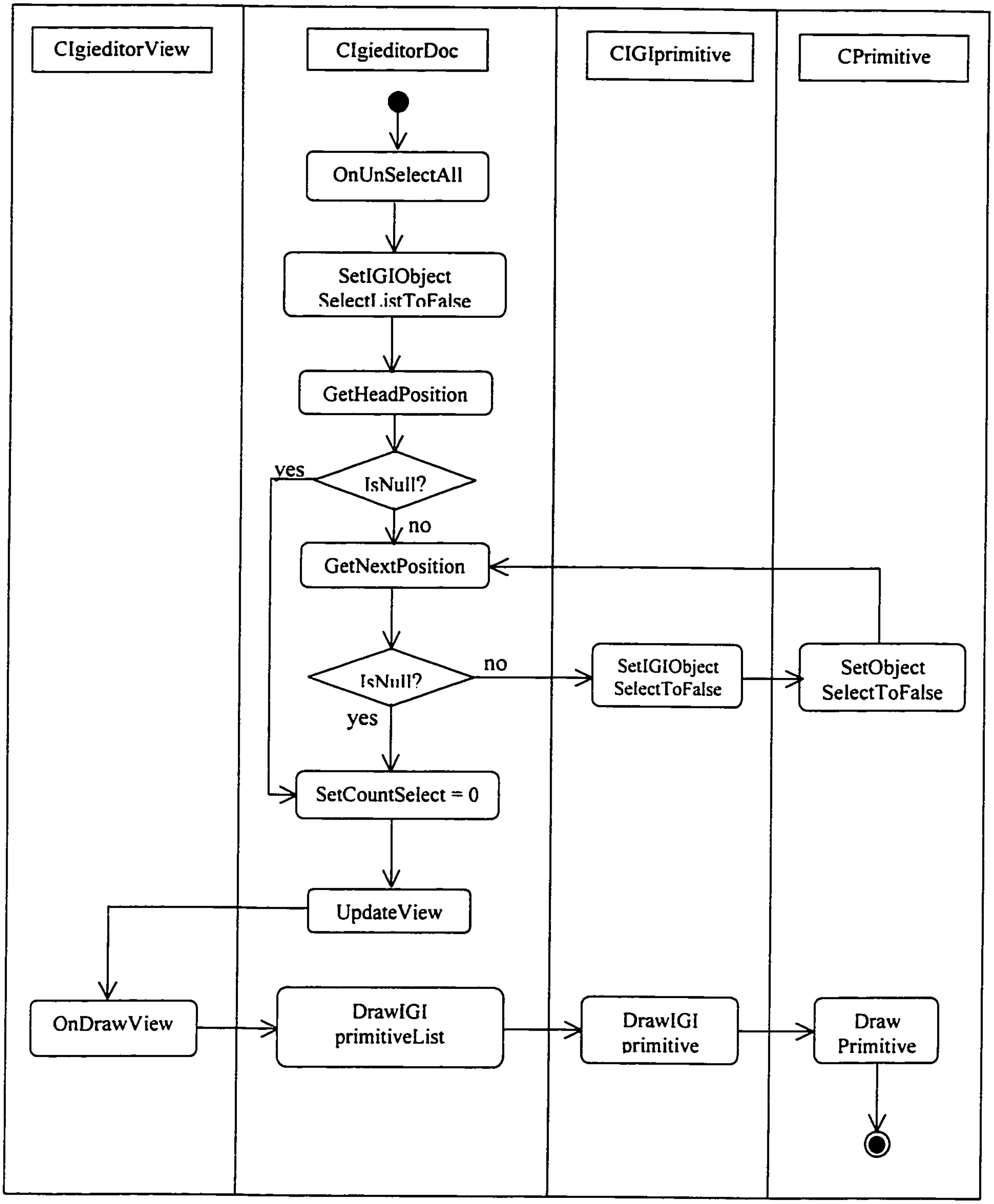


Figure 2-21 Active Diagram of unselecting all selected primitive object(s)

2.3.4.10 User cuts any selected primitive object(s)

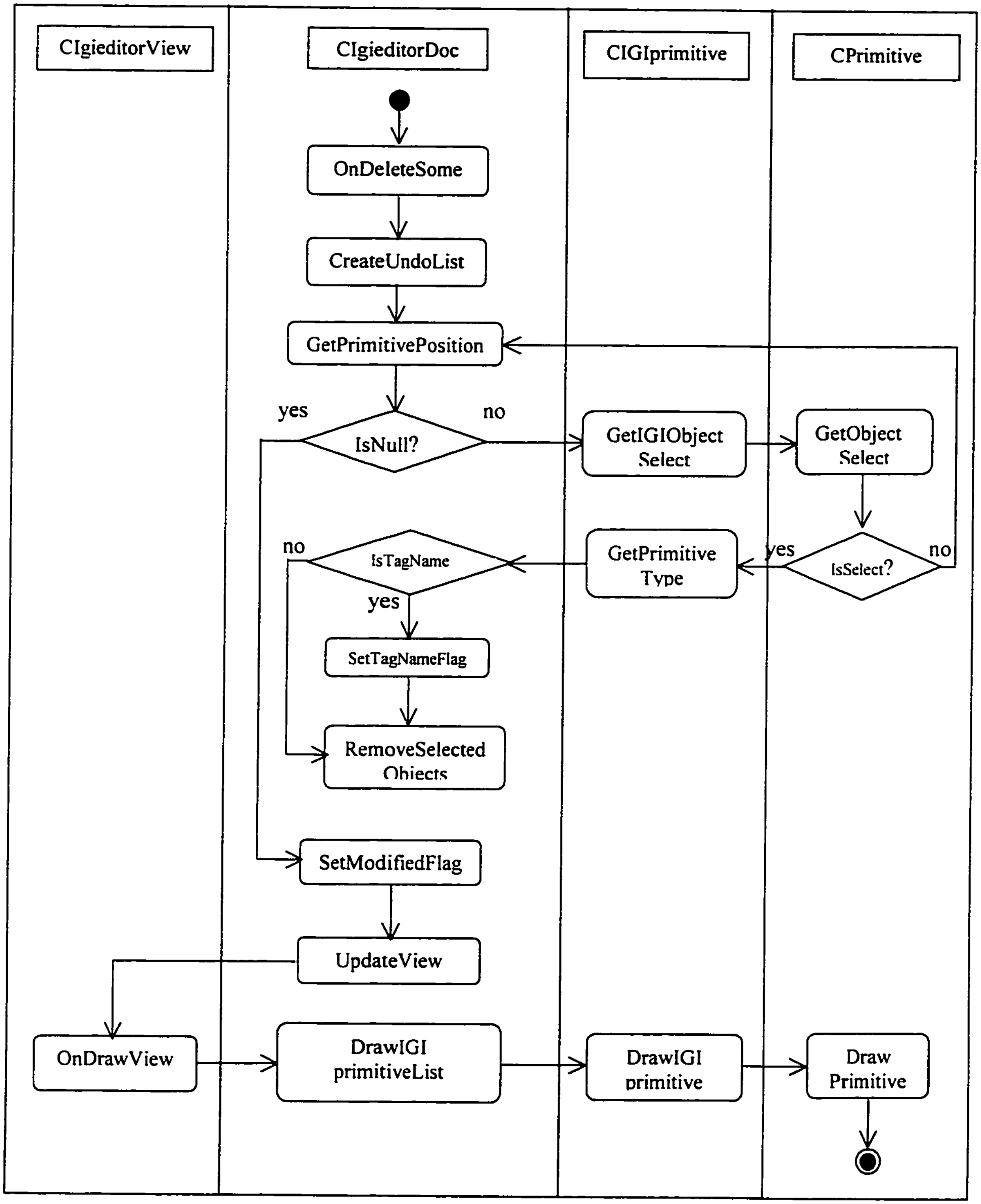


Figure 2-22 Active Diagram of cutting selected primitive object(s)

2.3.4.11 User cuts all primitive object(s)

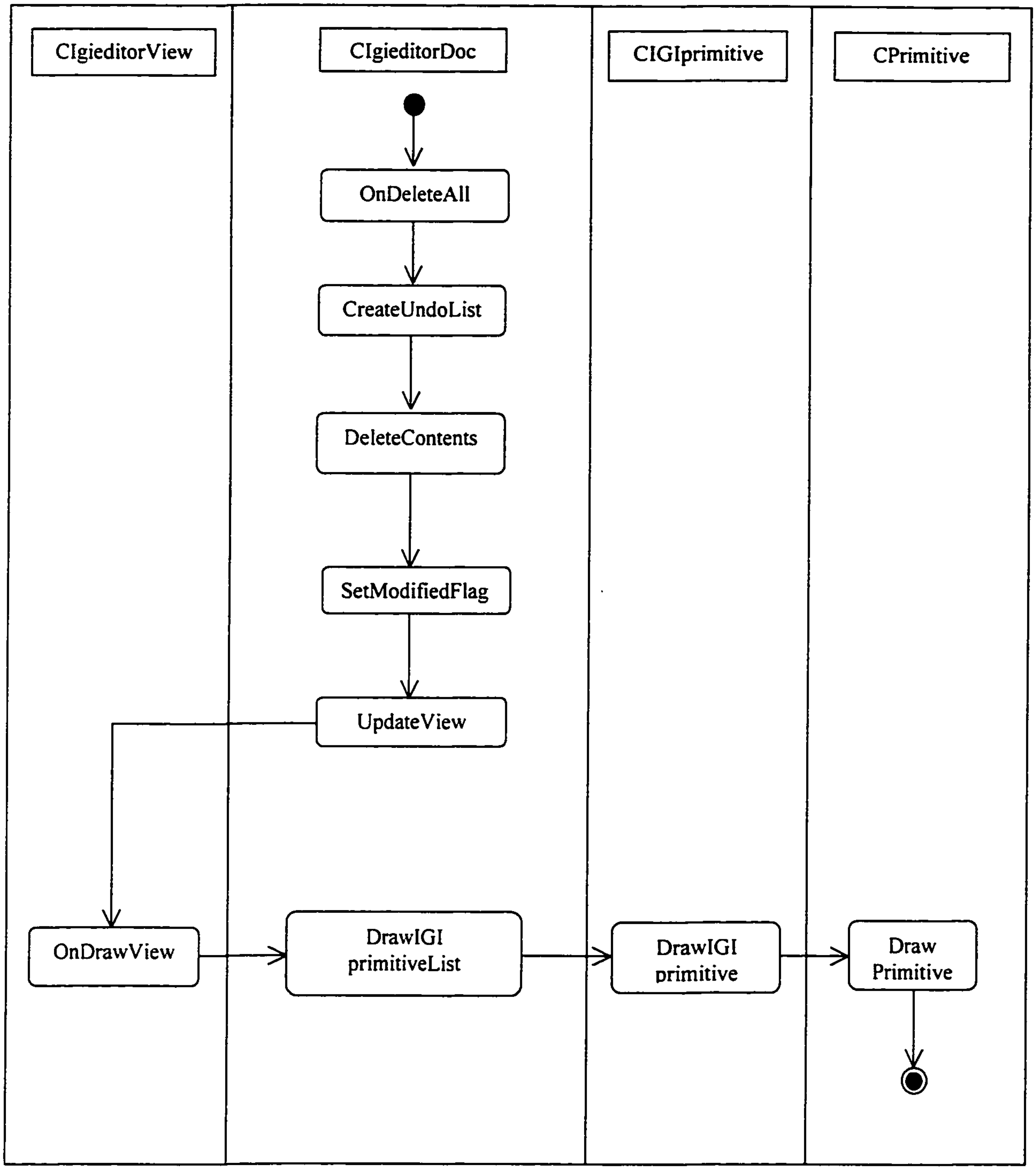


Figure 2-23 Active Diagram of cutting all primitive object(s)

2.3.4.12 User undoes the last edit action

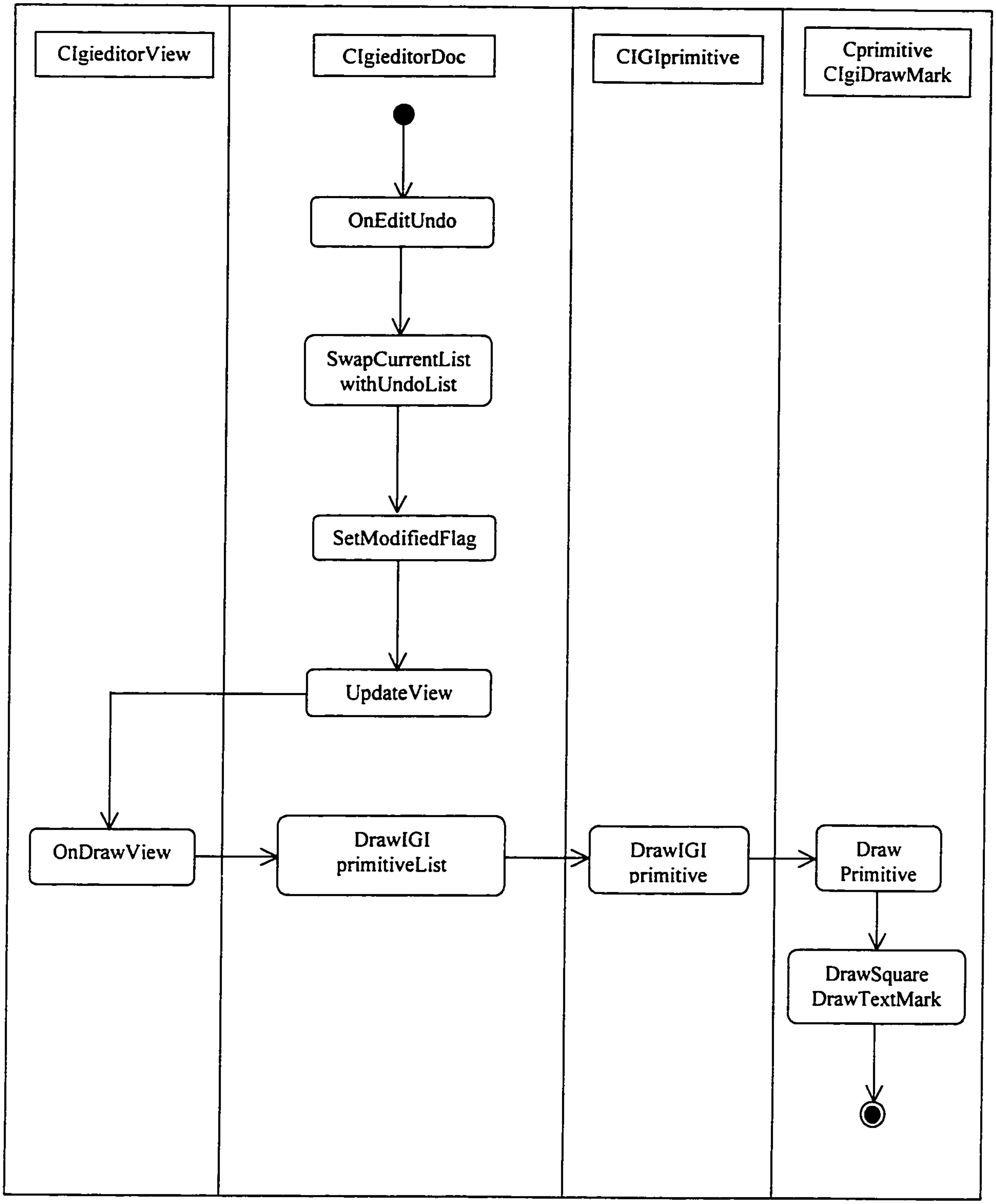


Figure 2-24 Active Diagram of undoing the last edit action

2.3.4.13 User pushes primitive object down

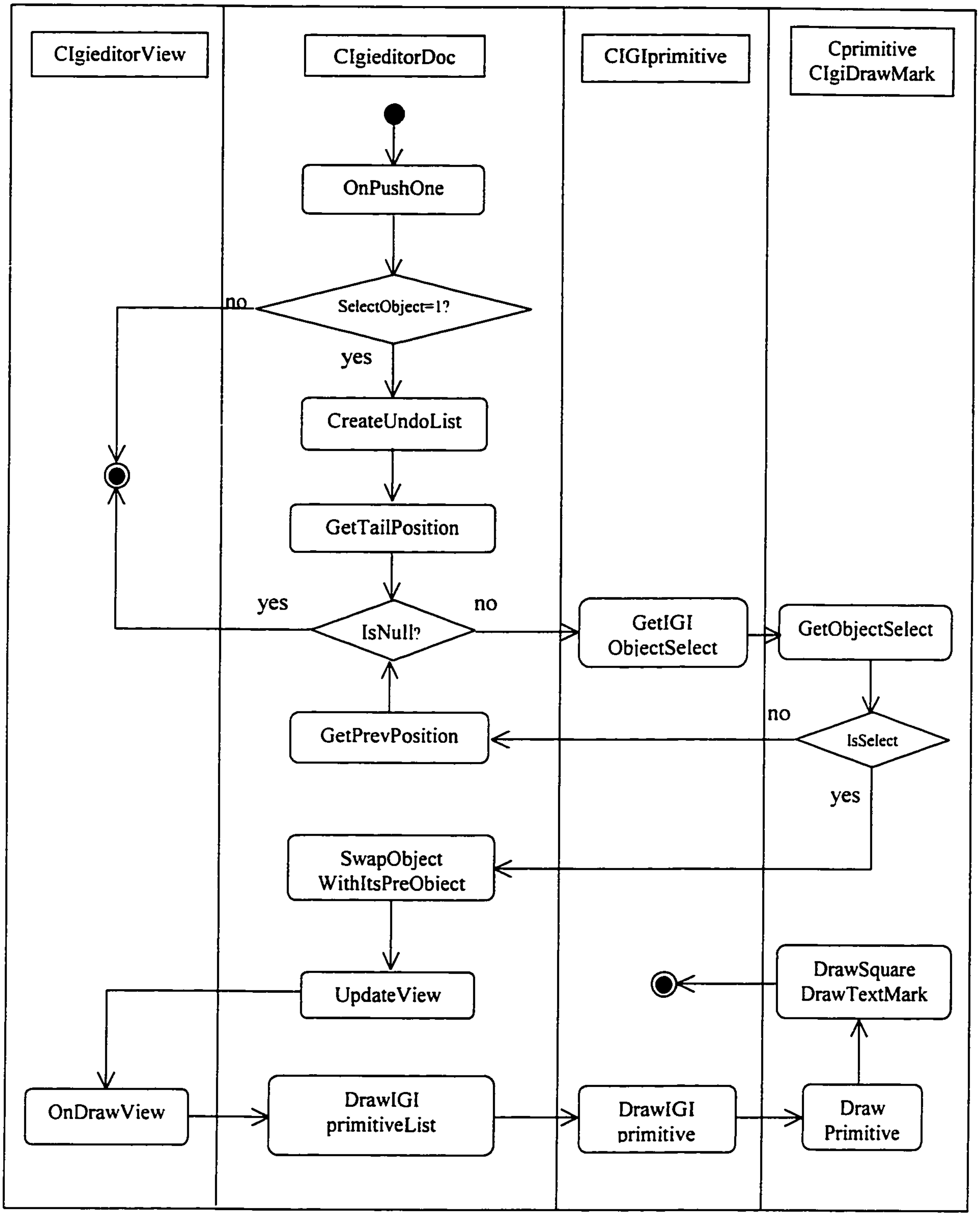


Figure 2-25 Active Diagram of pushing primitive object down

2.3.4.14 User pushes primitive object down to bottom level

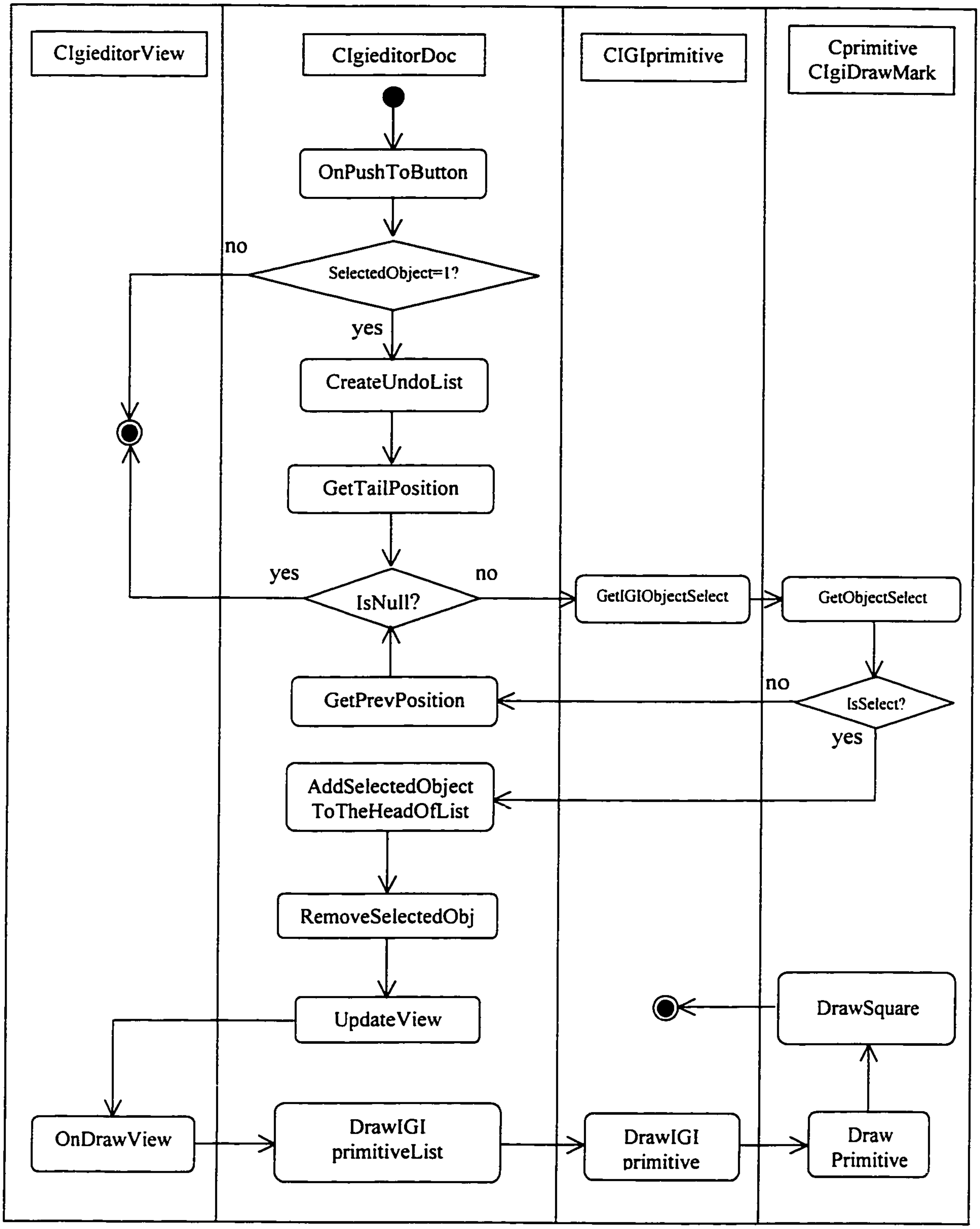


Figure 2-26 Active Diagram of pushing an object down to bottom level

2.3.4.15 User pops a primitive object up one level

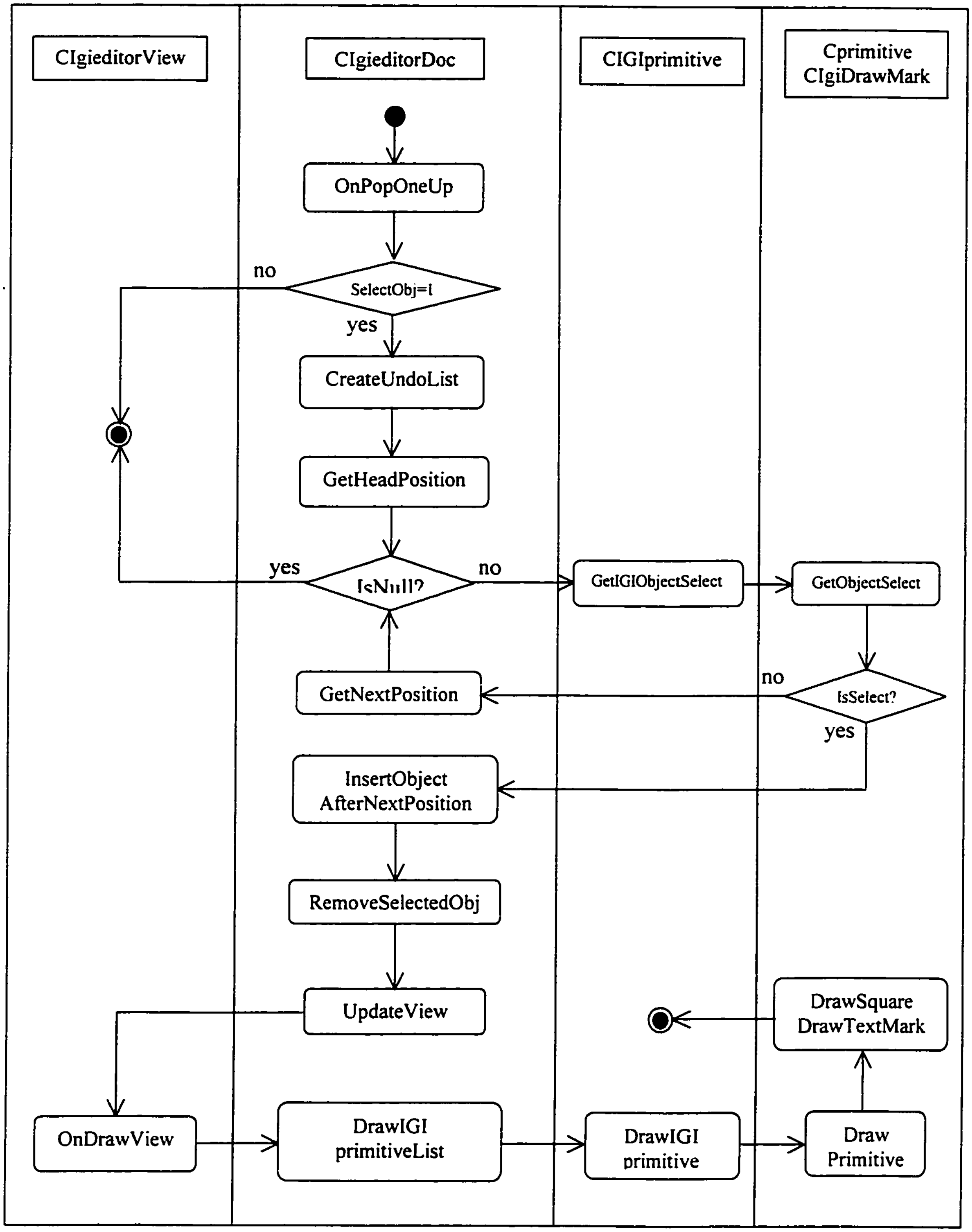


Figure 2-27 Active Diagram of popping an object up one level

2.3.4.16 User pops primitive object up to top level

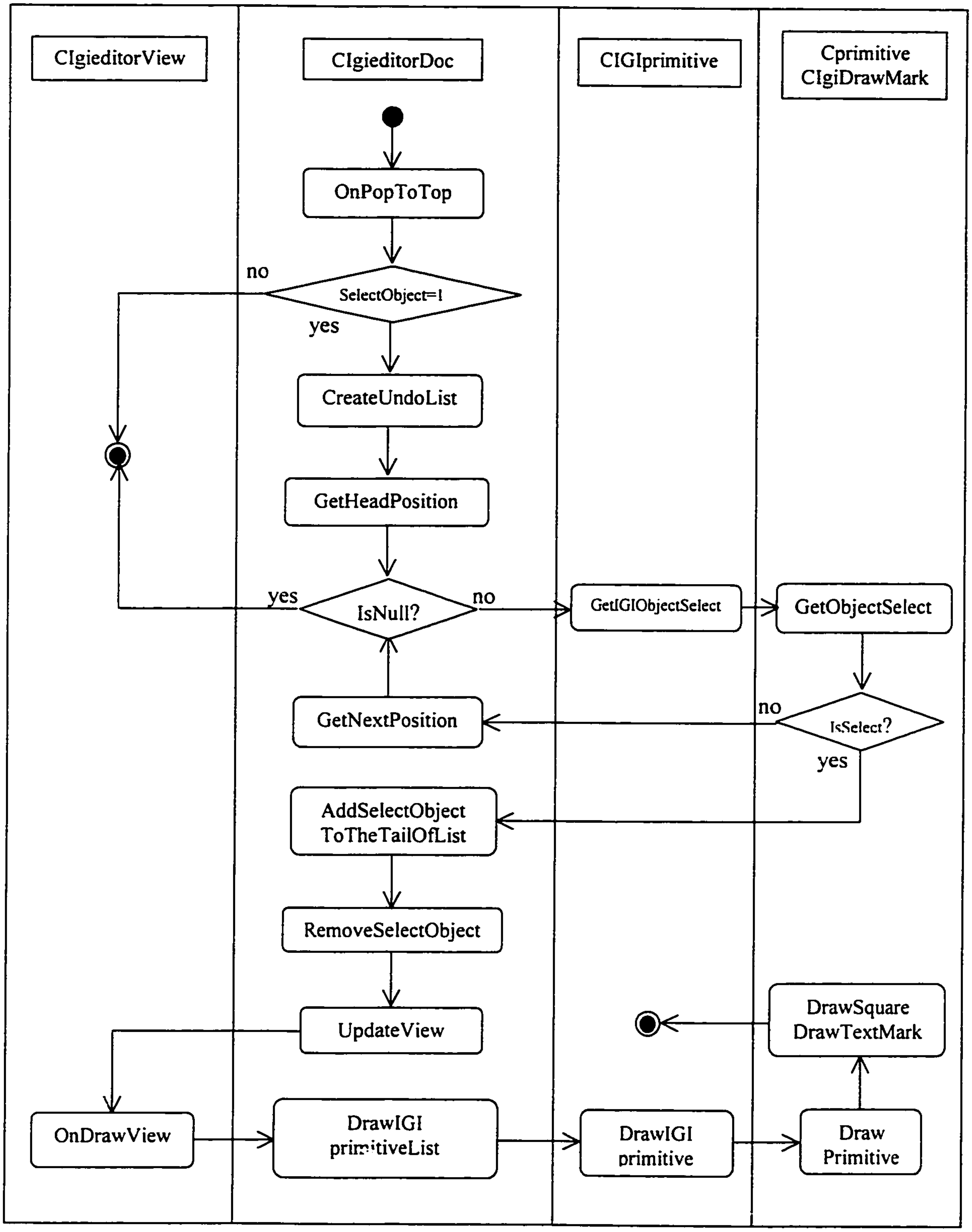


Figure 2-28 Active Diagram of popping an object up to top level

2.3.4.17 User copies any selected primitive object(s)

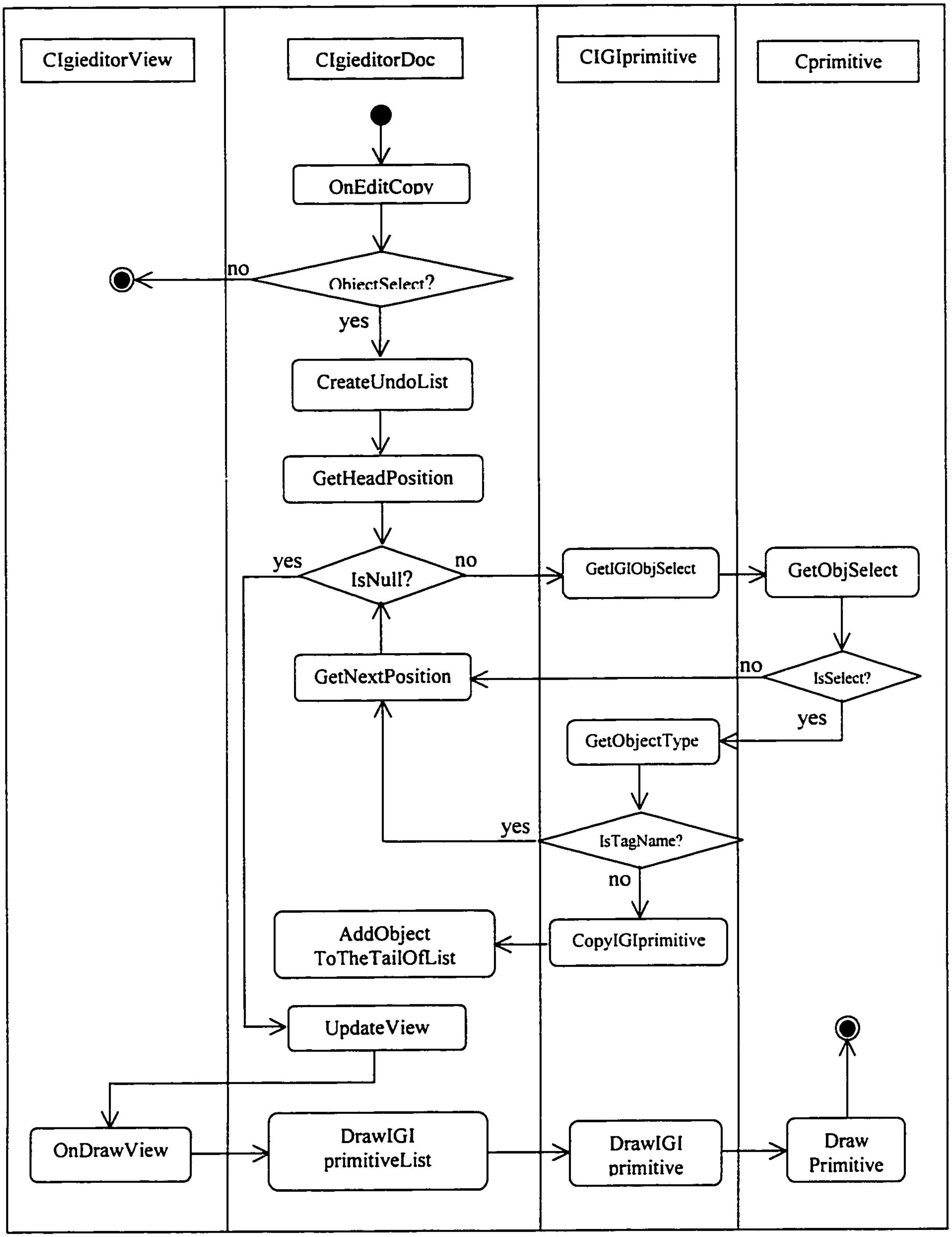


Figure 2-29 Active Diagram of copying selected primitive object(s)

2.3.4.18 User changes any selected primitive object(s) color

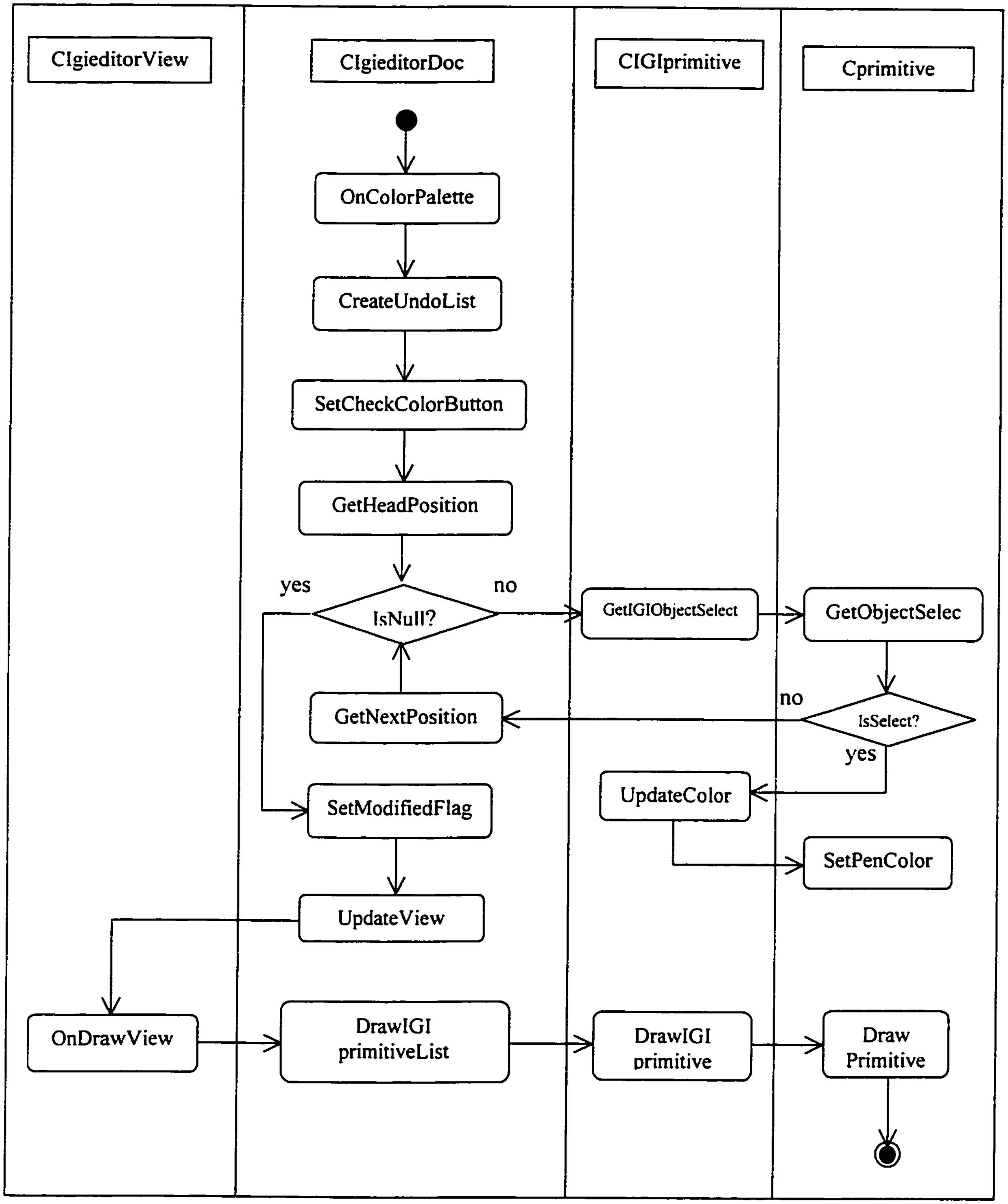


Figure 2-30 Active Diagram of changing selected object(s) color

2.3.4.19 User rotates a selected stream object

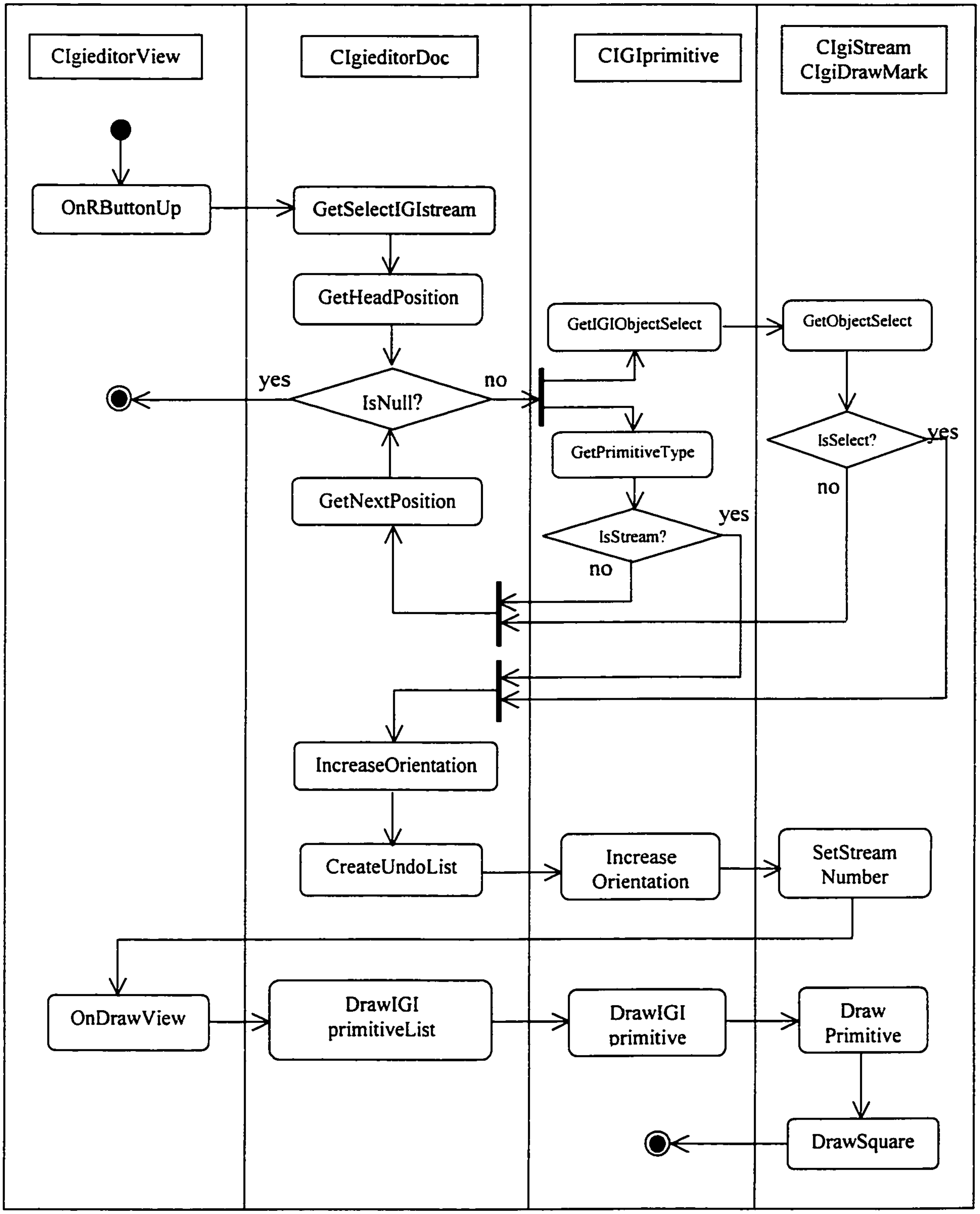


Figure 2-31 Active Diagram of rotating a selected stream object

2.3.4.20 User modifies a selected primitive object

Only the following primitive objects can be modified: Rectangle/Fill Rectangle, Ellipse/Fill Ellipse, Triangle/Fill Triangle, and Line.

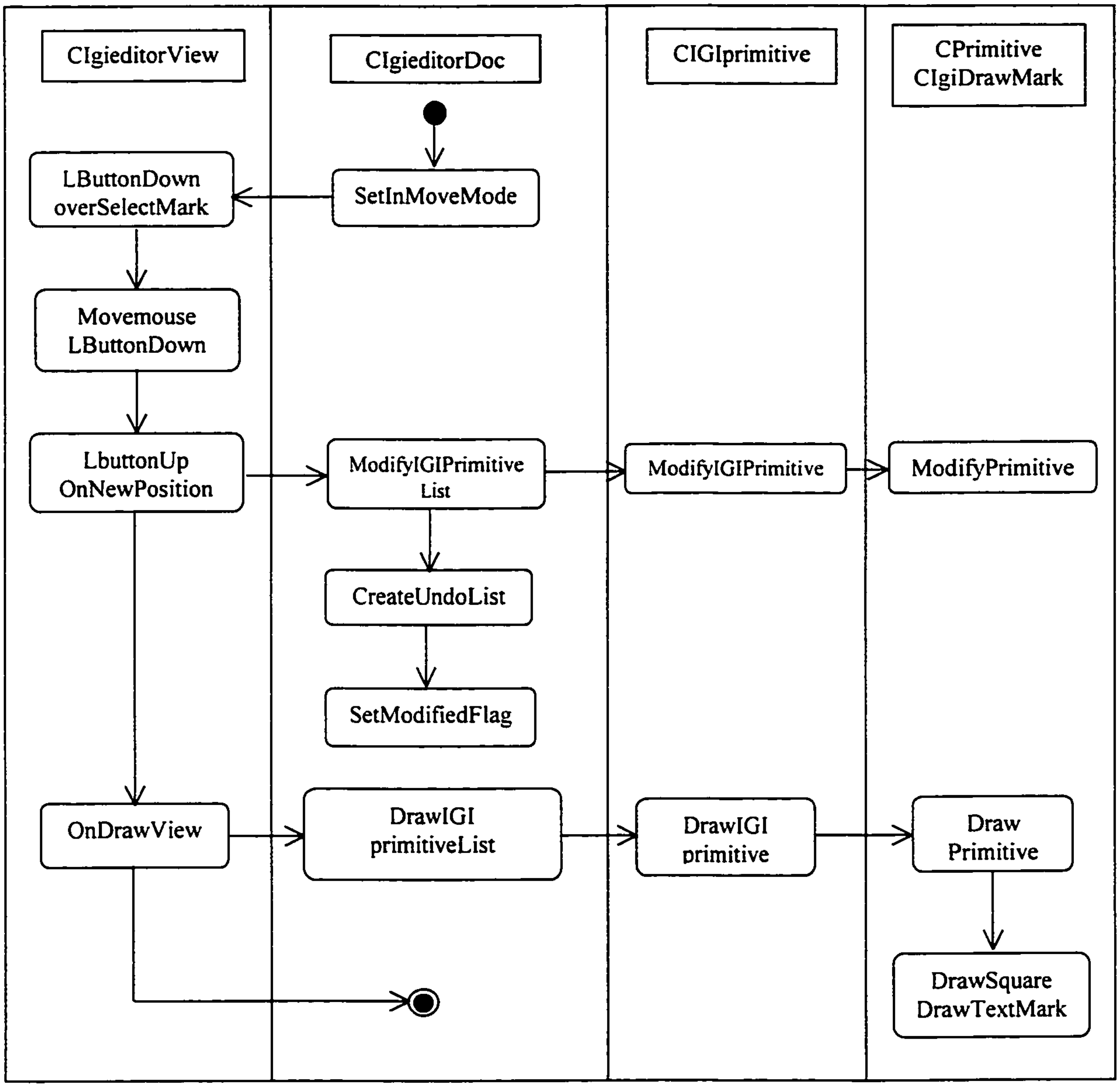


Figure 2-32 Active Diagram of modifying a selected object

2.3.5 Functional Model

In this section, a Data-Flow Diagram records the dependencies between each function of the system, and illustrates the data flows that are the Inputs/Outputs of the functions.

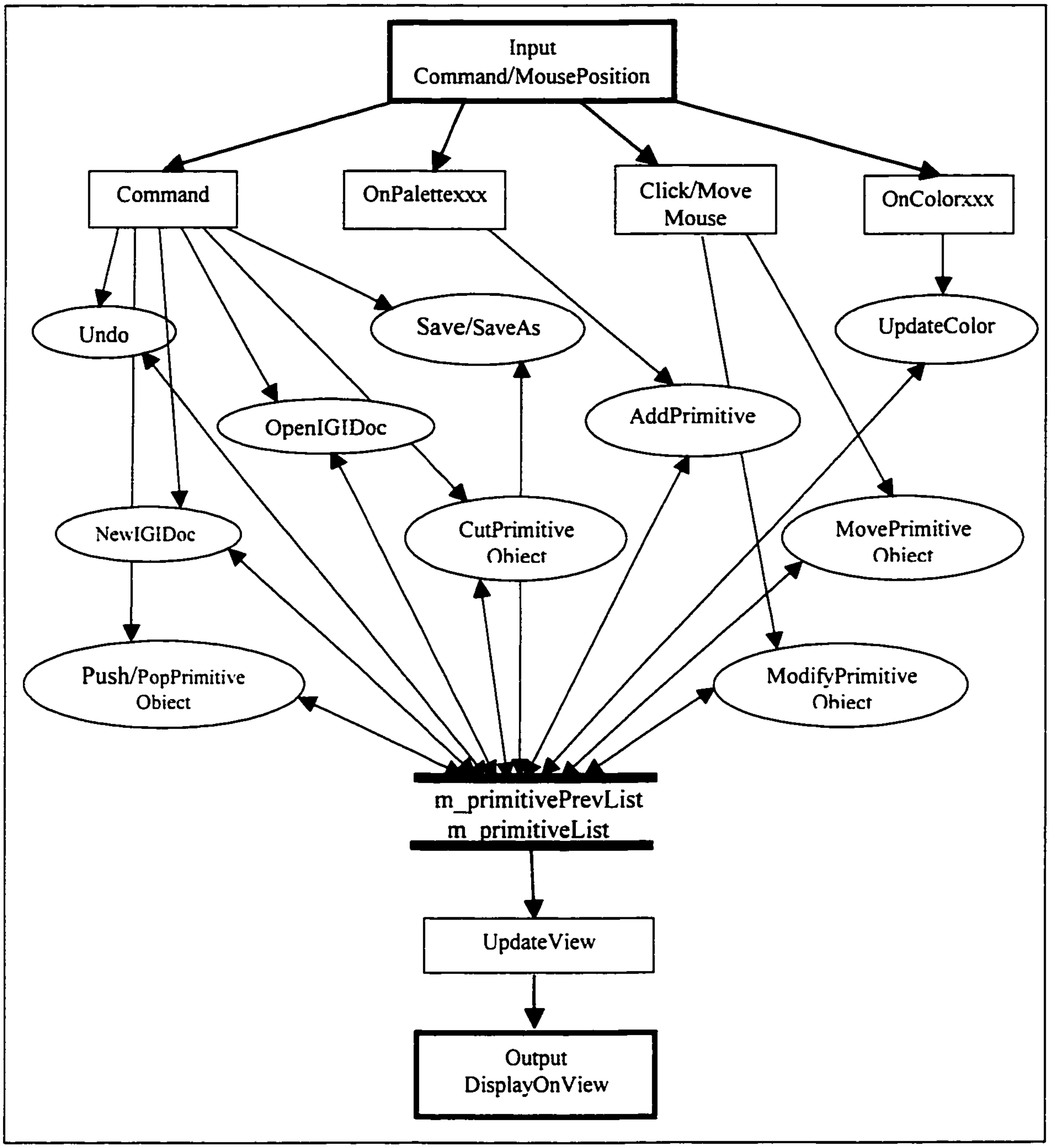


Figure 2-33 Data Flow Diagram

In the dataflow diagram, the inputs from user that clicks the mouse on the primitive palette and color palette or on the edit area include edit command, the primitive object type, pen color, edit mode, object filled mode, object select status and object position, etc.

The output is the primitive objects displayed on the edit view window according to the edit command.

All the edit commands (or functions of the system) are operated on the main data structure – a link list of the IGIprimitive node that records each primitive object's properties.

The data flows that are the Input/Outputs of the each function is described as following:

1. NewIGIDoc

Input: User clicks on the "New" command.
Function: OnNewDocument() is called.
Action: Current primitive object list "m_primitveList" is initialized to empty.
Output: Edit window is cleared.

2. OpenIGIDoc

Input: User clicks on the "Open" command and a file name is selected.
Function: OnOpenDocument(LPCTSTR lpszPathName) is called.
Action: The primitive objects in the file are loaded into the current primitive object list "m_primitveList".
Output: The objects in the "m_primitveList" are displayed on the edit window.

3. Save/SaveAs IGIDoc

Input: User clicks on the "Save" or "Save As" command and a file name is selected for "Save As" and for "Save" if it is necessary.
Function: Serialize(CArchive& ar) is called.
Action: The objects in the "m_primitveList" are written to the current or the given file.
Output: The current file is updated or a new file is created.

4. AddPrimitiveObject

Input: User clicks on one of the primitive button on the primitive palette. The edit mode, pen color, primitive type, primitive fill mode are decided. User clicks mouse on the view window to decide the object's position.
Function: OnPalettexxx(), NewIgiPrimitive(PalettePrimitive item, int x[6]) are called. Here, xxx is the primitive name depending on which button is selected.
Action: An IGIprimitve node is created and is added to the object list "m_primitveList".

Output: The new primitive object is displayed on the selected position in the view window.

5. CutPrimitiveObject

Input: User clicks on the “Cut” or “Cut All” command.
Function: OnDeleteSome() is called for “Cut” and OnDeleteAll() is called for “Cut All” command.
Action: All objects that are selected are removed from the object list “m_primitveList” for “Cut”. All objects are removed and “m_primitveList” is set to empty for “Cut All”.
Output: The selected object is removed from the view window for “Cut” and all the view window is cleared for the “Cut All”.

6. MovePrimitiveObject

Input: User drag a selected primitive object and drop it to a new desired position.
Function: MoveIGIPrimitiveList(int dx, int dy) is called.
Action: The selected object's position in the “m_primitveList” is updated.
Output: The selected object is displayed at a new desired position.

7. ModifyPrimitiveObject

Input: User clicks mouse on a select mark of a primitive object and change it to a new desired size.
Function: ModifyIGIPrimitive(CIGIprimitive* p, int dx, int dy, int index) is called.
Action: The selected object's size in the “m_primitveList” is updated.
Output: The selected object is displayed with its new desired size.

8. Push/PopPrimitiveObject

Input: User clicks on “Push Down/Push To Button” command or “Pop Up/Pop To Top” command.
Function: OnPushOne() or OnPushBottom() is called for “Push Down/Push To Button”. OnPopOneUp() or OnPopToTop() is called for “Pop Up/Pop To Top” command.
Action: OnPushOne() will swap the selected object node with its previous node in the “m_primitveList”.
OnPopOneUp()will swap the selected object node with its next node in the “m_primitveList”.
OnPushBottom() will move the selected object node to the head of the “m_primitveList”.
OnPopToTop() will move the selected object node to the tail of the “m_primitveList”.
Output: The objects in the “m_primitveList” are redrawn with the new order.

9. UpdateColor

Input: User clicks on one of color button on the color palette.
Function: OnColorxxx(), UpdateColor() are called. Here, xxx is the color name depending on which button is selected.
Action: The colors of the objects that are selected are changed to the selected color in the object list "m_primitveList".
Output: The selected object is redrawn with the new desired color.

10. Undo

Input: User clicks on "Undo" command.
Function: OnEditUndo(), SwapList() are called.
Action: SwapList() will swap two link lists "m_primitivePrevList" and "m_primitveList".
Output: The objects in the "m_primitivePrevList" are displayed on the view window.

2.4 Interface Specification (IS)

In this section, the interface specification of the system is listed that shows everything that a designer, programmer, or tester needs to know to use the design entities that make up the system. The interfaces include the exported types and specify operations.

2.4.1 Exported Types

In IGI Editor program, we defined the following variable types:

1. Color Palette Index

```
typedef enum{
        Black,Blue,Green,Cyan,Red,Magenta,Yellow,White,
        Darkblue,Darkgreen,Darkcyan,Darkred,Darkmagenta,
        Darkyellow,Darkgray,Lightgray
        } ColorIndex;
```

2. Primitive Palette Index

```
typedef enum{
        IgiRectangle, FillRectangle,
        IgiTriangle, FillTriangle,
        IgiEllipse, FillEllipse,
        IgiText, Tag,
        ProcessIn, ProcessOut,
        LogicIn, LogicOut,
        AnalogIn, AnalogOut,
        IgiLine, Indicator
        } PaletteIndex;
```

3. Edit Mode Index

```
typedef enum {
        ADD_PRIMITIVE_MODE,
        MOVE_MODE,
} EditModeIndex;
```

4. Palette Primitive Item Index

```
typedef enum {
        IGI_PRIM_NONE,
        IGI_PRIM_ELLIPSE,
        IGI_PRIM_BOX,
        IGI_PRIM_LINE,
        IGI_PRIM_TRIANGLE,
        IGI_PRIM_TAG,
        IGI_PRIM_TEXT,
        IGI_PRIM_INPUT_ANALOG_STREAM,
        IGI_PRIM_OUTPUT_ANALOG_STREAM,
        IGI_PRIM_INPUT_LOGIC_STREAM,
        IGI_PRIM_OUTPUT_LOGIC_STREAM,
        IGI_PRIM_INPUT_PROCESS_STREAM,
        IGI_PRIM_OUTPUT_PROCESS_STREAM
        } PalettePrimitive;
```

5. Primitive Object Type Index

```
typedef enum {
        IGI_NONE, IGI_ELLIPSE, IGI_BOX,
        IGI_LINE, IGI_TRIANGLE, IGI_STREAM,
        IGI_TAGNAME, IGI_TEXT, IGI_BOUNDBOX
        } PrimitiveType;
```

6. Primitive Object Pointer Index

```
typedef union PrimitivePointer
{
        CIgiEllipse    *pIGIELLIPSE;          // pointer to ellipse object
        CIgiBox               *pIGIBOX;       // pointer to box object
        CIgiTriangle   *pIGITRIANGLE;         // pointer to triangle object
        CIgiLine              *pIGILINE;      // pointer to line object
        CIgiText              *pIGITEXT;      // pointer to text object
        CIgiTagName    *pIGITAGNAME;          // pointer to tagname object
        CIgiStream     *pIGISTREAM;           // pointer to stream object
} ObjectPointer;
```

2.4.2 Specify Operations

2.4.2.1 Operations for class CIgieditorDoc

1. BOOL CIgieditorDoc::OnNewDocument()

Description:
This function creates a new igi document with MFC function OnNewDocument(). It calls InitDocument() to initialize the new document.

2. BOOL CIgieditorDoc::OnOpenDocument(LPCTSTR lpszPathName)

Description:

This function opens an existed IGI document with MFC function OnOpenDocument. It calls InitDocument() to initialize the opened document. The existed IGI document has the file name "lpszPathName".

3. void CIgieditorDoc::Serialize(CArchive& ar)

Description:
This function reads or writes the primitive object properties from/to IGI document file.

4. void CIgieditorDoc::OnSelectAll()

Description:
This command sets all primitive objects to be selected in the edit area.

5. void CIgieditorDoc:: OnUnSelectAll()

Description:
This command sets all primitive objects to be unselected in the edit area.

6. void CIgieditorDoc:: OnDeleteAll()

Description:
This command deletes all primitive objects in the edit area.

7. void CIgieditorDoc:: OnDeleteSome()

Description:
This command deletes all selected primitive objects in the edit area.

8. void CIgieditorDoc:: OnEditCopy()

Description:
This command copies all selected primitive objects in the edit area.

9. void CIgieditorDoc:: OnEditUndo ()

Description:
This command undoes the last edit action.

10. void CIgieditorDoc:: OnPushOne ()

Description:
This command pushes a selected primitive object one level down. It swaps the selected object node position with its previous node in the primitive list.

11. void CIgieditorDoc:: OnPushBottom ()

Description:
This command pushes a selected primitve object to bottom level. It moves the selected object position to the head of the primitive list.

12. void CIgieditorDoc:: OnPopOneUp ()

Description:

This command pops a selected primitive object one level up. It swaps the selected object position with its next node in the primitive list.

13. void CIgieditorDoc:: OnPopToTop ()

Description:
This command pops a selected primitive object to top level. It moves the selected object position to the tail of the primitive list.

14. void CIgieditorDoc:: OnUpdateSelectAll(CCmdUI* pCmdUI)

Description:
This command updates the "Select All" button on the main toolbar. The button is disabled if there is no primitive object in the edit area.

15. void CIgieditorDoc:: OnUpdateUnSelectAll(CCmdUI* pCmdUI)

Description:
This command updates the "Unselect All" button on the main toolbar. The button is disabled if there is no object selected in the edit area.

16. void CIgieditorDoc:: OnUpdateDeleteAll(CCmdUI* pCmdUI)

Description:
This command updates the "Cut All" button on the main toolbar. The button is disabled if there is no primitive object in the edit area.

17. void CIgieditorDoc:: OnUpdateDeleteSome (CCmdUI* pCmdUI)

Description:
This command updates the "Cut" button on the main toolbar. The button is checked if there is no primitive object selected in the edit area.

18. void CIgieditorDoc:: OnUpdateEditCopy (CCmdUI* pCmdUI)

Description:
This command updates the "Copy" button on the main toolbar. The button is checked if there is no primitive object selected in the edit area.

19. void CIgieditorDoc:: OnUpdateEditUndo (CCmdUI* pCmdUI)

Description:
This command updates the "Undo" button on the main toolbar. The button is checked if there is no change for the IGI document.

20. void CIgieditorDoc:: OnUpdatePushOne (CCmdUI* pCmdUI)

Description:
This command updates the "Push Down" button on the main toolbar. The button is checked if the number of selected primitive object is not equal to 1.

21. void CIgieditorDoc:: OnUpdatePushBottom (CCmdUI* pCmdUI)

Description:

This command updates the "Push To Bottom" button on the main toolbar. The button is checked if the number of selected primitive object is not equal to 1.

22. void CIgieditorDoc:: OnUpdatePopOneUp (CCmdUI* pCmdUI)

Description:
This command updates the "Pop Up" button on the main toolbar. The button is checked if the number of selected primitive object is not equal to 1.

23. void CIgieditorDoc:: OnUpdatePopToTop (CCmdUI* pCmdUI)

Description:
This command updates the "Pop To Top" button on the main toolbar. The button is checked if the number of selected primitive object is not equal to 1.

24. void CIgieditorDoc:: OnColorBlack()

Description:
This command sets black color as default color for adding new primitive object. It changes the selected primitive object's color to black and checks the black color button.

25. void CIgieditorDoc:: OnColorYellow ()

Description:
This command sets yellow color as default color for adding new primitive object. It changes the selected primitive object's color to yellow and checks the yellow color button.

26. void CIgieditorDoc:: OnColorRed ()

Description:
This command sets red color as default color for adding new primitive object. It changes the selected primitive object's color to red and checks the red color button.

27. void CIgieditorDoc:: OnColorBlue ()

Description:
This command sets blue color as default color for adding new primitive object. It changes the selected primitive object's color to blue and checks the blue color button.

28. void CIgieditorDoc:: OnColorGreen ()

Description:
This command sets green color as default color for adding new primitive object. It changes the selected primitive object's color to green and checks the green color button.

29. void CIgieditorDoc:: OnColorCyan ()

Description:
This command sets cyan color as default color for adding new primitive object. It changes the selected primitive object's color to cyan and checks the cyan color button.

30. void CIgieditorDoc:: OnColorMagenta ()

Description:

This command sets magenta color as default color for adding new primitive object. It changes the selected primitive object's color to magenta and checks the magenta color button.

31. void CIgieditorDoc:: OnColorWhite ()

Description:
This command sets white color as default color for adding new primitive object. It changes the selected primitive object's color to white and checks the white color button.

32. void CIgieditorDoc:: OnColorDarkblue ()

Description:
This command sets darkblue color as default color for adding new primitive object. It changes the selected primitive object's color to darkblue and checks the darkblue color button.

33. void CIgieditorDoc:: OnColorDarkgreen ()

Description:
This command sets darkgreen color as default color for adding new primitive object. It changes the selected primitive object's color to darkgreen and checks the darkgreen color button.

34. void CIgieditorDoc:: OnColorDarkcyan ()

Description:
This command sets darkcyan color as default color for adding new primitive object. It changes the selected primitive object's color to darkcyan and checks the darkcyan color button.

35. void CIgieditorDoc:: OnColorDarkred ()

Description:
This command sets darkred color as default color for adding new primitive object. It changes the selected primitive object's color to darkred and checks the darkred color button.

36. void CIgieditorDoc:: OnColorDarkmagenta ()

Description:
This command sets darkmagenta color as default color for adding new primitive object. It changes the selected primitive object's color to darkmagenta and checks the darkmagenta color button.

37. void CIgieditorDoc:: OnColorDarkyellow ()

Description:
This command sets darkyellow color as default color for adding new primitive object. It changes the selected primitive object's color to darkyellow and checks the darkyellow color button.

38. void CIgieditorDoc:: OnColorDarkgray ()

Description:
This command sets darkgray color as default color for adding new primitive object. It changes the selected primitive object's color to darkgray and checks the darkgray color button.

39. void CIgieditorDoc:: OnColorLightgray ()

Description:
This command sets lightgray color as default color for adding new primitive object. It changes the selected primitive object's color to lightgray and checks the lightgray color button.

40. void CIgieditorDoc:: OnUpdateColorBlack(CCmdUI* pCmdUI)

Description:
This command updates the black color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

41. void CIgieditorDoc:: OnUpdateColorYellow (CCmdUI* pCmdUI)

Description:
This command updates the yellow color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

42. void CIgieditorDoc:: OnUpdateColorRed (CCmdUI* pCmdUI)

Description:
This command updates the red color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

43. void CIgieditorDoc:: OnUpdateColorBlue (CCmdUI* pCmdUI)

Description:
This command updates the blue color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

44. void CIgieditorDoc:: OnUpdateColorGreen (CCmdUI* pCmdUI)

Description:
This command updates the green color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

45. void CIgieditorDoc:: OnUpdateColorCyan (CCmdUI* pCmdUI)

Description:
This command updates the cyan color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

46. void CIgieditorDoc:: OnUpdateColorMagenta (CCmdUI* pCmdUI)

Description:
This command updates the magenta color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

47. void CIgieditorDoc:: OnUpdateColorWhite (CCmdUI* pCmdUI)

Description:
This command updates the white color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

48. void CIgieditorDoc:: OnUpdateColorDarkblue (CCmdUI* pCmdUI)

Description:
This command updates the darkblue color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

49. void CIgieditorDoc:: OnUpdateColorDarkgreen (CCmdUI* pCmdUI)

Description:
This command updates the darkgreen color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

50. void CIgieditorDoc:: OnUpdateColorDarkcyan (CCmdUI* pCmdUI)

Description:
This command updates the darkcyan color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

51. void CIgieditorDoc:: OnUpdateColorDarkred (CCmdUI* pCmdUI)

Description:
This command updates the darkred color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

52. void CIgieditorDoc:: OnUpdateColorDarkmagenta (CCmdUI* pCmdUI)

Description:
This command updates the darkmagenta color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

53. void CIgieditorDoc:: OnUpdateColorDarkyellow (CCmdUI* pCmdUI)

Description:
This command updates the darkyellow color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

54. void CIgieditorDoc:: OnUpdateColorDarkgray (CCmdUI* pCmdUI)

Description:
This command updates the darkgray color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

55. void CIgieditorDoc:: OnUpdateColorLightgray (CCmdUI* pCmdUI)

Description:
This command updates the lightgray color button on the color bar. The button is checked if the color button is clicked and it is unchecked when other color button is clicked.

56. void CIgieditorDoc:: OnPaletteRectangle ()

Description:
This command sets Rectangle as default primitive for adding new primitive object. It checks the Rectangle button on the primitive bar.

57. void CIgieditorDoc:: OnPaletteFillRectangle ()

Description:
This command sets Fill Rectangle as default primitive for adding new primitive object. It checks the Fill Rectangle button on the primitive bar.

58. void CIgieditorDoc:: OnPaletteTriangle ()

Description:
This command sets Triangle as default primitive for adding new primitive object. It checks the Triangle button on the primitive bar.

59. void CIgieditorDoc:: OnPaletteFillTriangle ()

Description:
This command sets Fill Triangle as default primitive for adding new primitive object. It checks the Fill Triangle button on the primitive bar.

60. void CIgieditorDoc:: OnPaletteEllipse ()

Description:
This command sets Ellipse as default primitive for adding new primitive object. It checks the Ellipse button on the primitive bar.

61. void CIgieditorDoc:: OnPaletteFillEllipse ()

Description:
This command sets Fill Ellipse as default primitive for adding new primitive object. It checks the Fill Ellipse button on the primitive bar.

62. void CIgieditorDoc:: OnPaletteText ()

Description:
This command sets Text as default primitive for adding new primitive object. It checks the Text button on the primitive bar. It pops up the text input dialog to allow user type added text.

63. void CIgieditorDoc:: OnPaletteTag ()

Description:
This command sets Tag as default primitive for adding new primitive object. It checks the Tag button on the primitive bar.

64. void CIgieditorDoc:: OnPaletteProcessIn ()

Description:
This command sets Process Input as default primitive for adding new primitive object. It checks the Process Input button on the primitive bar.

65. void CIgieditorDoc:: OnPaletteProcessOut ()

Description:

This command sets Process Output as default primitive for adding new primitive object. It checks the Process Output button on the primitive bar.

66. void CIgieditorDoc:: OnPaletteLogicIn ()

Description:
This command sets Logic Input as default primitive for adding new primitive object. It checks the Logic Input button on the primitive bar.

67. void CIgieditorDoc:: OnPaletteLogicOut ()

Description:
This command sets Logic Output as default primitive for adding new primitive object. It checks the Logic Output button on the primitive bar.

68. void CIgieditorDoc:: OnPaletteAnalogIn ()

Description:
This command sets Analog Input as default primitive for adding new primitive object. It checks the Analog Input button on the primitive bar.

69. void CIgieditorDoc:: OnPaletteAnalogOut ()

Description:
This command sets Analog Output as default primitive for adding new primitive object. It checks the Analog Output button on the primitive bar.

70. void CIgieditorDoc:: OnPaletteLine ()

Description:
This command sets Line as default primitive for adding new primitive object. It checks the Line button on the primitive bar.

71. void CIgieditorDoc:: OnPaletteIndicator ()

Description:
This command finishes adding primitive action and sets to move mode. It unchecks other primitive button on the primitive bar.

72. void CIgieditorDoc:: OnUpdatePaletteRectangle(CCmdUI* pCmdUI)

Description:
This command updates the Rectangle primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

73. void CIgieditorDoc:: OnUpdatePaletteFillRectangle (CCmdUI* pCmdUI)

Description:
This command updates the FillRectangle primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

74. void CIgieditorDoc:: OnUpdatePaletteTriangle (CCmdUI* pCmdUI)

Description:

This command updates the Triangle primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

75. void CIgieditorDoc:: OnUpdatePaletteFillTriangle (CCmdUI* pCmdUI)

Description:
This command updates the FillTriangle primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

76. void CIgieditorDoc:: OnUpdatePaletteEllipse (CCmdUI* pCmdUI)

Description:
This command updates the Ellipse primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

77. void CIgieditorDoc:: OnUpdatePaletteFillEllipse (CCmdUI* pCmdUI)

Description:
This command updates the FillEllipse primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

78. void CIgieditorDoc:: OnUpdatePaletteText (CCmdUI* pCmdUI)

Description:
This command updates the Text primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

79. void CIgieditorDoc:: OnUpdatePaletteTag (CCmdUI* pCmdUI)

Description:
This command updates the Tag primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

80. void CIgieditorDoc:: OnUpdatePaletteProcessIn (CCmdUI* pCmdUI)

Description:
This command updates the Process Input primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

81. void CIgieditorDoc:: OnUpdatePaletteProcessOut (CCmdUI* pCmdUI)

Description:
This command updates the Process Output primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

82. void CIgieditorDoc:: OnUpdatePaletteLogicIn (CCmdUI* pCmdUI)

Description:
This command updates the Logic Input primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

83. void CIgieditorDoc:: OnUpdatePaletteLogicOut (CCmdUI* pCmdUI)

Description:
This command updates the Logic Output primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

84. void CIgieditorDoc:: OnUpdatePaletteAnalogIn (CCmdUI* pCmdUI)

Description:
This command updates the Analog Input primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

85. void CIgieditorDoc:: OnUpdatePaletteAnalogOut (CCmdUI* pCmdUI)

Description:
This command updates the Analog Output primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

86. void CIgieditorDoc:: OnUpdatePaletteLine (CCmdUI* pCmdUI)

Description:
This command updates the Line primitive button on the primitive bar. The button is checked if the primitive button is clicked and it is unchecked when other primitive button is clicked.

87. void CIgieditorDoc:: OnUpdatePaletteIndicator (CCmdUI* pCmdUI)

Description:
This command updates the Indicator button on the primitive bar. Clicking this button will uncheck other checked primitive button and change edit mode from ADD MODE to MOVE MODE.

88. void CIgieditorDoc::InitDocument()

Description:
This function initializes the color bar and primitive bar. It sets black color as default color and MOVE MODE as default editor mode. It is called by commands OnNewDocument and OnOpenDocument.

89. void CIgieditorDoc:: SetColorBar(int index)

Description:
This function sets the color bar according to the mouse click on the color bar. It sets the color button that is clicked to TRUE and all other color buttons to FALSE. It is called when the user clicks on the color bar.

90. void CIgieditorDoc:: SetPaletteBar(int index)

Description:
This function sets the primitive bar according to the mouse click on the primitive bar. It sets the primitive button that is clicked to TRUE and all other primitive buttons to FALSE. It is called when the user clicks on the primitive bar.

91. void CIgieditorDoc:: UnSetPaletteBar()

Description:
This function sets all primitive buttons to FALSE. It is called when the user clicks right mouse button on the edit area to finish adding primitive object action.

92. COLORREF CIgieditorDoc:: GetCurrentColor ()

Description:
This function gets the current color that user clicks on the color bar.

93. EditModeIndex CIgieditorDoc:: GetEditMode ()

Description:
This function gets the current edit mode. It is initialized to move mode and changed to add primitive object mode after user clicks a primitive on the primitive bar.

94. PalettePrimitive CIgieditorDoc:: GetCurrentPrimitive ()

Description:
This function gets the type of primitive object that will be added to the edit area. It depends on the primitive button is clicked on the primitive bar.

95. BOOL CIgieditorDoc:: GetFillMode ()

Description:
This function gets the fill mode for adding a new primitive object. It is TRUE for FillRectangle, FillEllipse and FillTriangle and it is FALSE for all other kind of primitive.

96. int CIgieditorDoc:: GetSelectOrder ()

Description:
This function gets the select order for adding triangle object. It returns 1 when user selects first point and 2 for second point and 3 for the third point of the triangle.

97. void CIgieditorDoc:: SetSelectOrder(int selectOrder)

Description:
This function sets the select order for adding triangle object. It sets the order to 1 when user selects first point and 2 for second point and 3 for the third point of the triangle.

98. CIGIprimitive* CIgieditorDoc::NewIgiPrimitive(PalettePrimitive item, int x[6])

Description:
This function creates a new primitive according to the selected primitive and color button. The new primitive object is added to the tail of the primitive object list. IgieditorView class calls it after user selected the desired position on the edit area.

99. void CIgieditorDoc::DeleteContents()

Description:
This function deletes all primitive object nodes from the list. It is called when user does "Cut All" action. It sets the IGI document to empty.

100. CIGIprimitive* CIgieditorDoc::SelectIGIprimitive(CPoint point)

Description:
This function returns the pointer of a node in the primitive list that contains the selected primitive when user clicks the left mouse button over a primitive. Otherwise, it returns NULL. If the mouse is clicked over a primitive, the function sets the primitive selected and updates the number of selected primitives on the list.

101. void CIgieditorDoc::DrawIGIprimitiveList(CDC *pDC)

Description:
This function will draw each primitive object on the edit area. It will be called by command "OnDraw" in class IgieditorView.

102. void CIgieditorDoc::SetIGIObjectSelectList(BOOL select)

Description:
This function sets all primitive objects to select or unselect. If select is TRUE, it assigns select count as the node number of the list. Otherwise, it sets the select count to 0.

103. void CIgieditorDoc::MoveIGIPrimitiveList(int dx, int dy)

Description:
This function adds the moving distance (dx, dy) to each primitive object. It copies the current primitive list to previous list for the purpose of undo before it updates the each primitive object's position. After update, it sets document-modified flag to save document when the program exist or open another file.

104. void CIgieditorDoc::SetEditMode(EditModeIndex editMode)

Description:
This function sets the edit mode. It is called when user has finished adding new primitive object and changed the edit mode from adding mode to moving mode.

105. void CIgieditorDoc::UpdateColor()

Description:
This function updates each selected primitive object's color when user clicks on the color button.

106. void CIgieditorDoc::CreatePreviousList()

Description:
This function copies the primitive object list to another one as a backup list. The purpose of creating a backup list is for "Undo". When user wants to undo the last action, it only needs to copy the backup list to the primitive object list. This function is called when user modifies the primitive object list each time.

107. void CIgieditorDoc::SwapList()

Description:
This function switches the previous object list and current primitive object list. It is called when user runs action "Undo".

108. void CIgieditorDoc::ModifyIGIPrimitive(CIGIprimitive* p, int dx, int dy, int index)

Description:
This function updates the primitive object that is modified by user.

109. CIGIprimitive* CIgieditorDoc::CopyIGIprimitive(CIGIprimitive *pIGIprimitive, int dx)

Description:
This function makes a copy for an IGIprimitive that is a node of the primitive object list. It is called in two cases. One is called by creating previous list. In this case, the parameter dx = 0 and the ObjectSelect is keeped same as original IGIprimitive object. Another case it is called is that user runs the action "Copy". At this time, dx is not equal 0 and that add a small distance to new-copied primitive object. For new primitive object, the ObjectSelect is set to FALSE.

110. int CIgieditorDoc::GetCountSelect()

Description:
This function returns the total selected primitive object number. The purpose of this number is to check the button of the main tool bar. For example, if there is no object is selected, the "Copy" button is checked and that means the action is disable at this time.

111. int CIgieditorDoc::GetTagNameFlag()

Description:
This function returns the number of tag name in the IGI document. It only has one tag name in each IGI document. User is not allowed to add second tag name object in the primitive list. The flag 0 means there is no tag name in the list yet.

112. void CIgieditorDoc::SetTagNameFlag(int tagNameFlag)

Description:
This function sets the tag name flag to 1 when user add tag name object to the primitive object list. The flag is set to 0 when the tag name object has been deleted.

113. CString CIgieditorDoc::GetTextInput()

Description:
This function returns the input text string when user adds a new text.

114. int CIgieditorDoc::GetNewStreamNumber(int iType, int iDir)

Description:
This function gets a new stream number according to its type and direction. It is called when user adds or copies a stream.

115. void CIgieditorDoc::IncreaseOrientation(CIGIprimitive *p)

Description:

This function changes the stream orientation value when user rotates a stream.

116. CIGIprimitive* CIgieditorDoc::GetSelectIGIstream()

Description:
This function gets the pointer of primitive stream object in the list that has been selected. It is called when user clicks right mouse button to rotate a stream orientation.

117. void CIgieditorDoc::InvertIgiPrimitive(CDC *pDC, CIGIprimitive *p, int dx, int dy)

Description:
This function inverts the primitive object display on the edit area during the object is moving.

118. void CIgieditorDoc::OnKeyDelete()

Description:
This function calls "OnDeleteSome" command to delete the selected primitive objects. This function is called when user presses the "Delete" key to cut the selected primitive objects.

2.4.2.2 Operations for class CIgieditorView

1. void CIgieditorView::OnDraw(CDC* pDC)

Description:
This function delegates the drawing of individual primitive to CIGIPrimitve::DrawIGIPrimitive(). It is called when the view window updated.

2. int CIgieditorView::PreTranslateMessage(MSG* pMsg)

Description:
This function is CIgieditorView message handlers.
Nonzero if the message was fully processed in PreTranslateMessage and should not be processed further. Zero if the message should be processed in the normal way.

3. void CIgieditorView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

Description:
This command moves the selected primitive objects with the direction key: VK_UP, VK_DOWN, VK_LEFT, and VK_RIGHT. This command also deletes the selected primitive objects when user press key "Delete"

4. void CIgieditorView::OnLButtonDown(UINT nFlags, CPoint point)

Description:
This function returns the pointer of a primitive object when the mouse is clicked on. If user modifies the object, the function also returns the mark index to show which point will be modified.

5. void CIgieditorView::OnLButtonUp(UINT nFlags, CPoint point)

Description:
This function calls "Modify" or "Move" functions in document class to update each primitive object if user modifies or moves the objects. If mouse is not clicked on a object, the function sets all object to unselected.

6. void CIgieditorView::OnRButtonUp (UINT nFlags, CPoint point)

Description:
This function calls "IncreaseOrientation" functions in document class to update the orientation of a stream object when user rotates it.

7. void CIgieditorView::OnMouseMove(UINT nFlags, CPoint point)

Description:
This function updates the objects display during user modifies or moves the selected objects. If a stream object has been selected, it could be rotated and proper message is displayed on the status bar.

8. void CIgieditorView::ModifyPrimitive(CPoint point)

Description:
This function inverts a primitive object during the object modified.

9. BOOL CIgieditorView::MovePrimitive(CPoint point)

Description:
This function inverts the primitive objects during the objects moved.

10. void CIgieditorView::InvertBox(CDC *pDC, CPoint ptFrom, CPoint ptTo)

Description:
This function inverts a rectangle object during adding a Rectangle/FillRectangle object.

11. void CIgieditorView::InvertEllipse(CDC *pDC, CPoint ptFrom, CPoint ptTo)

Description:
This function inverts a ellipse object during adding a Ellipse/FillEllipse object.

12. void CIgieditorView::InvertLine(CDC *pDC, CPoint ptFrom, CPoint ptTo)

Description:
This function inverts a line object during adding a line or triangle object.

2.4.2.3 Operations for class CIGIprimitive

1. BOOL CIGIprimitive::DrawIGIprimitive(CDC *pDC)

Description:
This function calls each primitive to draw its object on the edit area.

2. void CIGIprimitive::SetPrimitiveType(PalettePrimitive item)

Description:

This function sets the current primitive object type when user adds an object to an IGI document. It is called when a new object to be created at adding new object or copying an object.

3. void CIGIprimitive::SetParameter(int x[])

Description:
This function saves a primitive object parameter to IGIprimitive object. It is called when user adds a new primitive object or copies an object.

4. void CIGIprimitive::NewPrimitive()

Description:
This function calls each primitive to create an object. It will set object color and fill mode. For new object, it is always unselected.

5. void CIGIprimitive::SetPenColor(COLORREF currentColor)

Description:
This function saves display color for primitive object in IGIprimitive object.

6. void CIGIprimitive::SetFillMode(BOOL bFillMode)

Description:
This function saves fill mode for primitive object to IGIprimitive object.

7. BOOL CIGIprimitive::SelectIGIprimitive(CPoint point)

Description:
This function calls each primitive to check whether the "point" is over the primitive object.

8. BOOL CIGIprimitive::GetSelectMark()

Description:
This function calls each primitive to return whether the mouse is clicked on the select mark. When the mouse is clicked on the select mark, which means user can modify the primitive object. Otherwise, the object will only be moved. Only primitive object "Line, Rectangle/FillRectangle, Ellipse/FillEllipse, and Triangle/FillTriangle" have select marks and can be modified.

9. int CIGIprimitive::GetMarkIndex()

Description:
This function calls each primitive to return the mark index on which the mouse is clicked. The primitive Line has two select marks. Triangle has three and Rectangle (Box) has four.

10. void CIGIprimitive::ModifyPrimitive(int dx, int dy, int index)

Description:
This function calls each primitive to update the object status because of the modification.

11. void CIGIprimitive::MoveIGIPrimitive(int dx, int dy)

Description:
This function calls each primitive to update the object position.

12. void CIGIprimitive::InvertIgiPrimitive(CDC* pDC, int dx, int dy)

Description:
This function calls each primitive to invert the primitive object during the object is moving.

13. void CIGIprimitive::SetIGIObjectSelect(BOOL select)

Description:
This function calls each primitive to set object select to TRUE or FALSE.

14. void CIGIprimitive::UpdateColor(COLORREF currentColor)

Description:
This function calls each primitive to update the object color.

15. BOOL CIGIprimitive::GetIGIObjectSelect()

Description:
This function calls each primitive to get status of the object selection.

16. void CIGIprimitive::GetParameter(int* x, int dx)

Description:
This function calls each primitive to get parameters of each object.

17. PalettePrimitive CIGIprimitive::GetPrimitiveItem()

Description:
This function converts primitive type to its proper item.

18. COLORREF CIGIprimitive::GetPenColor()

Description:
This function calls each primitive to get its object color.

19. BOOL CIGIprimitive::GetFillMode()

Description:
This function calls each primitive to get its object fill mode.

20. void CIGIprimitive::SetIGItextinput(CString cTextInput)

Description:
This function sets input text string for the TEXT primitive.

21. CString CIGIprimitive::GetIGItextinput()

Description:
This function gets input text string from the TEXT primitive object.

22. BOOL CIGIprimitive::IsLookForStream(int iType, int iDir)

Description:
This function checks the stream according to its type and direction.

23. int CIGIprimitive::GetStreamNumber(int iType, int iDir)

Description:

This function checks the stream according to its type and direction.

24. void CIGIprimitive::SetStreamNumber(int iNoStream)

Description:
This function sets the stream number to a stream object.

25. void CIGIprimitive::IncreaseOrientation()

Description:
This function sets the stream orientation for a stream object when a stream has been rotated.

26. int CIGIprimitive::GetStreamType()

Description:
This function gets a stream type from a stream object.

27. int CIGIprimitive::GetStreamDir()

Description:
This function gets a stream direction from a stream object.

28. int CIGIprimitive::GetPrimitiveType()

Description:
This function gets a primitive object type from IGIprimitive object.

29. CPoint CIGIprimitive::GetModifyPoint(int iMarkIndex, int iFromIndex)

Description:
This function calls each primitive to get the start point position after the object has been modified.

2.4.2.4 Operations for class CigiBox

1. CIgiBox::CIgiBox()

Description:
This constructor sets the object not selected when it is created.

2. CIgiBox::CIgiBox(int x1, int y1, int x2, int y2)

Description:
This constructor creates an object that position is decided by x1, y1, x2 and y2.

3. BOOL CIgiBox::SelectPrimitive(int x, int y)

Description:
This function returns TURE if the (x, y) is on the object. It also sets the mark index if the mouse position is on the select mark. If the mouse position is not on the object, it returns FALSE.

4. void CIgiBox::ModifyPrimitive(int dx, int dy, int index)

Description:
This function updates the position of the object according to the modification value (dx, dy) and which point the mouse has been clicked on (index).

5. void CIgiBox::DrawPrimitive(CDC *pDC)

Description:
This function draws the box according to its fill mode. If the object is selected, the function draws a small square as the select mark on its control point.

6. void CIgiBox::MovePrimitive(int dx, int dy)

Description:
This function updates the object position according to its moving distance (dx, dy).

7. void CIgiBox::InvertIgiPrimitive(CDC *pDC, int dx, int dy)

Description:
This function inverts the object during it is moving.

8. void CIgiBox::Serialize(CArchive &ar)

Description:
This function reads/writes the object from/to a file on the hard disk.

9. COLORREF CIgiBox::GetPenColor()

Description:
This function returns the color that the object is displayed.

10. BOOL CIgiBox::GetObjectSelect()

Description:
This function returns the status that the object is selected.

11. void CIgiBox::SetObjectSelect(BOOL select)

Description:
This function sets the status that the selection of the object.

12. void CIgiBox::SetPenColor(COLORREF currentColor)

Description:
This function sets the color that the object is displayed.

13. BOOL CIgiBox::GetFillMode()

Description:
This function gets the fill mode for the object.

14. void CIgiBox::SetFillMode(BOOL bFillMode)

Description:
This function sets the fill mode for the object.

15. CPoint CIgiBox::GetModifyPoint(int iMarkIndex)

Description:

This function returns the point that is not changed when the object is modified.

16. void CIgiBox::GetParameter(int *x, int dx)

Description:
This function returns the position of the object.

17. BOOL CIgiBox::GetSelectMark()

Description:
This function returns the selection status of the object. It returns TRUE if the mouse is clicked on the select mark.

18. int CIgiBox::GetMarkIndex()

Description:
This function returns the mark index when the mouse is clicked on the select mark.

2.4.2.5 Operations for class CIgiEllipse

1. CIgiEllipse::CIgiEllipse ()

Description:
This constructor sets the object not selected when it is created.

2. CIgiEllipse::CIgiEllipse(int x1, int y1, int x2, int y2)

Description:
This constructor creates an object that position is decided by x1, y1, x2 and y2.

3. BOOL CIgiEllipse::SelectPrimitive(int x, int y)

Description:
This function returns TURE if the (x, y) is on the object. It also sets the mark index if the mouse position is on the select mark. If the mouse position is not on the object, it returns FALSE.

4. void CIgiEllipse::ModifyPrimitive(int dx, int dy, int index)

Description:
This function updates the position of the object according to the modification value (dx, dy) and which point the mouse has been clicked on (index).

5. void CIgiEllipse::DrawPrimitive(CDC *pDC)

Description:
This function draws the box according to its fill mode. If the object is selected, the function draws a small square as the select mark on its control point.

6. void CIgiEllipse::MovePrimitive(int dx, int dy)

Description:
This function updates the object position according to its moving distance (dx, dy).

7. void CIgiEllipse::InvertIgiPrimitive(CDC *pDC, int dx, int dy)

Description:
This function inverts the object during it is moving.

8. void CIgiEllipse::Serialize(CArchive &ar)

Description:
This function reads/writes the object from/to a file on the hard disk.

9. COLORREF CIgiEllipse::GetPenColor()

Description:
This function returns the color that the object is displayed.

10. BOOL CIgiEllipse::GetObjectSelect()

Description:
This function returns the status that the object is selected.

11. void CIgiEllipse::SetObjectSelect(BOOL select)

Description:
This function sets the status that the selection of the object.

12. void CIgiEllipse::SetPenColor(COLORREF currentColor)

Description:
This function sets the color that the object is displayed.

13. BOOL CIgiEllipse::GetFillMode()

Description:
This function gets the fill mode for the object.

14. void CIgiEllipse::SetFillMode(BOOL bFillMode)

Description:
This function sets the fill mode for the object.

15. CPoint CIgiEllipse::GetModifyPoint(int iMarkIndex)

Description:
This function returns the point that is not changed when the object is modified.

16. void CIgiEllipse::GetParameter(int *x, int dx)

Description:
This function returns the position of the object.

17. BOOL CIgiEllipse::GetSelectMark()

Description:
This function returns the selection status of the object. It returns TRUE if the mouse is clicked on the select mark.

18. int CIgiEllipse::GetMarkIndex()

Description:

This function returns the mark index when the mouse is clicked on the select mark.

2.4.2.6 Operations for class CIgiLine

1. CIgiLine::CIgiLine()

Description:
This constructor sets the object not selected when it is created.

2. CIgiLine::CIgiLine(int x1, int y1, int x2, int y2)

Description:
This constructor creates an object that position is decided by x1, y1, x2 and y2.

3. BOOL CIgiLine::SelectPrimitive(int x, int y)

Description:
This function returns TURE if the (x, y) is on the object. It also sets the mark index if the mouse position is on the select mark. If the mouse position is not on the object, it returns FALSE.

4. void CIgiLine::ModifyPrimitive(int dx, int dy, int index)

Description:
This function updates the position of the object according to the modification value (dx, dy) and which point the mouse has been clicked on (index).

5. void CIgiLine::DrawPrimitive(CDC *pDC)

Description:
This function draws the object with the selected color. If the object is selected, the function draws a small square as the select mark.

6. void CIgiLine::MovePrimitive(int dx, int dy)

Description:
This function updates the object position according to its moving distance (dx, dy).

7. void CIgiLine::InvertIgiPrimitive(CDC *pDC, int dx, int dy)

Description:
This function inverts the object during it is moving.

8. void CIgiLine::Serialize(CArchive &ar)

Description:
This function reads/writes the object from/to hard disk.

9. COLORREF CIgiLine::GetPenColor()

Description:
This function returns the color that the object is displayed.

10. BOOL CIgiLine::GetObjectSelect()

Description:
This function returns the status that the object is selected.

11. void CIgiLine::SetObjectSelect(BOOL select)

Description:
This function sets the status that the selection of the object.

12. void CIgiLine::SetPenColor(COLORREF currentColor)

Description:
This function sets the color that the object is displayed.

13. CPoint CIgiLine::GetModifyPoint(int iMarkIndex)

Description:
This function returns the point that is not changed when the object is modified.

14. void CIgiLine::GetParameter(int *x, int dx)

Description:
This function returns the position of the object.

15. BOOL CIgiLine::GetSelectMark()

Description:
This function returns the selection status of the object. It returns TRUE if the mouse is clicked on the select mark.

16. int CIgiLine::GetMarkIndex()

Description:
This function returns the mark index when the mouse is clicked on the select mark.

2.4.2.7 Operations for class CIgiTriangle

1. CIgiTriangle::CIgiTriangle()

Description:
This constructor sets the object not selected when it is created.

2. CIgiTriangle::CIgiTriangle(int x1, int y1, int x2, int y2, int x3, int y3)

Description:
This constructor creates an object that position is decided by x1, y1, x2, y2, x3 and y3.

3. BOOL CIgiTriangle::SelectPrimitive(int x, int y)

Description:
This function returns TURE if the (x, y) is on the object. It also sets the mark index if the mouse position is on the select mark. If the mouse position is not on the object, it returns FALSE.

4. void CIgiTriangle::ModifyPrimitive(int dx, int dy, int index)

Description:
This function updates the position of the object according to the modification value (dx, dy) and which point the mouse has been clicked on (index).

5. void CIgiTriangle::DrawPrimitive(CDC *pDC)

Description:
This function draws the object according to its fill mode. If the object is selected, the function draws a small square as the select mark.

6. void CIgiTriangle::MovePrimitive(int dx, int dy)

Description:
This function updates the object position according to its moving distance (dx, dy).

7. void CIgiTriangle::InvertIgiPrimitive(CDC *pDC, int dx, int dy)

Description:
This function inverts the object during it is moving.

8. void CIgiTriangle::Serialize(CArchive &ar)

Description:
This function reads/writes the object from/to hard disk.

9. COLORREF CIgiTriangle::GetPenColor()

Description:
This function returns the color that the object is displayed.

10. BOOL CIgiTriangle::GetObjectSelect()

Description:
This function returns the status that the object is selected.

11. void CIgiTriangle::SetObjectSelect(BOOL select)

Description:
This function sets the status that the selection of the object.

12. void CIgiTriangle::SetPenColor(COLORREF currentColor)

Description:
This function sets the color that the object is displayed.

13. BOOL CIgiTriangle::GetFillMode()

Description:
This function gets the fill mode for the object.

14. void CIgiTriangle::SetFillMode(BOOL bFillMode)

Description:
This function sets the fill mode for the object.

15. CPoint CIgiTriangle::GetModifyPoint(int iMarkIndex, int iFromIndex)

Description:

This function returns the point that is not changed when the object is modified.

16. void CIgiTriangle::GetParameter(int *x, int dx)

Description:
This function returns the position of the object.

17. BOOL CIgiTriangle::GetSelectMark()

Description:
This function returns the selection status of the object. It returns TRUE if the mouse is clicked on the select mark.

18. int CIgiTriangle::GetMarkIndex()

Description:
This function returns the mark index when the mouse is clicked on the select mark.

2.4.2.8 Operations for class CigiText

1. CIgiText::CIgiText()

Description:
This constructor sets the object not selected when it is created.

2. CIgiText::CIgiText(int x, int y, int len, int high)

Description:
This constructor creates an object that position is decided by x, y. The text size is decided by len and high.

3. BOOL CIgiText::SelectPrimitive(int x, int y)

Description:
This function returns TURE if the (x, y) is on the object. If the mouse position is not on the object, it returns FALSE.

4. void CIgiText::DrawPrimitive(CDC *pDC)

Description:
This function draws the object with the selected color. If the object is selected, the function draws a box around text as the select mark.

5. void CIgiText::MovePrimitive(int dx, int dy)

Description:
This function draws the object with the selected color. If the object is selected, the function draws a box around text as the select mark.

6. void CIgiText::InvertIgiPrimitive(CDC *pDC, int dx, int dy)

Description:
This function inverts the object during it is moving.

7. void CIgiText::Serialize(CArchive &ar)

Description:
This function reads/writes the object from/to hard disk.

8. COLORREF CIgiText::GetPenColor()

Description:
This function returns the color that the object is displayed.

9. BOOL CIgiText::GetObjectSelect()

Description:
This function returns the status that the object is selected.

10. void CIgiText::SetObjectSelect(BOOL select)

Description:
This function sets the status that the selection of the object.

11. void CIgiText::SetPenColor(COLORREF currentColor)

Description:
This function sets the color that the object is displayed.

12. void CIgiText::GetParameter(int *x, int dx)

Description:
This function returns the position and the size of the object.

13. void CIgiText::SetTextInput(CString cTextInput)

Description:
This function set the text string for the object.

14. CString CIgiText::GetTextInput()

Description:
This function returns the text string of the object.

2.4.2.9 Operations for class CigiTagName

1. CIgiTagName::CIgiTagName()

Description:
This constructor sets the object not selected when it is created.

2. CIgiTagName::CIgiTagName(int x, int y, int len, int high)

Description:
This constructor creates an object that position is decided by x, y and its size decided by len and high.

3. BOOL CIgiTagName::SelectPrimitive(int x, int y)

Description:

This function returns TURE if the (x, y) is on the object. If the mouse position is not on the object, it returns FALSE.

4. void CIgiTagName::DrawPrimitive(CDC *pDC)

Description:
This function draws the object with the selected color. If the object is selected, the function draws a box around text as the select mark.

5. void CIgiTagName::MovePrimitive(int dx, int dy)

Description:
This function updates the object position according to its moving distance (dx, dy).

6. void CIgiTagName::InvertIgiPrimitive(CDC *pDC, int dx, int dy)

Description:
This function inverts the object during it is moving.

7. void CIgiTagName::Serialize(CArchive &ar)

Description:
This function reads/writes the object from/to hard disk.

8. COLORREF CIgiTagName::GetPenColor()

Description:
This function returns the color that the object is displayed.

9. BOOL CIgiTagName::GetObjectSelect()

Description:
This function returns the status that the object is selected.

10. void CIgiTagName::SetObjectSelect(BOOL select)

Description:
This function sets the status that the selection of the object.

11. void CIgiTagName::SetPenColor(COLORREF currentColor)

Description:
This function sets the color that the object is displayed.

12. void CIgiTagName::GetParameter(int *x, int dx)

Description:
This function returns the position and the size of the object.

2.4.2.10 Operations for class CigiStream

1. CIgiStream::CIgiStream()

Description:
This constructor sets the object not selected when it is created.

2. CIgiStream::CIgiStream(int x, int y, int iType, int iDir, int iOrientatio, int iNoStream)

Description:
This constructor creates an object that position is decided by x, y with giving iType, iDir, iOrientatio and iNoStream.

3. BOOL CIgiStream::SelectPrimitive(int x, int y)

Description:
This function returns TURE if the (x, y) is on the object. It also sets the mark index if the mouse position is on the select mark. If the mouse position is not on the object, it returns FALSE.

4. void CIgiStream::DrawPrimitive(CDC *pDC)

Description:
This function draws the object with different format according to its type. If the object is selected, the function draws a small square as the select mark.

5. void CIgiStream::MovePrimitive(int dx, int dy)

Description:
This function updates the object position according to its moving distance (dx, dy).

6. void CIgiStream::InvertIgiPrimitive(CDC *pDC, int dx, int dy)

Description:
This function inverts the object during it is moving.

7. void CIgiStream::Serialize(CArchive &ar)

Description:
This function reads/writes the object from/to hard disk.

8. void CIgiStream::IncreaseOrientation()

Description:
This function increases the orientation of an object. The relationship between a stream direction and its orientation value shows in the following table:

Table 2-18 Relationship of stream direction and its orientation

Stream direction	**Orientation value**
Point to east (→)	0
Point to north (↑)	1
Point to west (←)	2
Point to south (↓)	3

9. BOOL CIgiStream::GetObjectSelect()

Description:
This function returns the status that the object is selected.

10. void CIgiStream::SetObjectSelect(BOOL select)

Description:
This function sets the status that the selection of the object.

11. int CIgiStream::GetStreamNumber()

Description:
This function returns the stream number of the object.

12. void CIgiStream::SetStreamNumber(int iNoStream)

Description:
This function sets the stream number to the object.

13. int CIgiStream::GetStreamType()

Description:
This function returns the stream type of the object.

14. int CIgiStream::GetStreamDir()

Description:
This function returns the stream direction of the object.

15. int CIgiStream::GetStreamOrientation()

Description:
This function returns the stream orientation of the object.

16. void CIgiStream::GetParameter(int *x, int dx)

Description:
This function returns the position and other parameters of the object.

3. REFERENCES

1. Design Patterns, Elements of Reusable Object-Oriented Software by Erich Gamma, etc.

2. Applying UML and Patterns, an Introduction to object-oriented analysis and design by Craig Larman

3. A Practical Introduction to Software Design with C++ by Steven P. Reiss

4. Design Patterns Explained: A New Perspective on Object-Oriented Design by Alan Shalloway, etc.

5. COMP354 Course Notes by Olga Ormandjieva

    - IEEE template for SRS Document

    - Template for SDD Document

4. APPENDICES

4.1 Sample of Icon Graphic Image (IGI)

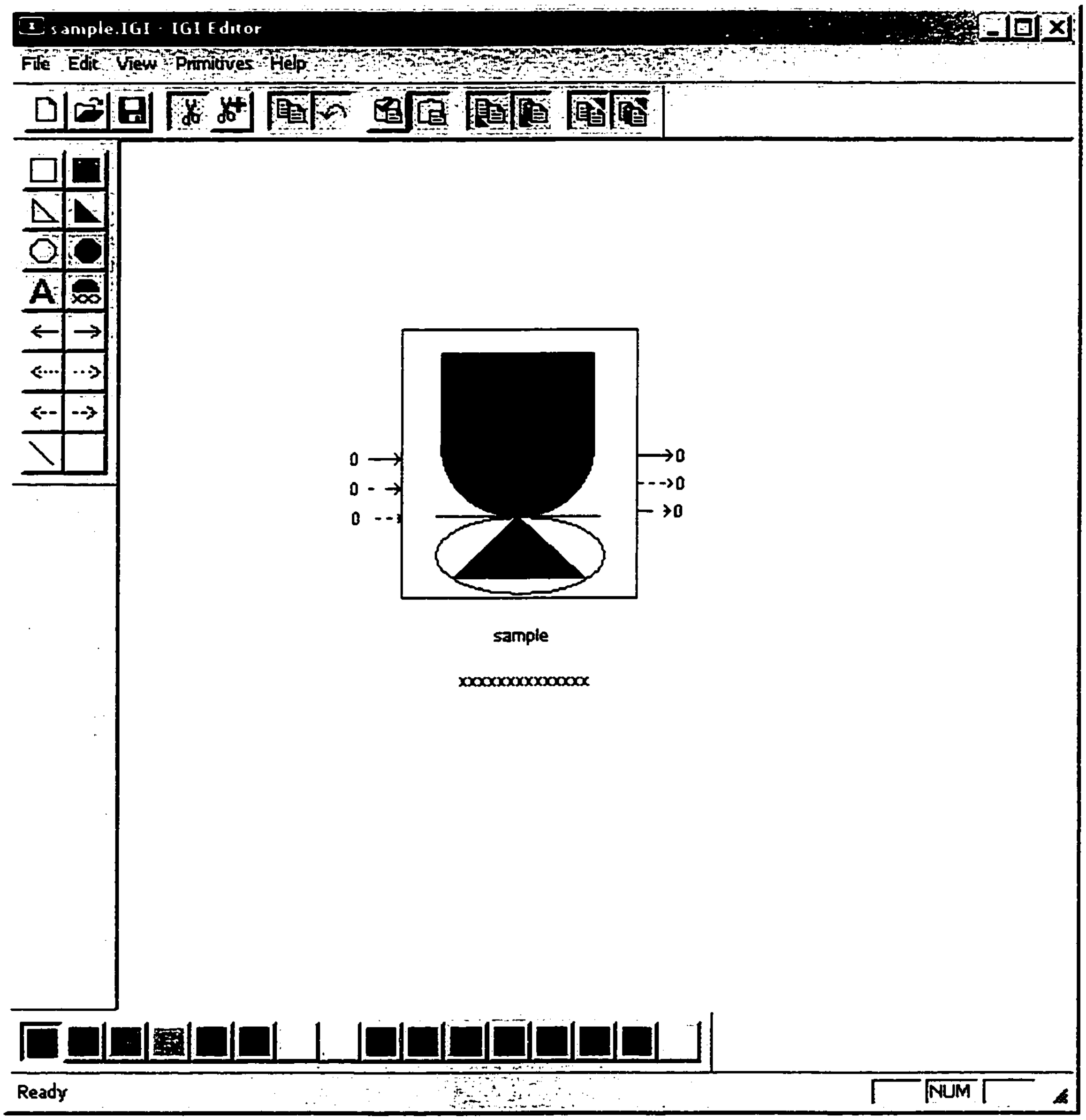


Figure 4-1 A sample of Icon Graphic Image (IGI)

4.2 Sample of Source Code

4.2.1 IGIprimitive.h

```
#if !defined(AFX_IGIPRIMITIVE_H__947C44F0_6E13_478B_AD69_FC2EF4860AB9__INCLUDED_)
#define AFX_IGIPRIMITIVE_H__947C44F0_6E13_478B_AD69_FC2EF4860AB9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// IGIprimitive.h : header file
//

#include "Primitive.h"
#include "IgiCircle.h"
#include "IgiBox.h"
#include "IgiTriangle.h"
#include "IgiLine.h"
#include "IgiText.h"
#include "IgiTagName.h"
#include "IgiStream.h"

typedef enum  {
                                    Black,Blue,Green,Cyan,Red,Magenta,Yellow,White,
                                    Darkblue,Darkgreen,Darkcyan,Darkred,Darkmagenta,
                                    Darkyellow,Darkgray,Lightgray
                              } ColorIndex;


typedef enum  {
                                    IgiRectangle,FillRectangle,
                                    IgiTriangle,FillTriangle,
                                    IgiEllipse,FillEllipse,
                                    IgiText,Tag,
                                    ProcessIn,ProcessOut,
                                    LogicIn,LogicOut,
                                    AnalogIn,AnalogOut,
                                    IgiLine,Indicator
                              } PaletteIndex;

typedef enum  {
                                    ADD_PRIMITIVE_MODE,
                                    MOVE_MODE,
                            } EditModeIndex;


typedef enum  {
                                    IGI_PRIM_NONE,

                                    IGI_PRIM_ELLIPSE,
                                    IGI_PRIM_BOX,
                                    IGI_PRIM_LINE,
                                    IGI_PRIM_TRIANGLE,
                                    IGI_PRIM_TAG,
                                    IGI_PRIM_TEXT,
                                    IGI_PRIM_INPUT_ANALOG_STREAM,
                                    IGI_PRIM_OUTPUT_ANALOG_STREAM,
                                    IGI_PRIM_INPUT_LOGIC_STREAM,
                                    IGI_PRIM_OUTPUT_LOGIC_STREAM,
                                    IGI_PRIM_INPUT_PROCESS_STREAM,
                                    IGI_PRIM_OUTPUT_PROCESS_STREAM
                            } PalettePrimitive;

typedef enum  {
                                    IGI_NONE, IGI_ELLIPSE, IGI_BOX,
                                    IGI_LINE, IGI_TRIANGLE, IGI_STREAM,
                                    IGI_TAGNAME, IGI_TEXT, IGI_BOUNDBOX
                            } PrimitiveType;


typedef union PrimitivePointer
{
        CIgiEllipse         *pIGIELLIPSE;       // pointer to ellipse
        CIgiBox                     *pIGIBOX;               // pointer to box
        CIgiTriangle  *pIGITRIANGLE;            // pointer to triangle
        CIgiLine            *pIGILINE;                  // pointer to line
        CIgiText            *pIGITEXT;                  // pointer to text
        CIgiTagName         *pIGITAGNAME;       // pointer to tagname
        CIgiStream          *pIGISTREAM;        // pointer to stream
} ObjectPointer;

/////////////////////////////////////////////////////////////////////////////
// CIGIprimitive window

class CIGIprimitive : public CObject
{
```

```
// Construction
public:
        CIGIprimitive();
        virtual ~CIGIprimitive();
        DECLARE_SERIAL(CIGIprimitive)

// Operations
public:
        virtual void Serialize(CArchive& ar);


// Implementation
public:
        int GetPrimitiveType();
        void SetFillMode(BOOL bFillMode);
        int GetStreamDir();
        int GetStreamType();
        void IncreaseOrientation();
        void SetStreamNumber(int iNoStream);
        int GetStreamNumber(int iType,  int iDir);
        BOOL IsLookForStream(int iType, int iDir);
        CString GetIGItextinput();
        void SetIGItextinput(CString cTextInput);
        BOOL GetFillMode();
        COLORREF GetPenColor();
        PalettePrimitive GetPrimitiveItem();
        void GetParameter(int* x, int dx);
        BOOL GetIGIObjectSelect();
        void UpdateColor(COLORREF currentColor);
        void MoveIGIPrimitive(int dx, int dy);
        void SetIGIObjectSelect(BOOL select);
        void InvertIgiPrimitive(CDC* pDC, int dx, int dy);
        void ModifyPrimitive(int dx, int dy, int index);
        int GetMarkIndex();
        BOOL GetSelectMark();
        BOOL SelectIGIprimitive(CPoint point);
        void SetPenColor(COLORREF currentColor);
        void NewPrimitive();
        void SetParameter(int x[6]);
        void SetPrimitiveType(PalettePrimitive item);
        BOOL DrawIGIprimitive(CDC* pDC);
        CPoint GetModifyPoint(int iMarkIndex, int iFromIndex);
```

```
// Attributes
private:
	ObjectPointer m_pObject;
	int m_iVar[6];
	BOOL m_bFill;
	COLORREF m_cPenColor;
	PrimitiveType m_iType;
	CString m_cTextInput;
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_IGIPRIMITIVE_H__947C44F0_6E13_478B_AD69_FC2EF4860AB9__INCLUDED_)
```

4.2.2 IGIprimitive.cpp

```
// IGIprimitive.cpp : implementation file
//

#include "stdafx.h"
#include "igieditor.h"
#include "IGIprimitive.h"



#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CIGIprimitive
IMPLEMENT_SERIAL(CIGIprimitive, CObject, 1)
CIGIprimitive::CIGIprimitive()
{
}

CIGIprimitive::~CIGIprimitive()
```

```
{
}

/////////////////////////////////////////////////////////////////////////////
// CIGIprimitive message handlers

void CIGIprimitive::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {       // storing code
                ar << (int)m_iType;
        }
        else
        {
                int type;
                ar >> type;
                m_iType = (PrimitiveType)type;

                switch (m_iType)
                {
                case IGI_BOX:
                        m_pObject.pIGIBOX = new CIgiBox();
                        break;

                case IGI_ELLIPSE:
                        m_pObject.pIGIELLIPSE = new CIgiEllipse();
                        break;

                case IGI_LINE:
                        m_pObject.pIGILINE = new CIgiLine();
                        break;

                case IGI_TRIANGLE:
                        m_pObject.pIGITRIANGLE = new CIgiTriangle();
                        break;

                case IGI_TAGNAME:
                        m_pObject.pIGITAGNAME = new CIgiTagName();
                        break;

                case IGI_TEXT:
                        m_pObject.pIGITEXT = new CIgiText();
                        break;
        case IGI_STREAM:
            m_pObject.pIGISTREAM = new CIgiStream();
        default:
            break;
        }
    }

    switch (m_iType)
    {
    case IGI_BOX:
        m_pObject.pIGIBOX->Serialize(ar);
        break;

    case IGI_ELLIPSE:
        m_pObject.pIGIELLIPSE->Serialize(ar);
        break;

    case IGI_LINE:
        m_pObject.pIGILINE->Serialize(ar);
        break;

    case IGI_TRIANGLE:
        m_pObject.pIGITRIANGLE->Serialize(ar);
        break;

    case IGI_TAGNAME:
        m_pObject.pIGITAGNAME->Serialize(ar);
        break;

    case IGI_TEXT:
        m_pObject.pIGITEXT->Serialize(ar);
        break;

    case IGI_STREAM:
        m_pObject.pIGISTREAM->Serialize(ar);
        break;

    default:
        break;
    }
}
```

```
//////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to draw its object on the edit area.
//
//////////////////////////////////////////////////////////////////////////////
BOOL CIGIprimitive::DrawIGIprimitive(CDC *pDC)
{
        switch (m_iType)
        {
        case IGI_BOX:
                m_pObject.pIGIBOX->DrawPrimitive(pDC);
                break;

        case IGI_ELLIPSE:
                m_pObject.pIGIELLIPSE->DrawPrimitive(pDC);
                break;

        case IGI_LINE:
                m_pObject.pIGILINE->DrawPrimitive(pDC);
                break;

        case IGI_TRIANGLE:
                m_pObject.pIGITRIANGLE->DrawPrimitive(pDC);
                break;

        case IGI_TAGNAME:
                m_pObject.pIGITAGNAME->DrawPrimitive(pDC);
                break;

        case IGI_TEXT:
                m_pObject.pIGITEXT->DrawPrimitive(pDC);
                break;

        case IGI_STREAM:
                m_pObject.pIGISTREAM->DrawPrimitive(pDC);
                break;

        default:
                break;
        }
        return TRUE;
}
```

```
/////////////////////////////////////////////////////////////////////////////
//
//      This function sets the current primitive object type when user adds an
//      object to an IGI document. It is called when a new object to be created
//      at adding new object or copying an object.
//
/////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::SetPrimitiveType(PalettePrimitive item)
{
        switch (item)
        {
                case IGI_PRIM_NONE:
                        m_iType = IGI_NONE;
                        break;

                case IGI_PRIM_ELLIPSE:
                        m_iType = IGI_ELLIPSE;
                        break;

                case IGI_PRIM_BOX:
                        m_iType = IGI_BOX;
                        break;

                case IGI_PRIM_LINE:
                        m_iType = IGI_LINE;
                        break;

                case IGI_PRIM_TRIANGLE:
                        m_iType = IGI_TRIANGLE;
                        break;

                case IGI_PRIM_TAG:
                        m_iType = IGI_TAGNAME;
                        break;

                case IGI_PRIM_TEXT:
                        m_iType = IGI_TEXT;
                        break;

                case IGI_PRIM_INPUT_ANALOG_STREAM:
                        m_iType = IGI_STREAM;
                        break;
                case IGI_PRIM_OUTPUT_ANALOG_STREAM:
                        m_iType = IGI_STREAM;
                        break;

                case IGI_PRIM_INPUT_LOGIC_STREAM:
                        m_iType = IGI_STREAM;
                        break;

                case IGI_PRIM_OUTPUT_LOGIC_STREAM:
                        m_iType = IGI_STREAM;
                        break;

                case IGI_PRIM_INPUT_PROCESS_STREAM:
                        m_iType = IGI_STREAM;
                        break;

                case IGI_PRIM_OUTPUT_PROCESS_STREAM:
                        m_iType = IGI_STREAM;
                        break;

                default:
                        break;
        }

}

/////////////////////////////////////////////////////////////////////////////
//
//      This function saves a primitive object parameter to IGIprimitive object.
//      It is called when user adds a new primitive object or copies an object.
//
/////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::SetParameter(int x[])
{
        for (int i = 0; i < 6; i++)
                m_iVar[i] = x[i];
}

/////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to create an object. It will set object
//      color and fill mode. For new object, it is always unselected.
//
```

```
//////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::NewPrimitive()
{
	switch (m_iType)
	{
	case IGI_BOX:
		m_pObject.pIGIBOX = new CIgiBox(m_iVar[0], m_iVar[1], m_iVar[2], m_iVar[3]);
		m_pObject.pIGIBOX->SetPenColor(m_cPenColor);
		m_pObject.pIGIBOX->SetFillMode(m_bFill);
		m_pObject.pIGIBOX->SetObjectSelect(FALSE);
		break;

	case IGI_ELLIPSE:
		m_pObject.pIGIELLIPSE = new CIgiEllipse(m_iVar[0], m_iVar[1], m_iVar[2],
m_iVar[3]);
		m_pObject.pIGIELLIPSE->SetPenColor(m_cPenColor);
		m_pObject.pIGIELLIPSE->SetFillMode(m_bFill);
		m_pObject.pIGIELLIPSE->SetObjectSelect(FALSE);
		break;

	case IGI_LINE:
		m_pObject.pIGILINE = new CIgiLine(m_iVar[0], m_iVar[1], m_iVar[2], m_iVar[3]);
		m_pObject.pIGILINE->SetPenColor(m_cPenColor);
		m_pObject.pIGILINE->SetObjectSelect(FALSE);
		break;

	case IGI_TRIANGLE:
		m_pObject.pIGITRIANGLE = new CIgiTriangle(m_iVar[0], m_iVar[1], m_iVar[2],
m_iVar[3], m_iVar[4], m_iVar[5]);
		m_pObject.pIGITRIANGLE->SetPenColor(m_cPenColor);
		m_pObject.pIGITRIANGLE->SetFillMode(m_bFill);
		m_pObject.pIGITRIANGLE->SetObjectSelect(FALSE);
		break;

	case IGI_TAGNAME:
		m_pObject.pIGITAGNAME = new CIgiTagName(m_iVar[0], m_iVar[1], m_iVar[2],
m_iVar[3]);
		m_pObject.pIGITAGNAME->SetPenColor(m_cPenColor);
		m_pObject.pIGITAGNAME->SetObjectSelect(FALSE);
		break;

	case IGI_TEXT:
		m_pObject.pIGITEXT = new CIgiText(m_iVar[0], m_iVar[1], m_iVar[2], m_iVar[3]);
		m_pObject.pIGITEXT->SetPenColor(m_cPenColor);
		m_pObject.pIGITEXT->SetObjectSelect(FALSE);
		m_pObject.pIGITEXT->SetTextInput(m_cTextInput);
		break;

	case IGI_STREAM:
		m_pObject.pIGISTREAM = new CIgiStream(m_iVar[0], m_iVar[1], m_iVar[2],
m_iVar[3], m_iVar[4], m_iVar[5]);
		m_pObject.pIGISTREAM->SetObjectSelect(FALSE);
		break;

	default:
		break;
	}
}

////////////////////////////////////////////////////////////////////////////
//
//	This function saves display color for primitive object in IGIprimitive object
//
////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::SetPenColor(COLORREF currentColor)
{
	m_cPenColor = currentColor;
}

////////////////////////////////////////////////////////////////////////////
//
//	This function saves fill mode for primitive object to IGIprimitive object
//
////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::SetFillMode(BOOL bFillMode)
{
	m_bFill = bFillMode;
}

////////////////////////////////////////////////////////////////////////////
//
//	This function calls each primitive to check whether the "point" is over
//	the primitive object.
//
////////////////////////////////////////////////////////////////////////////
BOOL CIGIprimitive::SelectIGIprimitive(CPoint point)
{
	BOOL select = FALSE;
```

```
        switch (m_iType)
        {
        case IGI_BOX:
                select = m_pObject.pIGIBOX->SelectPrimitive(point.x, point.y);
                break;

        case IGI_ELLIPSE:
                select = m_pObject.pIGIELLIPSE->SelectPrimitive(point.x, point.y);
                break;

        case IGI_LINE:
                select = m_pObject.pIGILINE->SelectPrimitive(point.x, point.y);
                break;

        case IGI_TRIANGLE:
                select = m_pObject.pIGITRIANGLE->SelectPrimitive(point.x, point.y);
                break;

        case IGI_TAGNAME:
                select = m_pObject.pIGITAGNAME->SelectPrimitive(point.x, point.y);
                break;

        case IGI_TEXT:
                select = m_pObject.pIGITEXT->SelectPrimitive(point.x, point.y);
                break;

        case IGI_STREAM:
                select = m_pObject.pIGISTREAM->SelectPrimitive(point.x, point.y);
                break;

        default:
                break;
        }
        return select;
}

/////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to return whether the mouse is clicked
//      on the select mark. When the mouse is clicked on the select mark, that means
//      user can modify the primitive object. Otherwise, the object will only be
//      moved. Only primitive object "Line, Rectangle/FillRectangle, Ellipse/Fill
//     Ellipse, and Triangle/FillTriangle" have select marks and can be modified.
//
////////////////////////////////////////////////////////////////////////////////
BOOL CIGIprimitive::GetSelectMark()
{
        BOOL selectMark = FALSE;
        switch (m_iType)
        {
        case IGI_BOX:
                selectMark = m_pObject.pIGIBOX->GetSelectMark();
                break;

        case IGI_ELLIPSE:
                selectMark = m_pObject.pIGIELLIPSE->GetSelectMark();

        case IGI_LINE:
                selectMark = m_pObject.pIGILINE->GetSelectMark();
                break;

        case IGI_TRIANGLE:
                selectMark = m_pObject.pIGITRIANGLE->GetSelectMark();
                break;

        case IGI_TAGNAME:
                selectMark = FALSE;
                break;

        case IGI_TEXT:
                selectMark = FALSE;
                break;

        case IGI_STREAM:
                selectMark = FALSE;
                break;

        default:
                break;
        }
        return selectMark;
}

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

```
//      This function calls each primitive to return the mark index on which the
//      mouse is clicked. The primitive Line has two select marks. Triangle has
//      three and Rectangle(Box) has four.
//
//////////////////////////////////////////////////////////////////////////////
int CIGIprimitive::GetMarkIndex()
{
        int markIndex;
        switch (m_iType)
        {
        case IGI_BOX:
                markIndex = m_pObject.pIGIBOX->GetMarkIndex();
                break;

        case IGI_ELLIPSE:
                markIndex = m_pObject.pIGIELLIPSE->GetMarkIndex();
                break;

        case IGI_LINE:
                markIndex = m_pObject.pIGILINE->GetMarkIndex();
                break;

        case IGI_TRIANGLE:
                markIndex = m_pObject.pIGILINE->GetMarkIndex();
                break;

        default:
                break;
        }
        return markIndex;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to update the object status because of
//      the modification.
//
//////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::ModifyPrimitive(int dx, int dy, int index)
{
        switch (m_iType)
        {
        case IGI_BOX:
            m_pObject.pIGIBOX->ModifyPrimitive( dx, dy, index );
        break;

    case IGI_ELLIPSE:
            m_pObject.pIGIELLIPSE->ModifyPrimitive( dx, dy, index );
        break;

    case IGI_LINE:
            m_pObject.pIGILINE->ModifyPrimitive( dx, dy, index );
        break;

    case IGI_TRIANGLE:
            m_pObject.pIGITRIANGLE->ModifyPrimitive( dx, dy, index );
        break;

    default : break;
    }  // end of switch
}

///////////////////////////////////////////////////////////////////////////////
//
//     This function calls each primitive to invert the primitive object during
//     the object is moving.
//
///////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::InvertIgiPrimitive(CDC* pDC, int dx, int dy)
{
    switch (m_iType)
    {
    case IGI_BOX:
        m_pObject.pIGIBOX->InvertIgiPrimitive(pDC, dx, dy);
        break;

    case IGI_ELLIPSE:
        m_pObject.pIGIELLIPSE->InvertIgiPrimitive(pDC, dx, dy);
        break;

    case IGI_LINE:
        m_pObject.pIGILINE->InvertIgiPrimitive(pDC, dx, dy);
        break;

    case IGI_TRIANGLE:
        m_pObject.pIGITRIANGLE->InvertIgiPrimitive(pDC, dx, dy);
```

```
                break;

        case IGI_TAGNAME:
                m_pObject.pIGITAGNAME->InvertIgiPrimitive(pDC, dx, dy);
                break;

        case IGI_TEXT:
                m_pObject.pIGITEXT->InvertIgiPrimitive(pDC, dx, dy);
                break;

        case IGI_STREAM:
                m_pObject.pIGISTREAM->InvertIgiPrimitive(pDC, dx, dy);
                break;

        default :
                break;
        }  //  end of switch
}

////////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to set object select to TRUE or FALSE.
//
////////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::SetIGIObjectSelect(BOOL select)
{
        switch ( m_iType )
        {
        case IGI_BOX:
                m_pObject.pIGIBOX->SetObjectSelect(select);
                break;

        case IGI_ELLIPSE:
                m_pObject.pIGIELLIPSE->SetObjectSelect(select);
                break;

        case IGI_LINE:
                m_pObject.pIGILINE->SetObjectSelect(select);
                break;

        case IGI_TRIANGLE:
                m_pObject.pIGITRIANGLE->SetObjectSelect(select);
                break;
        case IGI_TAGNAME:
              m_pObject.pIGITAGNAME->SetObjectSelect(select);
              break;

        case IGI_TEXT:
              m_pObject.pIGITEXT->SetObjectSelect(select);
              break;

        case IGI_STREAM:
              m_pObject.pIGISTREAM->SetObjectSelect(select);
              break;

        default :
              break;
        }  //  end of switch
}

////////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to update the object position.
//
////////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::MoveIGIPrimitive(int dx, int dy)
{

        switch ( m_iType )
        {
        case IGI_BOX:
                    m_pObject.pIGIBOX->MovePrimitive(dx, dy);
              break;

        case IGI_ELLIPSE:
                    m_pObject.pIGIELLIPSE->MovePrimitive( dx, dy );
              break;

        case IGI_LINE:
                    m_pObject.pIGILINE->MovePrimitive( dx, dy );
              break;

        case IGI_TRIANGLE:
                    m_pObject.pIGITRIANGLE->MovePrimitive( dx, dy );
              break;
```

```
    case IGI_TAGNAME:
            m_pObject.pIGITAGNAME->MovePrimitive( dx, dy );
        break;

    case IGI_TEXT:
            m_pObject.pIGITEXT->MovePrimitive( dx, dy );
        break;

    case IGI_STREAM:
            m_pObject.pIGISTREAM->MovePrimitive( dx, dy );
        break;

    default:
        break;
    }  // end of switch
}

/////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to update the object color.
//
/////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::UpdateColor(COLORREF currentColor)
{
    switch ( m_iType )
    {
    case IGI_BOX:
            m_pObject.pIGIBOX->SetPenColor(currentColor);
        break;

    case IGI_ELLIPSE:
            m_pObject.pIGIELLIPSE->SetPenColor(currentColor);
        break;

    case IGI_LINE:
            m_pObject.pIGIELLIPSE->SetPenColor(currentColor);
        break;

    case IGI_TRIANGLE:
            m_pObject.pIGITRIANGLE->SetPenColor(currentColor);
        break;
        case IGI_TAGNAME:
                        m_pObject.pIGITAGNAME->SetPenColor(currentColor);
                break;

        case IGI_TEXT:
                        m_pObject.pIGITEXT->SetPenColor(currentColor);
                break;

        default :
                break;
        }  // end of switch
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to get status of the object selection.
//
//////////////////////////////////////////////////////////////////////////////
BOOL CIGIprimitive::GetIGIObjectSelect()
{
        BOOL bSelect;
        switch ( m_iType )
        {
        case IGI_BOX:
                bSelect = m_pObject.pIGIBOX->GetObjectSelect();
                break;

        case IGI_ELLIPSE:
                bSelect = m_pObject.pIGIELLIPSE->GetObjectSelect();
                break;

        case IGI_LINE:
                bSelect = m_pObject.pIGILINE->GetObjectSelect();
                break;

        case IGI_TRIANGLE:
                bSelect = m_pObject.pIGITRIANGLE->GetObjectSelect();
                break;

        case IGI_TAGNAME:
                bSelect = m_pObject.pIGITAGNAME->GetObjectSelect();
                break;
        case IGI_TEXT:
                bSelect = m_pObject.pIGITEXT->GetObjectSelect();
                break;

        case IGI_STREAM:
                bSelect = m_pObject.pIGISTREAM->GetObjectSelect();
                break;

        default :break;
        }  //  end of switch
        return bSelect;
}

/////////////////////////////////////////////////////////////////////////////
//
//     This function calls each primitive to get parameters of each object.
//
/////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::GetParameter(int* x, int dx)
{
        switch ( m_iType )
        {
        case IGI_BOX:
                m_pObject.pIGIBOX->GetParameter(x, dx);
                break;

        case IGI_ELLIPSE:
                m_pObject.pIGIELLIPSE->GetParameter(x, dx);
                break;

        case IGI_LINE:
                m_pObject.pIGILINE->GetParameter(x, dx);
                break;

        case IGI_TRIANGLE:
                m_pObject.pIGITRIANGLE->GetParameter(x, dx);
                break;

        case IGI_TAGNAME:
                m_pObject.pIGITAGNAME->GetParameter(x, dx);
                break;

        case IGI_TEXT:
                m_pObject.pIGITEXT->GetParameter(x, dx);;
                break;

        case IGI_STREAM:
                m_pObject.pIGISTREAM->GetParameter(x, dx);
                break;

        default :break;
        }   //  end of switch
}

///////////////////////////////////////////////////////////////////////////////////////
//
//      This function converts primitive type to its proper item.
//
///////////////////////////////////////////////////////////////////////////////////////
PalettePrimitive CIGIprimitive::GetPrimitiveItem()
{
        PalettePrimitive item;
        switch (m_iType)
        {
                case IGI_NONE:
                        item = IGI_PRIM_NONE;
                        break;

                case IGI_ELLIPSE:
                        item  = IGI_PRIM_ELLIPSE;
                        break;

                case IGI_BOX:
                        item = IGI_PRIM_BOX;
                        break;

                case IGI_LINE:
                        item = IGI_PRIM_LINE;
                        break;

                case IGI_TRIANGLE:
                        item = IGI_PRIM_TRIANGLE;
                        break;

                case IGI_TAGNAME:
                        item = IGI_PRIM_TAG;
                        break;

                case IGI_TEXT:
                        item = IGI_PRIM_TEXT;
                        break;

                case IGI_STREAM:
                        item = IGI_PRIM_INPUT_ANALOG_STREAM;
                        break;

                default:
                        break;
        }
        return item;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to get its object color.
//
//////////////////////////////////////////////////////////////////////////////
COLORREF CIGIprimitive::GetPenColor()
{

        COLORREF currentColor;
        switch ( m_iType )
        {
        case IGI_BOX:
                currentColor = m_pObject.pIGIBOX->GetPenColor();
                break;

        case IGI_ELLIPSE:
                currentColor = m_pObject.pIGIELLIPSE->GetPenColor();
                break;

        case IGI_LINE:
                currentColor = m_pObject.pIGILINE->GetPenColor();
                break;

        case IGI_TRIANGLE:
        currentColor = m_pObject.pIGITRIANGLE->GetPenColor();
                break;
        case IGI_TAGNAME:
                currentColor = m_pObject.pIGITAGNAME->GetPenColor();
                break;

        case IGI_TEXT:
                currentColor = m_pObject.pIGITEXT->GetPenColor();
                break;

        default:
                break;
        }   //  end of switch
        return currentColor;
}

////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to get its object fill mode.
//
////////////////////////////////////////////////////////////////////////////
BOOL CIGIprimitive::GetFillMode()
{
        BOOL bFill;
        switch ( m_iType )
        {
        case IGI_BOX:
                bFill = m_pObject.pIGIBOX->GetFillMode();
                break;

        case IGI_ELLIPSE:
                bFill = m_pObject.pIGIELLIPSE->GetFillMode();
                break;

        case IGI_TRIANGLE:
                bFill = m_pObject.pIGITRIANGLE->GetFillMode();
                break;

        default:
                break;
        }   //  end of switch
        return bFill;
}

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

```
//
//      This function sets input text string for the TEXT primitive.
//
//////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::SetIGItextinput(CString cTextInput)
{
        m_cTextInput = cTextInput;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function gets input text string from the TEXT primitive object.
//
//////////////////////////////////////////////////////////////////////////////
CString CIGIprimitive::GetIGItextinput()
{
        return m_cTextInput = m_pObject.pIGITEXT->GetTextInput();
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function checks the stream according to its type and direction.
//
//////////////////////////////////////////////////////////////////////////////
BOOL CIGIprimitive::IsLookForStream(int iType, int iDir)
{
        if (GetStreamType() == iType && GetStreamDir() == iDir)
                return TRUE;
        else
                return FALSE;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function checks the stream according to its type and direction.
//
//////////////////////////////////////////////////////////////////////////////
int CIGIprimitive::GetStreamNumber(int iType, int iDir)
{
        if (GetStreamType() == iType && GetStreamDir() == iDir)
        {
                return m_pObject.pIGISTREAM->GetStreamNumber();
        }
```

```
       else
       {
              return -1;
       }
}

//////////////////////////////////////////////////////////////////////////////
//
//     This function sets the stream number to a stream object.
//
//////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::SetStreamNumber(int iNoStream)
{
       m_pObject.pIGISTREAM->SetStreamNumber(iNoStream);
}

//////////////////////////////////////////////////////////////////////////////
//
//     This function sets the stream orientation for a stream object when a
//     stream has been rotated.
//
//////////////////////////////////////////////////////////////////////////////
void CIGIprimitive::IncreaseOrientation()
{
       m_pObject.pIGISTREAM->IncreaseOrientation();
}

//////////////////////////////////////////////////////////////////////////////
//
//     This function gets a stream type from a stream object.
//
//////////////////////////////////////////////////////////////////////////////
int CIGIprimitive::GetStreamType()
{
       return m_pObject.pIGISTREAM->GetStreamType();
}

//////////////////////////////////////////////////////////////////////////////
//
//     This function gets a stream direction from a stream object.
//
//////////////////////////////////////////////////////////////////////////////
int CIGIprimitive::GetStreamDir()
```

```
{
        return m_pObject.pIGISTREAM->GetStreamDir();
}

/////////////////////////////////////////////////////////////////////////////
//
//      This function gets a primitive object type from IGIprimitive.
//
/////////////////////////////////////////////////////////////////////////////
int CIGIprimitive::GetPrimitiveType()
{
        return m_iType;
}

/////////////////////////////////////////////////////////////////////////////
//
//      This function calls each primitive to get the start point position after
//      the object has been modified.
//
/////////////////////////////////////////////////////////////////////////////
CPoint CIGIprimitive::GetModifyPoint(int iMarkIndex, int iFromIndex )
{
        CPoint ptFrom;
        switch (m_iType)
        {
        case IGI_BOX:
                ptFrom = m_pObject.pIGIBOX->GetModifyPoint(iMarkIndex);
                break;

        case IGI_ELLIPSE:
                ptFrom = m_pObject.pIGIELLIPSE->GetModifyPoint(iMarkIndex);
                break;

        case IGI_LINE:
                ptFrom = m_pObject.pIGILINE->GetModifyPoint(iMarkIndex);
                break;

        case IGI_TRIANGLE:
                ptFrom = m_pObject.pIGITRIANGLE->GetModifyPoint(iMarkIndex, iFromIndex);
                break;

        default:
                break;
        }
        return ptFrom;
}
```

4.2.3 igieditorDoc.cpp

```
// igieditorDoc.cpp : implementation of the CIgieditorDoc class
//

#include "stdafx.h"
#include "igieditor.h"
#include "TextInputDlg.h"

#include "igieditorDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////////
                // CIgieditorDoc

IMPLEMENT_DYNCREATE(CIgieditorDoc, CDocument)

BEGIN_MESSAGE_MAP(CIgieditorDoc, CDocument)
        //{{AFX_MSG_MAP(CIgieditorDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        ON_COMMAND(ID_EDIT_CUT, OnDeleteSome)
        ON_COMMAND(ID_IGICUT_ALL, OnDeleteAll)
        ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
        ON_COMMAND(ID_EDIT_UNDO, OnEditUndo)
        ON_COMMAND(ID_SELECT_ALL, OnSelectAll)
        ON_COMMAND(ID_UNSELECT_ALL, OnUnSelectAll)
        ON_COMMAND(ID_PUSH_ONE, OnPushOne)
        ON_COMMAND(ID_PUSH_BOTTOM, OnPushBottom)
        ON_COMMAND(ID_POP_ONE, OnPopOneUp)
        ON_COMMAND(ID_POP_TOP, OnPopToTop)

        ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
ON_COMMAND(ID_COLOR_YELLOW, OnColorYellow)
ON_COMMAND(ID_COLOR_RED, OnColorRed)
ON_COMMAND(ID_COLOR_BLUE, OnColorBlue)
ON_COMMAND(ID_COLOR_GREEN, OnColorGreen)
ON_COMMAND(ID_COLOR_CYAN, OnColorCyan)
ON_COMMAND(ID_COLOR_MAGENTA, OnColorMagenta)
ON_COMMAND(ID_COLOR_WHITE, OnColorWhite)
ON_COMMAND(ID_COLOR_DARKBLUE, OnColorDarkblue)
ON_COMMAND(ID_COLOR_DARKGREEN, OnColorDarkgreen)
ON_COMMAND(ID_COLOR_DARKCYAN, OnColorDarkcyan)
ON_COMMAND(ID_COLOR_DARKRED, OnColorDarkred)
ON_COMMAND(ID_COLOR_DARKMAGENTA, OnColorDarkmagenta)
ON_COMMAND(ID_COLOR_DARKYELLOW, OnColorDarkyellow)
ON_COMMAND(ID_COLOR_DARKGRAY, OnColorDarkgray)
ON_COMMAND(ID_COLOR_LIGHTGRAY, OnColorLightgray)

ON_COMMAND(ID_PALETTE_RECTANGLE, OnPaletteRectangle)
ON_COMMAND(ID_PALETTE_FILLRECTANGLE, OnPaletteFillRectangle)
ON_COMMAND(ID_PALETTE_TRIANGLE, OnPaletteTriangle)
ON_COMMAND(ID_PALETTE_FILLTRIANGLE, OnPaletteFillTriangle)
ON_COMMAND(ID_PALETTE_ELLIPSE, OnPaletteEllipse)
ON_COMMAND(ID_PALETTE_FILLELLIPSE, OnPaletteFillEllipse)
ON_COMMAND(ID_PALETTE_TEXT, OnPaletteText)
ON_COMMAND(ID_PALETTE_TAG, OnPaletteTag)
ON_COMMAND(ID_PALETTE_PROCESSIN, OnPaletteProcessIn)
ON_COMMAND(ID_PALETTE_PROCESSOUT, OnPaletteProcessOut)
ON_COMMAND(ID_PALETTE_LOGICIN, OnPaletteLogicIn)
ON_COMMAND(ID_PALETTE_LOGICOUT, OnPaletteLogicOut)
ON_COMMAND(ID_PALETTE_ANALOGIN, OnPaletteAnalogIn)
ON_COMMAND(ID_PALETTE_ANALOGOUT, OnPaletteAnalogOut)
ON_COMMAND(ID_PALETTE_LINE, OnPaletteLine)
ON_COMMAND(ID_PALETTE_INDICATOR, OnPaletteIndicator)

ON_UPDATE_COMMAND_UI(ID_SELECT_ALL, OnUpdateSelectAll)
ON_UPDATE_COMMAND_UI(ID_UNSELECT_ALL, OnUpdateUnSelectAll)
ON_UPDATE_COMMAND_UI(ID_IGICUT_ALL, OnUpdateDeleteAll)
ON_UPDATE_COMMAND_UI(ID_EDIT_CUT, OnUpdateDeleteSome)
ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
ON_UPDATE_COMMAND_UI(ID_EDIT_UNDO, OnUpdateEditUndo)
ON_UPDATE_COMMAND_UI(ID_PUSH_ONE, OnUpdatePushOne)
ON_UPDATE_COMMAND_UI(ID_PUSH_BOTTOM, OnUpdatePushBottom)
ON_UPDATE_COMMAND_UI(ID_POP_ONE, OnUpdatePopOneUp)
ON_UPDATE_COMMAND_UI(ID_POP_TOP, OnUpdatePopToTop)
```

```
	ON_UPDATE_COMMAND_UI(ID_COLOR_BLACK, OnUpdateColorBlack)
	ON_UPDATE_COMMAND_UI(ID_COLOR_YELLOW, OnUpdateColorYellow)
	ON_UPDATE_COMMAND_UI(ID_COLOR_RED, OnUpdateColorRed)
	ON_UPDATE_COMMAND_UI(ID_COLOR_BLUE, OnUpdateColorBlue)
	ON_UPDATE_COMMAND_UI(ID_COLOR_GREEN, OnUpdateColorGreen)
	ON_UPDATE_COMMAND_UI(ID_COLOR_CYAN, OnUpdateColorCyan)
	ON_UPDATE_COMMAND_UI(ID_COLOR_MAGENTA, OnUpdateColorMagenta)
	ON_UPDATE_COMMAND_UI(ID_COLOR_WHITE, OnUpdateColorWhite)
	ON_UPDATE_COMMAND_UI(ID_COLOR_DARKBLUE, OnUpdateColorDarkblue)
	ON_UPDATE_COMMAND_UI(ID_COLOR_DARKGREEN, OnUpdateColorDarkgreen)
	ON_UPDATE_COMMAND_UI(ID_COLOR_DARKCYAN, OnUpdateColorDarkcyan)
	ON_UPDATE_COMMAND_UI(ID_COLOR_DARKRED, OnUpdateColorDarkred)
	ON_UPDATE_COMMAND_UI(ID_COLOR_DARKMAGENTA, OnUpdateColorDarkmagenta)
	ON_UPDATE_COMMAND_UI(ID_COLOR_DARKYELLOW, OnUpdateColorDarkyellow)
	ON_UPDATE_COMMAND_UI(ID_COLOR_DARKGRAY, OnUpdateColorDarkgray)
	ON_UPDATE_COMMAND_UI(ID_COLOR_LIGHTGRAY, OnUpdateColorLightgray)

	ON_UPDATE_COMMAND_UI(ID_PALETTE_RECTANGLE, OnUpdatePaletteRectangle)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_FILLRECTANGLE, OnUpdatePaletteFillRectangle)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_TRIANGLE, OnUpdatePaletteTriangle)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_FILLTRIANGLE, OnUpdatePaletteFillTriangle)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_ELLIPSE, OnUpdatePaletteEllipse)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_FILLELLIPSE, OnUpdatePaletteFillEllipse)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_TEXT, OnUpdatePaletteText)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_TAG, OnUpdatePaletteTag)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_PROCESSIN, OnUpdatePaletteProcessIn)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_PROCESSOUT, OnUpdatePaletteProcessOut)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_LOGICIN, OnUpdatePaletteLogicIn)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_LOGICOUT, OnUpdatePaletteLogicOut)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_ANALOGIN, OnUpdatePaletteAnalogIn)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_ANALOGOUT, OnUpdatePaletteAnalogOut)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_LINE, OnUpdatePaletteLine)
	ON_UPDATE_COMMAND_UI(ID_PALETTE_INDICATOR, OnUpdatePaletteIndicator)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIgieditorDoc construction/destruction

CIgieditorDoc::CIgieditorDoc()
{
	// TODO: add one-time construction code here
```

```
}

CIgieditorDoc::~CIgieditorDoc()
{
}

/////////////////////////////////////////////////////////////////////////////
//
//      This function creates a new igi document with MFC function OnNewDocument()
//      It calls InitDocument() to initialize the new document.
//
/////////////////////////////////////////////////////////////////////////////
BOOL CIgieditorDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
        InitDocument();
        return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
//
//      This function opens an existed igi document with MFC function OnOpenDocument
//      It calls InitDocument() to initialize the opened document.
//
/////////////////////////////////////////////////////////////////////////////
BOOL CIgieditorDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
        if (!CDocument::OnOpenDocument(lpszPathName))
                return FALSE;
        InitDocument();
        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CIgieditorDoc serialization

void CIgieditorDoc::Serialize(CArchive& ar)
```

```
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
        ar << m_iTagNameFlag;
    }
    else
    {
        // TODO: add loading code here
        ar >> m_iTagNameFlag;
    }
    m_primitiveList.Serialize(ar);
}

/////////////////////////////////////////////////////////////////////////////
// CIgieditorDoc diagnostics

#ifdef _DEBUG
void CIgieditorDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CIgieditorDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CIgieditorDoc commands

/////////////////////////////////////////////////////////////////////////////
//
//    This command sets all primitive objects to be selected in the edit area.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnSelectAll()
{
    SetIGIObjectSelectList(TRUE);
    UpdateAllViews(NULL);
}
```

```
//////////////////////////////////////////////////////////////////////////////
//
//      This command sets all primitive objects to be unselected in the edit area.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUnSelectAll()
{
        SetIGIObjectSelectList(FALSE);
        UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command deletes all primitive objects in the edit area.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnDeleteAll()
{
        CreatePreviousList();       // Create a backup list of the primitives for purposes
                                           // of Undo command.
        DeleteContents();
        SetModifiedFlag();  // Mark the document as having been modified, for
                                           // purposes of confirming File Close.
        UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command deletes all selected primitive objects in the edit area.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnDeleteSome()
{
        if (m_iCountSelect == 0)    // if there is no primitive selected, return.
                return;

        CreatePreviousList();                   // for purpose of Undo

        POSITION pos = m_primitiveList.GetHeadPosition();
        POSITION posDelete;
```

```
    while (pos != NULL)
    {
        posDelete = pos;
        CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
        if (pIGIprimitive->GetIGIObjectSelect())  // the object is selected
        {
            // set tagname flag to allow adding tagname later.
            if (pIGIprimitive->GetPrimitiveType() == IGI_TAGNAME)
            {

                m_iTagNameFlag = 0;
            }
            m_primitiveList.RemoveAt(posDelete);
        }
    }
    m_iCountSelect = 0;
    SetModifiedFlag();
    UpdateAllViews(NULL);
}


/////////////////////////////////////////////////////////////////////////////
//
//      This command copies all selected primitive objects in the edit area.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnEditCopy()
{
    if (m_iCountSelect == 0)    // if there is no primitive selected, do nothing.
        return;

    CreatePreviousList();                       // for purpose of Undo

    POSITION pos = m_primitiveList.GetHeadPosition();
    while (pos != NULL)
    {
        CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
        if (pIGIprimitive->GetIGIObjectSelect())  // the object is selected
        {
            if (pIGIprimitive->GetPrimitiveType() != IGI_TAGNAME) // don't copy
tagname
            {
                CIGIprimitive* pIgiPrimitive = CopyIGIprimitive(pIGIprimitive,
12);
```

```
                                m_primitiveList.AddTail(pIgiPrimitive);
                        }
                }
        }
        UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command undoes the last edit action.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnEditUndo()
{
        if (IsModified())    // if the document is modified
        {
                SwapList();          // set backup list as current working list
                SetModifiedFlag();
                UpdateAllViews(NULL);
        }
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command pushes a selected primitive object one level down.
//      It swaps the selected object node position with its previous node
//      in the primitive list.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPushOne()
{
        if (m_iCountSelect != 1)    // if more than one objects selected, do nothing.
        {
                return;
        }

        CreatePreviousList();
        POSITION posPush;
        POSITION pos = m_primitiveList.GetTailPosition();
        while (pos != NULL)
        {
```

```
        posPush = pos;

        CIGIprimitive* pIGIprimitive = m_primitiveList.GetPrev(pos);
        if (pIGIprimitive->GetIGIObjectSelect())  // the object is selected
        {
            m_primitiveList.InsertBefore(pos, pIGIprimitive);
            m_primitiveList.RemoveAt(posPush);
            UpdateAllViews(NULL);
            return;
        }
    }
}


///////////////////////////////////////////////////////////////////////////
//
//    This command pushes a selected primitve object to bottom level.
//    It moves the selected object position to the head of the primitive
//    list.
//
///////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPushBottom()
{
    if (m_iCountSelect != 1)
    {
        return;
    }

    CreatePreviousList();
    POSITION posPush;
    POSITION pos = m_primitiveList.GetTailPosition();
    while (pos != NULL)
    {
        posPush = pos;
        CIGIprimitive* pIGIprimitive = m_primitiveList.GetPrev(pos);
        if (pIGIprimitive->GetIGIObjectSelect())  // the object is selected
        {
            m_primitiveList.AddHead(pIGIprimitive);
            m_primitiveList.RemoveAt(posPush);
            UpdateAllViews(NULL);
            return;
        }
    }
```

```
}


////////////////////////////////////////////////////////////////////////////
//
//      This command pops a selected primitive object one level up.
//      It swaps the selected object position with its next node
//      in the primitive list.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPopOneUp()
{
        if (m_iCountSelect != 1)
        {
                return;
        }

        CreatePreviousList();
        POSITION posPop;
        POSITION pos = m_primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                posPop = pos;
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
                if (pIGIprimitive->GetIGIObjectSelect())  // the object is selected
                {
                        m_primitiveList.InsertAfter(pos, pIGIprimitive);
                        m_primitiveList.RemoveAt(posPop);
                        UpdateAllViews(NULL);
                        return;
                }
        }
}


////////////////////////////////////////////////////////////////////////////
//
//      This command pops a selected primitive object to top level.
//      It moves the selected object position to the tail of the primitive
//      list.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPopToTop()
```

```
{
        if (m_iCountSelect != 1)
        {
                return;
        }

        CreatePreviousList();
        POSITION posPop;
        POSITION pos = m_primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                posPop = pos;
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
                if (pIGIprimitive->GetIGIObjectSelect())  // the object is selected
                {
                        m_primitiveList.AddTail(pIGIprimitive);
                        m_primitiveList.RemoveAt(posPop);
                        UpdateAllViews(NULL);
                        return;
                }
        }
}


/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the "Select All" button on the main toolbar.
//      The button is disabled if there is no primitive in the edit area.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateSelectAll(CCmdUI* pCmdUI)
{
        if (m_primitiveList.GetCount() == 0)
                pCmdUI->SetCheck(TRUE);
        else
                pCmdUI->SetCheck(FALSE);
}


/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the "Unselect All" button on the main toolbar.
//      The button is disabled if there is no object selected in the edit area.
```

```
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateUnSelectAll(CCmdUI* pCmdUI)
{
       if (m_iCountSelect == 0)
              pCmdUI->SetCheck(TRUE);
       else
              pCmdUI->SetCheck(FALSE);
}


////////////////////////////////////////////////////////////////////////////
//
//     This command updates the "Cut All" button on the main toolbar.
//     The button is disabled if there is no primitive object in the edit area.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateDeleteAll(CCmdUI* pCmdUI)
{
       if (m_primitiveList.GetCount() == 0)
              pCmdUI->SetCheck(TRUE);
       else
              pCmdUI->SetCheck(FALSE);
}


////////////////////////////////////////////////////////////////////////////
//
//     This command updates the "Cut" button on the main toolbar.
//     The button is checked if there is no primitive object selected in the
//     edit area.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateDeleteSome(CCmdUI* pCmdUI)
{
       if (m_iCountSelect == 0)
              pCmdUI->SetCheck(TRUE);
       else
              pCmdUI->SetCheck(FALSE);
}


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

```
//
//     This command updates the "Copy" button on the main toolbar.
//     The button is checked if there is no primitive object selected in the
//     edit area.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
       if (m_iCountSelect == 0)
              pCmdUI->SetCheck(TRUE);
       else
              pCmdUI->SetCheck(FALSE);
}


//////////////////////////////////////////////////////////////////////////////
//
//     This command updates the "Undo" button on the main toolbar.
//     The button is checked if there is no change for the IGI document.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateEditUndo(CCmdUI* pCmdUI)
{
       if (IsModified( ))
              pCmdUI->SetCheck(FALSE);
       else
              pCmdUI->SetCheck(TRUE);
}


//////////////////////////////////////////////////////////////////////////////
//
//     This command updates the "Push Down" button on the main toolbar.
//     The button is checked if the number of selected primitive object
//     is not equal to 1.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePushOne(CCmdUI* pCmdUI)
{
       if (m_iCountSelect != 1)
              pCmdUI->SetCheck(TRUE);
       else
              pCmdUI->SetCheck(FALSE);
}


/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the "Push To Bottom" button on the main toolbar.
//      The button is checked if the number of selected primitive object
//      is not equal to 1.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePushBottom(CCmdUI* pCmdUI)
{
        if (m_iCountSelect != 1)
                pCmdUI->SetCheck(TRUE);
        else
                pCmdUI->SetCheck(FALSE);
}


/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the "Pop Up" button on the main toolbar.
//      The button is checked if the number of selected primitive object
//      is not equal to 1.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePopOneUp(CCmdUI* pCmdUI)
{
        if (m_iCountSelect != 1)
                pCmdUI->SetCheck(TRUE);
        else
                pCmdUI->SetCheck(FALSE);
}


/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the "Pop To Top" button on the main toolbar.
//      The button is checked if the number of selected primitive object
//      is not equal to 1.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePopToTop(CCmdUI* pCmdUI)
```

```
{
        if (m_iCountSelect != 1)
                pCmdUI->SetCheck(TRUE);
        else
                pCmdUI->SetCheck(FALSE);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command sets black color as default color for adding new primitive
//      object. It changes the selected primitive object's color to black and
//      checks the black color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorBlack()
{
        CreatePreviousList();               // for purpose of Undo
        m_cCurrentColor = RGB(0, 0, 0);     // set current color to black
        SetColorBar(Black);                 // check color button
        UpdateColor();                              // change the selected primitive objects
color
        UpdateAllViews(NULL);               // redraw all primitive objects.
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command sets yellow color as default color for adding new primitive
//      object. It changes the selected primitive object's color to yellow and
//      checks the yellow color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorYellow()
{
        CreatePreviousList();
        m_cCurrentColor = RGB(255, 255, 0);
        SetColorBar(Yellow);
        UpdateColor();
        UpdateAllViews(NULL);
}


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

```
//
//     This command sets red color as default color for adding new primitive.
//  It changes the selected primitive's color to red and checks the red
//     color button.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorRed()
{
	CreatePreviousList();
	m_cCurrentColor = RGB(255, 0, 0);
	SetColorBar(Red);
	UpdateColor();
	UpdateAllViews(NULL);
}


/////////////////////////////////////////////////////////////////////////////
//
//     This command sets blue color as default color for adding new primitive.
//  It changes the selected primitive's color to blue and checks the blue
//     color button.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorBlue()
{
	CreatePreviousList();
	m_cCurrentColor = RGB(0, 0, 255);
	SetColorBar(Blue);
	UpdateColor();
	UpdateAllViews(NULL);
}


/////////////////////////////////////////////////////////////////////////////
//
//     This command sets green color as default color for adding new primitive.
//  It changes the selected primitive's color to green and checks the green
//     color button.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorGreen()
{
	CreatePreviousList();
```

```
        m_cCurrentColor = RGB(0, 255, 0);
        SetColorBar(Green);
        UpdateColor();
        UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command sets cyan color as default color for adding new primitive.
//  It changes the selected primitive's color to cyan and checks the cyan
//      color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorCyan()
{
        CreatePreviousList();
        m_cCurrentColor = RGB(0, 255, 255);
        SetColorBar(Cyan);
        UpdateColor();
        UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command sets magenta color as default color for adding new primitive.
//  It changes the selected primitive's color to magenta and checks the magenta
//      color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorMagenta()
{
        CreatePreviousList();
        m_cCurrentColor = RGB(255, 0, 255);
        SetColorBar(Magenta);
        UpdateColor();
        UpdateAllViews(NULL);
}


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

```
//      This command sets white color as default color for adding new primitive.
//   It changes the selected primitive's color to white and checks the white
//      color button.
//
//////////////////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorWhite()
{
        CreatePreviousList();
        m_cCurrentColor = RGB(255, 255, 255);
        SetColorBar(White);
        UpdateColor();
        UpdateAllViews(NULL);
}


///////////////////////////////////////////////////////////////////////////////////!//////////////
//
//      This command sets darkblue color as default color for adding new primitive.
//   It changes the selected primitive's color to darkblue and checks the dark
//      blue color button.
//
//////////////////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorDarkblue()
{
        CreatePreviousList();
        m_cCurrentColor = RGB(0, 0, 128);
        SetColorBar(Darkblue);
        UpdateColor();
        UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////////////////////////
//
//      This command sets darkgreen color as default color for adding new primitive.
//   It changes the selected primitive's color to darkgreen and checks the dark
//      green color button.
//
//////////////////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorDarkgreen()
{
        CreatePreviousList();
        m_cCurrentColor = RGB(0, 128, 0);
```

```
       SetColorBar(Darkgreen);
       UpdateColor();
       UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//     This command sets darkcyan color as default color for adding new primitive.
//  It changes the selected primitive's color to darkcyan and checks the dark
//     cyan color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorDarkcyan()
{
       CreatePreviousList();
       m_cCurrentColor = RGB(0, 128, 128);
       SetColorBar(Darkcyan);
       UpdateColor();
       UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//     This command sets darkred color as default color for adding new primitive.
//  It changes the selected primitive's color to darkred and checks the dark
//     red color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorDarkred()
{
       CreatePreviousList();
       m_cCurrentColor = RGB(128, 0, 0);
       SetColorBar(Darkred);
       UpdateColor();
       UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//     This command sets darkmagenta color as default color for adding new primitive.
//  It changes the selected primitive's color to darkmagenta and checks the dark
//     magenta color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorDarkmagenta()
{
       CreatePreviousList();
       m_cCurrentColor = RGB(128, 0, 128);
       SetColorBar(Darkmagenta);
       UpdateColor();
       UpdateAllViews(NULL);
}


//////////////////////i///////////////////////////////////////////////////////
//
//     This command sets darkyellow color as default color for adding new primitive.
//  It changes the selected primitive's color to darkyellow and checks the dark
//     yellow color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorDarkyellow()
{
       CreatePreviousList();
       m_cCurrentColor = RGB(128, 128, 0);
       SetColorBar(Darkyellow);
       UpdateColor();
       UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//     This command sets darkgray color as default color for adding new primitive.
//  It changes the selected primitive's color to darkgray and checks the dark
//     gray color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorDarkgray()
{
       CreatePreviousList();
       m_cCurrentColor = RGB(128, 128, 128);
       SetColorBar(Darkgray);
```

```
        UpdateColor();
        UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command sets lightgray color as default color for adding new primitive.
//  It changes the selected primitive's color to lightgray and checks the light
//      gray color button.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnColorLightgray()
{
        CreatePreviousList();
        m_cCurrentColor = RGB(192, 192, 192);
        SetColorBar(Lightgray);
        UpdateColor();
        UpdateAllViews(NULL);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the black color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorBlack(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Black]);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the yellow color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorYellow(CCmdUI* pCmdUI)
```

```
{
        pCmdUI->SetCheck(m_iColorPalette[Yellow]);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the red color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorRed(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Red]);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the blue color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorBlue(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Blue]);
}


//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the green color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorGreen(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Green]);
}
```

```
////////////////////////////////////////////////////////////////////////////////////
//
//      This command updates the cyan color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorCyan(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Cyan]);
}


////////////////////////////////////////////////////////////////////////////////////
//
//      This command updates the magenta color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorMagenta(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Magenta]);
}


////////////////////////////////////////////////////////////////////////////////////
//
//      This command updates the white color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorWhite(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[White]);
}


////////////////////////////////////////////////////////////////////////////////////
//
//      This command updates the darkblue color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorDarkblue(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Darkblue]);
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the darkgreen color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorDarkgreen(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Darkgreen]);
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the darkcyan color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorDarkcyan(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Darkcyan]);
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the darkred color button on the color bar.
//      The button is checked if the color button is clicked and it is unchecked
//      when other color button is clicked.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorDarkred(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Darkred]);
}

///////////////////////////////////////////////////////////////////////////////
//
//       This command updates the darkmagenta color button on the color bar.
//       The button is checked if the color button is clicked and it is unchecked
//       when other color button is clicked.
//
///////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorDarkmagenta(CCmdUI* pCmdUI)
{
       pCmdUI->SetCheck(m_iColorPalette[Darkmagenta]);
}

///////////////////////////////////////////////////////////////////////////////
//
//       This command updates the darkyellow color button on the color bar.
//       The button is checked if the color button is clicked and it is unchecked
//       when other color button is clicked.
//
///////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorDarkyellow(CCmdUI* pCmdUI)
{
       pCmdUI->SetCheck(m_iColorPalette[Darkyellow]);
}

///////////////////////////////////////////////////////////////////////////////
//
//       This command updates the darkgray color button on the color bar.
//       The button is checked if the color button is clicked and it is unchecked
//       when other color button is clicked.
//
///////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorDarkgray(CCmdUI* pCmdUI)
{
       pCmdUI->SetCheck(m_iColorPalette[Darkgray]);
}

///////////////////////////////////////////////////////////////////////////////
//
//       This command updates the lightgray color button on the color bar.
//       The button is checked if the color button is clicked and it is unchecked
//       when other color button is clicked.
```

```
//
////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdateColorLightgray(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iColorPalette[Lightgray]);
}

////////////////////////////////////////////////////////////////////////////////
//
//      This command sets Rectangle as default primitive for adding new primitive
//      object. It checks the Rectangle button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteRectangle()
{
        CreatePreviousList();
        SetPaletteBar(IgiRectangle);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_BOX;
        m_bFillMode = FALSE;
}

////////////////////////////////////////////////////////////////////////////////
//
//      This command sets Fill Rectangle as default primitive for adding new
//      primitive. It checks the Fill Rectangle button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteFillRectangle()
{
        CreatePreviousList();
        SetPaletteBar(FillRectangle);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_BOX;
        m_bFillMode = TRUE;
}

////////////////////////////////////////////////////////////////////////////////
//
//      This command sets Triangle as default primitive for adding new
//      primitive. It checks the Triangle button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////////
```

```
void CIgieditorDoc::OnPaletteTriangle()
{
        CreatePreviousList();
        SetPaletteBar(IgiTriangle);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_TRIANGLE;
        m_bFillMode = FALSE;
        m_iSelectOrder = 1;
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command sets Fill Triangle as default primitive for adding new
//      primitive. It checks the Fill Triangle button on the primitive bar.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteFillTriangle()
{
        CreatePreviousList();
        SetPaletteBar(FillTriangle);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_TRIANGLE;
        m_bFillMode = TRUE;
        m_iSelectOrder = 1;
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command sets Ellipse as default primitive for adding new
//      primitive. It checks the Ellipse button on the primitive bar.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteEllipse()
{
        CreatePreviousList();
        SetPaletteBar(IgiEllipse);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_ELLIPSE;
        m_bFillMode = FALSE;
}

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

```
//      This command sets Fill Ellipse as default primitive for adding new
//      primitive. It checks the Fill Ellipse button on the primitive bar.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteFillEllipse()
{
        CreatePreviousList();
        SetPaletteBar(FillEllipse);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_ELLIPSE;
        m_bFillMode = TRUE;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command sets Text as default primitive for adding new
//      primitive. It checks the Text button on the primitive bar.
//      It pops up the text input dialog to allow user type added text.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteText()
{
        CreatePreviousList();
        SetPaletteBar(IgiText);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_TEXT;
        CTextInputDlg TextInputDlg;         // show text input dialog box
        TextInputDlg.DoModal();
        m_cTextInput = TextInputDlg.m_cTextInput; // get input text
        if (m_cTextInput == "")             // there is no input
        {
                m_iEditMode = MOVE_MODE;
                m_iCurrentPrimitive = IGI_PRIM_NONE;
                UnSetPaletteBar();
        }
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command sets Tag as default primitive for adding new
//      primitive object. It checks the Tag button on the primitive bar.
//
//////////////////////////////////////////////////////////////////////////////
```

```
void CIgieditorDoc::OnPaletteTag()
{
       CreatePreviousList();
       SetPaletteBar(Tag);
       m_iEditMode = ADD_PRIMITIVE_MODE;
       m_iCurrentPrimitive = IGI_PRIM_TAG;
}

////////////////////////////////////////////////////////////////////////////////////
//
//     This command sets Process Input as default primitive for adding new
//     primitive object. It checks the Process Input button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteProcessIn()
{
       CreatePreviousList();
       SetPaletteBar(ProcessIn);
       m_iEditMode = ADD_PRIMITIVE_MODE;
       m_iCurrentPrimitive = IGI_PRIM_INPUT_PROCESS_STREAM;
}

////////////////////////////////////////////////////////////////////////////////////
//
//     This command sets Process Output as default primitive for adding new
//     primitive object. It checks the Process Output button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteProcessOut()
{
       CreatePreviousList();
       SetPaletteBar(ProcessOut);
       m_iEditMode = ADD_PRIMITIVE_MODE;
       m_iCurrentPrimitive = IGI_PRIM_OUTPUT_PROCESS_STREAM;
}

////////////////////////////////////////////////////////////////////////////////////
//
//     This command sets Logic Input as default primitive for adding new
//     primitive object. It checks the Logic Input button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteLogicIn()
```

```
{
        CreatePreviousList();
        SetPaletteBar(LogicIn);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_INPUT_LOGIC_STREAM;
}

////////////////////////////////////////////////////////////////////////////
//
//      This command sets Logic Output as default primitive for adding new
//      primitive object. It checks the Logic Output button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteLogicOut()
{
        CreatePreviousList();
        SetPaletteBar(LogicOut);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_OUTPUT_LOGIC_STREAM;
}

////////////////////////////////////////////////////////////////////////////
//
//      This command sets Analog Input as default primitive for adding new
//      primitive object. It checks the Analog Input button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteAnalogIn()
{
        CreatePreviousList();
        SetPaletteBar(AnalogIn);
        m_iEditMode = ADD_PRIMITIVE_MODE;
        m_iCurrentPrimitive = IGI_PRIM_INPUT_ANALOG_STREAM;
}

////////////////////////////////////////////////////////////////////////////
//
//      This command sets Analog Output as default primitive for adding new
//      primitive object. It checks the Analog Output button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteAnalogOut()
{
       CreatePreviousList();
       SetPaletteBar(AnalogOut);
       m_iEditMode = ADD_PRIMITIVE_MODE;
       m_iCurrentPrimitive = IGI_PRIM_OUTPUT_ANALOG_STREAM;
}

////////////////////////////////////////////////////////////////////////////
//
//     This command sets Line as default primitive for adding new
//     primitive object. It checks the Line button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteLine()
{
       CreatePreviousList();
       SetPaletteBar(IgiLine);
       m_iEditMode = ADD_PRIMITIVE_MODE;
       m_iCurrentPrimitive = IGI_PRIM_LINE;
}

////////////////////////////////////////////////////////////////////////////
//
//     This command finishes adding primitive action and sets to move mode.
//     It unchecks other primitive button on the primitive bar.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnPaletteIndicator()
{
       SetPaletteBar(Indicator);
       m_iEditMode = MOVE_MODE;
       m_iCurrentPrimitive = IGI_PRIM_NONE;
}

////////////////////////////////////////////////////////////////////////////
//
//     This command updates the Rectangle primitive button on the primitive bar.
//     The button is checked if the primitive button is clicked and it is unchecked
//     when other primitive button is clicked.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteRectangle(CCmdUI* pCmdUI)
{
       pCmdUI->SetCheck(m_iPrimitivePalette[IgiRectangle]);
```

```
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the FillRectangle primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteFillRectangle(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[FillRectangle]);
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Triangle primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteTriangle(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[IgiTriangle]);
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the FillTriangle primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteFillTriangle(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[FillTriangle]);
}

/////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Ellipse primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
```

```
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteEllipse(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(m_iPrimitivePalette[IgiEllipse]);
}

////////////////////////////////////////////////////////////////////////////////////
//
//	This command updates the FillEllipse primitive button on the primitive bar.
//	The button is checked if the primitive button is clicked and it is unchecked
//	when other primitive button is clicked.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteFillEllipse(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(m_iPrimitivePalette[FillEllipse]);
}

////////////////////////////////////////////////////////////////////////////////////
//
//	This command updates the Text primitive button on the primitive bar.
//	The button is checked if the primitive button is clicked and it is unchecked
//	when other primitive button is clicked.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteText(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(m_iPrimitivePalette[IgiText]);
}

////////////////////////////////////////////////////////////////////////////////////
//
//	This command updates the Tag primitive button on the primitive bar.
//	The button is checked if the primitive button is clicked and it is unchecked
//	when other primitive button is clicked.
//
////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteTag(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(m_iPrimitivePalette[Tag]);
}
```

```
//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Process Input primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteProcessIn(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[ProcessIn]);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Process Output primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteProcessOut(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[ProcessOut]);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Logic Input primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteLogicIn(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[LogicIn]);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Logic Output primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
```

```
void CIgieditorDoc::OnUpdatePaletteLogicOut(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[LogicOut]);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Analog Input primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteAnalogIn(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[AnalogIn]);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Analog Output primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteAnalogOut(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[AnalogOut]);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command updates the Line primitive button on the primitive bar.
//      The button is checked if the primitive button is clicked and it is unchecked
//      when other primitive button is clicked.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteLine(CCmdUI* pCmdUI)
{
        pCmdUI->SetCheck(m_iPrimitivePalette[IgiLine]);
}

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

```
//     This command updates the Indicator button on the primitive bar.
//     Clicking this button will uncheck other checked primitive button and
//     change edit mode from ADD MODE to MOVE MODE.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnUpdatePaletteIndicator(CCmdUI* pCmdUI)
{
       pCmdUI->SetCheck(m_iPrimitivePalette[Indicator]);
}

////////////////////////////////////////////////////////////////////////////
//
//     This function initializes the color bar and primitive bar. It sets black
//     color as default color and MOVE MODE as default editor mode.
//     It is called by commands OnNewDocument and OnOpenDocument.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::InitDocument()
{
       m_iEditMode = MOVE_MODE;
       m_iCountSelect = 0;

       for (int i = Black; i <= Lightgray; i++)
              m_iColorPalette[i] = FALSE;

       for (i = IgiRectangle; i <= Indicator; i++)
              m_iPrimitivePalette[i] = FALSE;

       m_iColorPalette[Black] = TRUE;     // Default color is black
}

////////////////////////////////////////////////////////////////////////////
//
//     This function sets the color bar according to the mouse click on the
//     color bar. It sets the color button which is clicked to TRUE and all other
//     color buttons to FALSE. It is called when the user clicks on the color bar.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::SetColorBar(int index)
{
       int i;
       if (index == Black)         // if click on the first button
       {
```

```
                m_iColorPalette[index] = TRUE;
                for (i = Blue; i < Lightgray + 1; i++)
                {
                        m_iColorPalette[i] = FALSE;
                }
        }
        else if (index == Lightgray)      // if click on the last button
        {
                m_iColorPalette[index] = TRUE;
                for (i = Black; i < Lightgray; i++)
                {
                        m_iColorPalette[i] = FALSE;
                }
        }
        else   // if click on the any button between the Black and Lightgray
        {
                m_iColorPalette[index] = TRUE;
                for (i = Black; i < index; i++)
                {
                        m_iColorPalette[i] = FALSE;
                }

                for (i = index + 1; i < Lightgray + 1; i++)
                {
                        m_iColorPalette[i] = FALSE;
                }
        }
}

/////////////////////////////////////////////////////////////////////////////
//
//      This function sets the primitive bar according to the mouse click on the
//      primitive bar. It sets the primitive button which is clicked to TRUE and
//      all other primitive buttons to FALSE. It is called when the user clicks on
//      the primitive bar.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::SetPaletteBar(int index)
{
        int i;
        if (index == IgiRectangle)  // click on the first one, Rectangle
        {
                m_iPrimitivePalette[index] = TRUE;
```

```
		for (i = FillRectangle; i < Indicator + 1; i++)
		{
			m_iPrimitivePalette[i] = FALSE;
		}
	}
	else if (index == Indicator)	// click on the first one, Indicator
	{
		m_iPrimitivePalette[index] = FALSE;
		for (i = IgiRectangle; i < Indicator; i++)
		{
			m_iPrimitivePalette[i] = FALSE;
		}
	}
	else	// click on any middle one
	{
		m_iPrimitivePalette[index] = TRUE;
		for (i = IgiRectangle; i < index; i++)
		{
			m_iPrimitivePalette[i] = FALSE;
		}

		for (i = index + 1; i < Indicator + 1; i++)
		{
			m_iPrimitivePalette[i] = FALSE;
		}
	}
}

////////////////////////////////////////////////////////////!///////////////////
//
//	This function sets all primitive buttons to FALSE.
//	It is called when the user clicks right mouse button on the edit area to
//	finish adding primitive object action.
//
////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::UnSetPaletteBar()
{
	int i;

	for (i = 0; i < Indicator + 1; i++)
	{
		m_iPrimitivePalette[i] = FALSE;
	}
```

```
}

////////////////////////////////////////////////////////////////////////////
//
//      This function gets the current color that user clicks on the color bar
//
////////////////////////////////////////////////////////////////////////////
COLORREF CIgieditorDoc::GetCurrentColor()
{
        return m_cCurrentColor;
}

////////////////////////////////////////////////////////////////////////////
//
//      This function gets the current edit mode. It is initialized to move mode
//      and changed to add primitive object mode after user clicks a primitive on the
//      primitive bar.
//
////////////////////////////////////////////////////////////////////////////
EditModeIndex CIgieditorDoc::GetEditMode()
{
        return m_iEditMode;
}

////////////////////////////////////////////////////////////////////////////
//
//      This function gets the type of primitive object that will be added to the
//      edit area. It depends on the which primitive button is clicked on the
//      primitive bar.
//
////////////////////////////////////////////////////////////////////////////
PalettePrimitive CIgieditorDoc::GetCurrentPrimitive()
{
        return m_iCurrentPrimitive;
}

////////////////////////////////////////////////////////////////////////////
//
//      This function gets the fill mode for adding a new primitive object.
//      It is TRUE for FillRectangle, FillEllipse and FillTriangle and it is FALSE
//      for all other kind of primitive.
//
////////////////////////////////////////////////////////////////////////////
```

```
BOOL CIgieditorDoc::GetFillMode()
{
	return m_bFillMode;
}

////////////////////////////////////////////////////////////////////////////
//
//	This function gets the select order for adding triangle object. It returns
//	1 when user selects first point and 2 for second point and 3 for the third
//	point of the triangle.
//
////////////////////////////////////////////////////////////////////////////
int CIgieditorDoc::GetSelectOrder()
{
	return m_iSelectOrder;
}

////////////////////////////////////////////////////////////////////////////
//
//	This function sets the select order for adding triangle object. It sets
//	the order to 1 when user selects first point and 2 for second point and 3
//	for the third point of the triangle.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::SetSelectOrder(int selectOrder)
{
	m_iSelectOrder = selectOrder;
}

////////////////////////////////////////////////////////////////////////////
//
//	This function creates a new primitive according to the selected primitve
//	and color button. The new primitive object is added to the tail of the
//	primitive object list. It is called by IgieditorView class after user
//	selected the desired position on the edit area.
//
////////////////////////////////////////////////////////////////////////////
CIGIprimitive* CIgieditorDoc::NewIgiPrimitive(PalettePrimitive item, int x[6])
{
	CIGIprimitive* pIgiPrimitive = new CIGIprimitive();	// create a new node of
primitive list
	pIgiPrimitive->SetPrimitiveType(item);		// set parameters of the new primitive
	pIgiPrimitive->SetParameter(x);
	pIgiPrimitive->SetPenColor(m_cCurrentColor);
```

```
        pIgiPrimitive->SetFillMode(m_bFillMode);
        pIgiPrimitive->SetIGItextinput(m_cTextInput);   // only for text primitive

        pIgiPrimitive->NewPrimitive();          // create a new primitive, such as line,
text, etc.

        m_primitiveList.AddTail(pIgiPrimitive);  // add the node to the tail of the list
        SetModifiedFlag();  // Mark the document as having been modified, for
                                            // purposes of confirming File Close.
        return pIgiPrimitive;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function deletes all the primitive object node from the list. It is
//      called when user does "Cut All" action. It sets the IGI document to empty.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::DeleteContents()
{
        while (!m_primitiveList.IsEmpty()) // if the primitive list is not empty
        {
                delete m_primitiveList.RemoveHead();     // delete the head of the list
        }
        m_iCountSelect = 0;         // there is no primitive selected
        m_iTagNameFlag = 0;         // user can add tag name again

        CDocument::DeleteContents();
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function returns the pointer of a node in the primitive list that
//      contains the selected primitive when user clicks the left mouse button over
//      a primitive. Otherwise, it returns NULL.
//      If the mouse is clicked over a primitive, the function sets the primitive
//      selected and updates the number of selected primitives on the list.
//
//////////////////////////////////////////////////////////////////////////////
CIGIprimitive* CIgieditorDoc::SelectIGIprimitive(CPoint point)
{
        POSITION pos = m_primitiveList.GetTailPosition();
        while (pos != NULL)
        {
                // get the node pointer of the list
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetPrev(pos);
                // if the mouse is clicked over a primitive
                if (pIGIprimitive->SelectIGIprimitive(point))
                {
                        // set the primitive "ObjectSelect" to TRUE
                        pIGIprimitive->SetIGIObjectSelect(TRUE);
                        // update the selected primitive count from the list
                        m_iCountSelect = GetCountSelect();
                        return pIGIprimitive;
                }
        }
        return NULL;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function will draw each primitive object on the edit area. It will
//      be called by command "OnDraw" in class IgieditorView.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::DrawIGIprimitiveList(CDC *pDC)
{
        POSITION pos = m_primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                // get pointer of each IGIprimitive node in the list
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
                // call each primitive class to draw the primitive and select mark
                pIGIprimitive->DrawIGIprimitive(pDC);
        }
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function sets all primitive objects to select or unselect.
//      If select is TRUE, it assigns select count as the node number of the list.
//      Otherwise, it sets the select count to 0.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::SetIGIObjectSelectList(BOOL select)
{
        POSITION pos = m_primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
                pIGIprimitive->SetIGIObjectSelect(select);
        }
        if (select)
                m_iCountSelect = m_primitiveList.GetCount();
        else
                m_iCountSelect = 0;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function adds the moving distance (dx, dy) to each primitive object.
//      It copies the current primitive list to previous list for the purpose of
//      undo before it updates the each primitive object's position. After update,
//      it sets document-modified flag to save document when the program exist or
//      open another file.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::MoveIGIPrimitiveList(int dx, int dy)
{
        CreatePreviousList();

        POSITION pos = m_primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
                if (pIGIprimitive->GetIGIObjectSelect())
                {
                        pIGIprimitive->MoveIGIPrimitive(dx, dy);
                }
        }
        SetModifiedFlag();  // Mark the document as having been modified, for
                                          // purposes of confirming File Close.
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function sets the edit mode. It is called when user has finished
//      adding new primitive object and changed the edit mode from adding mode
//      to moving mode.
//
```

```
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::SetEditMode(EditModeIndex editMode)
{
	m_iEditMode = editMode;
}

/////////////////////////////////////////////////////////////////////////////
//
//	This function updates each selected primitive object's color when user
//	clicks on the color button.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::UpdateColor()
{
	POSITION pos = m_primitiveList.GetHeadPosition();
	while (pos != NULL)
	{
		CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
		if (pIGIprimitive->GetIGIObjectSelect())
		{
			pIGIprimitive->UpdateColor(m_cCurrentColor);
		}
	}
	SetModifiedFlag();  // Mark the document as having been modified, for
	                                    // purposes of confirming File Close.
}

/////////////////////////////////////////////////////////////////////////////
//
//	This function copies the primitive object list to another one as a backup
//	list. The purpose of creating a backup list is for "Undo". When user wants
//	to undo the last action, it only needs to copy the backup list to the
//	primitive object list. This function is called when user modifies the
//	primitive object list each time.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::CreatePreviousList()
{
	// delete the previous list first
	while (!m_primitivePrevList.IsEmpty())
	{
		delete m_primitivePrevList.RemoveHead();
	}
```

```
    POSITION pos = m_primitiveList.GetHeadPosition();
    while (pos != NULL)
    {
        CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
        // create a new IGIprimitve node
        CIGIprimitive* pIgiPrimitive = CopyIGIprimitive(pIGIprimitive, 0);
        m_primitivePrevList.AddTail(pIgiPrimitive);
    }
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function switches the previous object list and current primitive object
//      list. It is called when user runs action "Undo".
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::SwapList()
{
    CTypedPtrList<CObList,CIGIprimitive*>      m_primitiveTempList;

    // copy the primitive list to a temporary list first
    POSITION pos = m_primitiveList.GetHeadPosition();
    while (pos != NULL)
    {
        CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
        CIGIprimitive* pIgiPrimitive = CopyIGIprimitive(pIGIprimitive, 0);
        m_primitiveTempList.AddTail(pIgiPrimitive);
    }

    // empty the current primitive list
    while (!m_primitiveList.IsEmpty())
    {
        delete m_primitiveList.RemoveHead();
    }

    // copy previous list to current primitive list
    pos = m_primitivePrevList.GetHeadPosition();
    while (pos != NULL)
    {
        CIGIprimitive* pIGIprimitive = m_primitivePrevList.GetNext(pos);
        CIGIprimitive* pIgiPrimitive = CopyIGIprimitive(pIGIprimitive, 0);
        m_primitiveList.AddTail(pIgiPrimitive);
        }

        // empty the previous primitive list
        while (!m_primitivePrevList.IsEmpty())
        {
                delete m_primitivePrevList.RemoveHead();
        }
        // copy the temporary primitive list to previous primitive list
        pos = m_primitiveTempList.GetHeadPosition();
        while (pos != NULL)
        {
                CIGIprimitive* pIGIprimitive = m_primitiveTempList.GetNext(pos);
                CIGIprimitive* pIgiPrimitive = CopyIGIprimitive(pIGIprimitive, 0);
                m_primitivePrevList.AddTail(pIgiPrimitive);
        }
        // empty the temporary primitive list to prevent memory leak
        while (!m_primitiveTempList.IsEmpty())
        {
                delete m_primitiveTempList.RemoveHead();
        }
        // get the number of selected primitive
        m_iCountSelect = GetCountSelect();
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function updates the primitive object which is modified by user
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::ModifyIGIPrimitive(CIGIprimitive* p, int dx, int dy, int index)
{
        CreatePreviousList();
        p->ModifyPrimitive(dx, dy, index);
        SetModifiedFlag();
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function makes a copy for an IGIprimitive that is a node of the
//      primitive list. It is called in two cases. One is called by creating
//      previous list. In this case, the parameter dx = 0 and the ObjectSelect is
//      keeped same as original IGIprimitive object. Another case it is called
//      is that user runs the action "Copy". At this time, dx is not equal 0 and that
//     add a small distance to new copied primitive object. For new primitive
//     object, the ObjectSelect is set to FALSE.
//
///////////////////////////////////////////////////////////////////////////////
CIGIprimitive* CIgieditorDoc::CopyIGIprimitive(CIGIprimitive *pIGIprimitive, int dx)
{
       PalettePrimitive item;
       int x[6];
       CString ctextInput;
       int iNoStream;

       // get original IGIprimitive node value
       item = pIGIprimitive->GetPrimitiveItem();
       pIGIprimitive->GetParameter(x, dx);
       COLORREF  currentColor = pIGIprimitive->GetPenColor();
       BOOL bFillMode = pIGIprimitive->GetFillMode();
       BOOL select = pIGIprimitive->GetIGIObjectSelect();

       // if primitive is text, get text string
       if (item == IGI_PRIM_TEXT)
       {
              ctextInput = pIGIprimitive->GetIGItextinput();
       }

       // if copy a stream, need to get new stream number
       if (pIGIprimitive->GetPrimitiveType() == IGI_STREAM && dx != 0)
       {
              int iType = pIGIprimitive->GetStreamType(); // get stream type
              int iDir = pIGIprimitive->GetStreamDir(); // get stream direction
              iNoStream = GetNewStreamNumber(iType, iDir);     // get new stream number
       }

       // create a new IGIprimitive node
       CIGIprimitive* pIgiPrimitive = new CIGIprimitive();
       // copy the value to new IGIprimitive node
       pIgiPrimitive->SetPrimitiveType(item);
       pIgiPrimitive->SetParameter(x);
       pIgiPrimitive->SetPenColor(currentColor);
       pIgiPrimitive->SetFillMode(bFillMode);
       pIgiPrimitive->SetIGItextinput(ctextInput);

       // create a new primitive object
       pIgiPrimitive->NewPrimitive();
```

```
        // if copy stream, set new stream number to the new stream object
        if (pIGIprimitive->GetPrimitiveType() == IGI_STREAM && dx != 0)
        {
                pIgiPrimitive->SetStreamNumber(iNoStream);
        }

        // if copy action
        if (dx != 0)
        {
                pIGIprimitive->SetIGIObjectSelect(FALSE);
                pIgiPrimitive->SetIGIObjectSelect(FALSE);
        }
        else    // if copy to previous list
        {
                pIgiPrimitive->SetIGIObjectSelect(select);
        }

        return pIgiPrimitive;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function returns the total selected primitive object number.
//      The purpose of this number is to check the button of the main tool bar.
//      For example, if there is no object is selected, the "Copy" button is
//      checked and that means the action is disable at this time.
//
//////////////////////////////////////////////////////////////////////////////
int CIgieditorDoc::GetCountSelect()
{
        int countSelect = 0;
        POSITION pos = m_primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
                if (pIGIprimitive->GetIGIObjectSelect())
                {
                        ++countSelect;
                }
        }
        return countSelect;
}
```

```
/////////////////////////////////////////////////////////////////////////////////////
//
//     This function returns the number of tag name in the IGI document.
//     It only has one tag name in each IGI document. User is not allowed
//     to add second tag name object in the primitive list. The flag 0
//     that means there is no tag name in the list yet.
//
/////////////////////////////////////////////////////////////////////////////////////
int CIgieditorDoc::GetTagNameFlag()
{
       return m_iTagNameFlag;
}

/////////////////////////////////////////////////////////////////////////////////////
//
//     This function sets the tag name flag to 1 when user add tag name object to
//     the primitive object list. The flag is set to 0 when the tag name object
//     has been deleted.
//
/////////////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::SetTagNameFlag(int tagNameFlag)
{
       m_iTagNameFlag = tagNameFlag;
}

/////////////////////////////////////////////////////////////////////////////////////
//
//     This function returns the input text string when user adds a new text.
//
/////////////////////////////////////////////////////////////////////////////////////
CString CIgieditorDoc::GetTextInput()
{
       return m_cTextInput;
}

/////////////////////////////////////////////////////////////////////////////////////
//
//     This function gets a new stream number according to its type and direction.
//     It is called when user adds or copies a stream.
//
/////////////////////////////////////////////////////////////////////////////////////
int CIgieditorDoc::GetNewStreamNumber(int iType, int iDir)
```

```
{
        int i;
        int value_tmp[100];

        // initialize array
        for (i = 0; i < 100; i++)
        {
       value_tmp[i] = -1;
        }

        POSITION pos = m_primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);

                if (pIGIprimitive->GetPrimitiveType() == IGI_STREAM)
                {
                        if (pIGIprimitive->IsLookForStream(iType, iDir))
                        {
                                i = pIGIprimitive->GetStreamNumber(iType, iDir);
                                value_tmp[i] = i;
                        }
                }
        }
        for (i = 0; i < 100; i++)   // assume maximun 100 stream
        {
                // the new stream number is the first available index for that position
                // the array value is not zero
                if (value_tmp[i] == -1)
                {
                        return (i);
                }
        }
        return 0;
}

///////////////////////////////////////////////////////////////////////////////
//
//      This function changes the stream orientation value when user rotates a
//      stream.
//
///////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::IncreaseOrientation(CIGIprimitive *p)
```

```
{
        CreatePreviousList();
        p->IncreaseOrientation();
}

////////////////////////////////////////////////////////////////////////////
//
//      This function gets the pointer of primitive stream object in the list
//      that has been selected. It is called when user clicks right mouse button
//      to rotate a stream orientation.
//
////////////////////////////////////////////////////////////////////////////
CIGIprimitive* CIgieditorDoc::GetSelectIGIstream()
{
        POSITION pos = m_primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                CIGIprimitive* pIGIprimitive = m_primitiveList.GetNext(pos);
                if (pIGIprimitive->GetIGIObjectSelect() &&
                        pIGIprimitive->GetPrimitiveType() == IGI_STREAM)
                {
                        return pIGIprimitive;
                }
        }
        return NULL;
}

////////////////////////////////////////////////////////////////////////////
//
//      This function inverts the primitive object display on the edit area during
//      the object is moving.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::InvertIgiPrimitive(CDC *pDC, CIGIprimitive *p, int dx, int dy)
{
        p->InvertIgiPrimitive(pDC, dx, dy);
}


////////////////////////////////////////////////////////////////////////////
//
//      This function calls "OnDeleteSome" command to delete the selected primitive
//      objects. This function is called when user presses the "Delete" key to cut
//      the selected primitive objects.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorDoc::OnKeyDelete()
{
        OnDeleteSome();
}
```

4.2.4 igieditorView.cpp

```
// igieditorView.cpp : implementation of the CIgieditorView class
//

#include "stdafx.h"
#include "igieditor.h"

#include "igieditorDoc.h"
#include "igieditorView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define StreamLength 18
#define StreamArrow         4

extern CStatusBar  m_wndStatusBar;
/////////////////////////////////////////////////////////////////////////////
// CIgieditorView

IMPLEMENT_DYNCREATE(CIgieditorView, CView)

BEGIN_MESSAGE_MAP(CIgieditorView, CView)
        //{{AFX_MSG_MAP(CIgieditorView)
        ON_WM_CAPTURECHANGED()
        ON_WM_CONTEXTMENU()
        ON_WM_CHAR()
        ON_WM_KEYDOWN()
        ON_WM_KEYUP()
        ON_WM_LBUTTONDBLCLK()
        ON_WM_LBUTTONDOWN()
        ON_WM_RBUTTONUP ()
        ON_WM_MOVE()
        ON_WM_SHOWWINDOW()
        ON_WM_LBUTTONUP()
        ON_WM_MOUSEMOVE()
        ON_WM_ERASEBKGND()
        ON_WM_CANCELMODE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIgieditorView construction/destruction

CIgieditorView::CIgieditorView()
{
        // TODO: add construction code here
        m_bTracking = FALSE;
        m_bCaptureEnabled = TRUE;
        m_pCurrent = NULL;
        m_ptFrom = 0;
        m_ptTo = 0;
}

CIgieditorView::~CIgieditorView()
{
}

BOOL CIgieditorView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
//      CIgieditorView drawing
//      This function delegates the drawing of individual primitive to
//      CIGIPrimitve::DrawIGIPrimitive(). It is called when the view window
//      updated.

void CIgieditorView::OnDraw(CDC* pDC)
{
        CIgieditorDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        // TODO: add draw code for native data here

        // The view delegates the drawing of individual primitive to
        // CIGIPrimitve::DrawIGIPrimitive().
        pDoc->DrawIGIprimitiveList(pDC);
}

/////////////////////////////////////////////////////////////////////////////
// CIgieditorView diagnostics

#ifdef _DEBUG
void CIgieditorView::AssertValid() const
{
        CView::AssertValid();
}

void CIgieditorView::Dump(CDumpContext& dc) const
{
        CView::Dump(dc);
}

CIgieditorDoc* CIgieditorView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CIgieditorDoc)));
        return (CIgieditorDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
//      This function is CIgieditorView message handlers
//      Nonzero if the message was fully processed in PreTranslateMessage and should
//      not be processed further. Zero if the message should be processed in the
//      normal way.
/////////////////////////////////////////////////////////////////////////////

int CIgieditorView::PreTranslateMessage(MSG* pMsg)
{
        CIgieditorDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // get primitive object parameter from document class
```

```
    m_iEditMode = pDoc->GetEditMode();
    m_iItem = pDoc->GetCurrentPrimitive();
    m_cCurrentColor = pDoc->GetCurrentColor();
    m_bFill = pDoc->GetFillMode();
    m_iSelectOrder = pDoc->GetSelectOrder();

    // release right mouse button on the edit area
    if (pMsg->message == WM_RBUTTONUP)
    {
        CWnd* pWnd = CWnd::FromHandlePermanent(pMsg->hwnd);
        CControlBar* pBar = DYNAMIC_DOWNCAST(CControlBar, pWnd);
        if (pBar == NULL)    // it is not clicked on the bar
        {
            m_iEditMode = MOVE_MODE;    // set to move mode

            pDoc->SetEditMode(m_iEditMode);    // send the mode status to doc
            pDoc->UnSetPaletteBar();    // unselect all button on the palette bar
            m_wndStatusBar.SetPaneText( 0, "Ready" , TRUE );// set message on the
status bar
        }
    }
    else if (pMsg->message == WM_LBUTTONDOWN && m_iEditMode == ADD_PRIMITIVE_MODE)
    {
        // in adding mode and click left button
        CWnd* pWnd = CWnd::FromHandlePermanent(pMsg->hwnd);
        CControlBar* pBar = DYNAMIC_DOWNCAST(CControlBar, pWnd);
        if (pBar == NULL && !m_bTracking)
        {
            // get mouse position
            m_point.x = LOWORD(pMsg->lParam);
            m_point.y = HIWORD(pMsg->lParam);

            CClientDC dc (this);
            dc.DPtoLP (&m_point);
            m_ptFrom = m_point;
            m_ptTo = m_point;
            m_bTracking = TRUE;

            if (m_bCaptureEnabled)
                SetCapture ();
        }
    }
    else if (pMsg->message == WM_LBUTTONUP && m_iEditMode == ADD_PRIMITIVE_MODE)
    {
        // in adding mode and release left button, add new primitive object to
        // the primitive list.
        CWnd* pWnd = CWnd::FromHandlePermanent(pMsg->hwnd);
        CControlBar* pBar = DYNAMIC_DOWNCAST(CControlBar, pWnd);
        if (pBar == NULL)
        {
            m_point.x = LOWORD(pMsg->lParam);
            m_point.y = HIWORD(pMsg->lParam);

            if (m_bTracking)
            {
                m_bTracking = FALSE;
                if (GetCapture () == this)
                    ReleaseCapture ();

                CClientDC dc (this);
                dc.DPtoLP (&m_point);
                COLORREF cBkColor = dc.GetBkColor();

                CPen pen (PS_SOLID, 1, m_cCurrentColor);
                dc.SelectObject (&pen);
                CBrush pBrush (cBkColor);
                dc.SelectObject(&pBrush);

                int x[6];
                switch (m_iItem)
                {
                case IGI_PRIM_BOX:
                    InvertBox (&dc, m_ptFrom, m_ptTo);
                    if (m_bFill)
                    {
                        CBrush pBrush (m_cCurrentColor);
                        dc.SelectObject(&pBrush);

                        dc.Rectangle (m_ptFrom.x, m_ptFrom.y, m_point.x,
m_point.y);
                    }
                    else
                    {
                        dc.Rectangle (m_ptFrom.x, m_ptFrom.y, m_point.x,
m_point.y);
                    }
                    // add box to object list (data base)
                    x[0] = min(m_ptFrom.x, m_point.x);
```

```
                                        x[1] = min(m_ptFrom.y, m_point.y);
                                        x[2] = max(m_ptFrom.x, m_point.x);
                                        x[3] = max(m_ptFrom.y, m_point.y);
                                        x[4] = 0;
                                        x[5] = 0;

                                        pDoc->NewIgiPrimitive(m_iItem, x);
                                        m_wndStatusBar.SetPaneText( 0, "left button for another
box, right button to end add box" , TRUE );
                                        break;

                                case IGI_PRIM_ELLIPSE:
                                        InvertEllipse (&dc, m_ptFrom, m_ptTo);
                                        if (m_bFill)
                                        {
                                                CBrush pBrush (m_cCurrentColor);
                                                dc.SelectObject(&pBrush);
                                                dc.Ellipse(m_ptFrom.x, m_ptFrom.y, m_point.x,
m_point.y);
                                        }
                                        else
                                        {
                                                dc.Ellipse(m_ptFrom.x, m_ptFrom.y, m_point.x,
m_point.y);
                                        }

                                        x[0] = min(m_ptFrom.x, m_point.x);
                                        x[1] = min(m_ptFrom.y, m_point.y);
                                        x[2] = max(m_ptFrom.x, m_point.x);
                                        x[3] = max(m_ptFrom.y, m_point.y);
                                        x[4] = 0;
                                        x[5] = 0;

                                        pDoc->NewIgiPrimitive(m_iItem, x);
                                        m_wndStatusBar.SetPaneText( 0, "left button for another
ellipse, right button to end add ellipse" , TRUE );
                                        break;

                                case IGI_PRIM_LINE:
                                        InvertLine (&dc, m_ptFrom, m_ptTo);
                                        dc.MoveTo (m_ptFrom);
                                        dc.LineTo (m_ptTo);

                                        x[0] = m_ptFrom.x;
                                        x[1] = m_ptFrom.y;
```

```
                                   x[2] = m_ptTo.x;
                                   x[3] = m_ptTo.y;
                                   x[4] = 0;
                                   x[5] = 0;

                                   pDoc->NewIgiPrimitive(m_iItem, x);
                                   m_wndStatusBar.SetPaneText( 0, "left button for another
line, right button to end add line" , TRUE );
                                   break;



                            case IGI_PRIM_TRIANGLE:
                                   if (m_iSelectOrder == 1)
                                   {
                                          m_ptPoly[0] = m_point;      // get first point
position
                                          m_ptTo = m_point;
                                          m_bTracking = TRUE;
                                          m_iSelectOrder = 2;
                                          pDoc->SetSelectOrder(m_iSelectOrder);
                                   }
                                   else if (m_iSelectOrder == 2)
                                   {
                                          m_ptPoly[1] = m_point;      // get second point
position
                                          InvertLine (&dc, m_ptPoly[0], m_ptPoly[1]);
                                          m_bTracking = TRUE;
                                          m_iSelectOrder = 3;
                                          pDoc->SetSelectOrder(m_iSelectOrder);
                                   }
                                   else if (m_iSelectOrder == 3)
                                   {
                                          m_ptPoly[2] = m_point;      // get third point
position
                                          m_bTracking = FALSE;
                                          m_iSelectOrder = 1;
                                          pDoc->SetSelectOrder(m_iSelectOrder);
                                          InvertLine (&dc, m_ptPoly[0], m_ptPoly[2]);
                                          InvertLine (&dc, m_ptPoly[1], m_ptPoly[2]);
                                          if (m_bFill)
                                          {
                                                 CBrush pBrush (m_cCurrentColor);
                                                 dc.SelectObject(&pBrush);

                                                 dc.Polygon(m_ptPoly, 3);
                                          }
                                          else
                                          {
                                                 dc.Polygon(m_ptPoly, 3);
                                          }

                                          x[0] = m_ptPoly[0].x;
                                          x[1] = m_ptPoly[0].y;
                                          x[2] = m_ptPoly[1].x;
                                          x[3] = m_ptPoly[1].y;
                                          x[4] = m_ptPoly[2].x;
                                          x[5] = m_ptPoly[2].y;

                                          pDoc->NewIgiPrimitive(m_iItem, x);
                                          m_wndStatusBar.SetPaneText( 0, "left button for
another triangle, right button to end add triangle" , TRUE );

                                   }
                                   break;

                            case IGI_PRIM_TAG:

                                   if ( pDoc->GetTagNameFlag() == 0 ) // there is no tag yet
                                   {
                                          pDoc->SetTagNameFlag(1);    // set flag to 1
                                          CFont m_fontMain;
                                          int nHeight = -((dc.GetDeviceCaps (LOGPIXELSY) *8) /
78);

                                          m_fontMain.CreateFont(nHeight, 0, 0, 0, FW_NORMAL,
0, 0, 0,
                                                 DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
CLIP_CHARACTER_PRECIS,
                                                 DEFAULT_QUALITY, DEFAULT_PITCH |
FF_DONTCARE,"Arial");

                                          dc.SelectObject( &m_fontMain );

                                          CString cString = "xxxxxxxxxxxxxx";

                                          SIZE size;
                                          GetTextExtentPoint32( dc, (LPCTSTR)cString,
cString.GetLength(), &size );

                                          dc.SetTextColor(m_cCurrentColor);
                                          dc.SetBkColor(cBkColor);
```

```
                        dc.TextOut ( m_point.x, m_point.y, cString );

                        x[0] = m_point.x;
                        x[1] = m_point.y;
                        x[2] = size.cx;
                        x[3] = size.cy;
                        x[4] = 0;
                        x[5] = 0;

                        pDoc->NewIgiPrimitive(m_iItem, x);
                        m_wndStatusBar.SetPaneText( 0, "click right button
to end add tag name" , TRUE );
                    }
                    else    // pop up message
                    {
                        MessageBox( "Only add one tag name !",
                        "SIMSMART IGI Editor Message",
                        MB_ICONINFORMATION | MB_OK );
                        pDoc->SetEditMode(MOVE_MODE);
                        pDoc->UnSetPaletteBar();
                    }
                    break;

                case IGI_PRIM_TEXT:
                    {
                        CFont m_fontMain;
                        int nHeight = -((dc.GetDeviceCaps (LOGPIXELSY) *8) /
78);

                        m_fontMain.CreateFont(nHeight, 0, 0, 0, FW_NORMAL,
0, 0, 0,
                            DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
CLIP_CHARACTER_PRECIS,
                            DEFAULT_QUALITY, DEFAULT_PITCH |
FF_DONTCARE,"Arial");

                        dc.SelectObject( &m_fontMain );

                        m_cTextInput = pDoc->GetTextInput();

                        SIZE size;
                        GetTextExtentPoint32( dc, (LPCTSTR)m_cTextInput,
m_cTextInput.GetLength(), &size );

                        dc.SetTextColor(m_cCurrentColor);
                        dc.SetBkColor(cBkColor);
```

```
                                        dc.TextOut ( m_point.x, m_point.y, m_cTextInput );

                                        x[0] = m_point.x;
                                        x[1] = m_point.y;
                                        x[2] = size.cx;
                                        x[3] = size.cy;
                                        x[4] = 0;
                                        x[5] = 0;

                                        pDoc->NewIgiPrimitive(m_iItem, x);
                                        m_wndStatusBar.SetPaneText( 0, "left button for
another text, right button to end add text" , TRUE );
                                }
                                break;

                        case IGI_PRIM_INPUT_PROCESS_STREAM:
                                {

                                        CPen pen (PS_SOLID, 1, RGB (255, 0, 0));
                                        dc.SelectObject (&pen);
                                        dc.SetBkColor(cBkColor);

                                        dc.MoveTo(m_point);
                                        dc.LineTo(m_point.x+StreamLength, m_point.y);
                                        dc.MoveTo(m_point);
                                        dc.LineTo(m_point.x+StreamArrow,
m_point.y+StreamArrow);
                                        dc.MoveTo(m_point);
                                        dc.LineTo(m_point.x+StreamArrow, m_point.y-
StreamArrow);


                                        m_iStreamType = 2;
                                        m_iStreamDir = 0;
                                        m_iStreamOrientation = 0;
                                        m_iStreamNumber = pDoc-
>GetNewStreamNumber(m_iStreamType, m_iStreamDir);

                                        x[0] = m_point.x;
                                        x[1] = m_point.y;
                                        x[2] = m_iStreamType;
                                        x[3] = m_iStreamDir;
                                        x[4] = m_iStreamOrientation;
                                        x[5] = m_iStreamNumber;
```

```
                                          pDoc->NewIgiPrimitive(m_iItem, x);
                                          m_wndStatusBar.SetPaneText( 0, "left button for
another one, right button to end add stream" , TRUE );
                                          InvalidateRect(NULL, TRUE);
                                   }
                                   break;

                            case IGI_PRIM_OUTPUT_PROCESS_STREAM:
                                   {
                                          CPen pen (PS_SOLID, 1, RGB (255, 0, 0));
                                          dc.SelectObject (&pen);
                                          dc.SetBkColor(cBkColor);

                                          dc.MoveTo(m_point);
                                          dc.LineTo(m_point.x+StreamLength, m_point.y);
                                          dc.LineTo(m_point.x-StreamArrow+StreamLength,
m_point.y-StreamArrow);
                                          dc.MoveTo(m_point.x+StreamLength, m_point.y);
                                          dc.LineTo(m_point.x-StreamArrow+StreamLength,
m_point.y+StreamArrow);

                                          m_iStreamType = 2;
                                          m_iStreamDir = 1;
                                          m_iStreamOrientation = 0;
                                          m_iStreamNumber = pDoc-
>GetNewStreamNumber(m_iStreamType, m_iStreamDir);

                                          x[0] = m_point.x;
                                          x[1] = m_point.y;
                                          x[2] = m_iStreamType;
                                          x[3] = m_iStreamDir;
                                          x[4] = m_iStreamOrientation;
                                          x[5] = m_iStreamNumber;

                                          pDoc->NewIgiPrimitive(m_iItem, x);
                                          m_wndStatusBar.SetPaneText( 0, "left button for
another one, right button to end add stream" , TRUE );
                                          InvalidateRect(NULL, TRUE);
                                   }
                                   break;

                            case IGI_PRIM_INPUT_LOGIC_STREAM:
                                   {
                                          CPen pen (PS_DOT, 1, RGB (255, 0, 0));
                                          dc.SelectObject (&pen);
                                          dc.SetBkColor(cBkColor);

                                          dc.MoveTo(m_point);
                                          dc.LineTo(m_point.x+StreamLength, m_point.y);
                                          dc.MoveTo(m_point);
                                          dc.LineTo(m_point.x+StreamArrow,
m_point.y+StreamArrow);
                                          dc.MoveTo(m_point);
                                          dc.LineTo(m_point.x+StreamArrow, m_point.y-
StreamArrow);

                                          m_iStreamType = 1;
                                          m_iStreamDir = 0;
                                          m_iStreamOrientation = 0;
                                          m_iStreamNumber = pDoc-
>GetNewStreamNumber(m_iStreamType, m_iStreamDir);

                                          x[0] = m_point.x;
                                          x[1] = m_point.y;
                                          x[2] = m_iStreamType;
                                          x[3] = m_iStreamDir;
                                          x[4] = m_iStreamOrientation;
                                          x[5] = m_iStreamNumber;

                                          pDoc->NewIgiPrimitive(m_iItem, x);
                                          m_wndStatusBar.SetPaneText( 0, "left button for
another one, right button to end add stream" , TRUE );
                                          InvalidateRect(NULL, TRUE);
                                   }
                                   break;


                            case IGI_PRIM_OUTPUT_LOGIC_STREAM:
                                   {
                                          CPen pen (PS_DOT, 1, RGB (255, 0, 0));
                                          dc.SelectObject (&pen);
                                          dc.SetBkColor(cBkColor);

                                          dc.MoveTo(m_point);
                                          dc.LineTo(m_point.x+StreamLength, m_point.y);
                                          dc.MoveTo(m_point.x+StreamLength, m_point.y);
                                          dc.LineTo(m_point.x-StreamArrow+StreamLength,
m_point.y-StreamArrow);
                                          dc.MoveTo(m_point.x+StreamLength, m_point.y);
                                          dc.LineTo(m_point.x-StreamArrow+StreamLength,
m_point.y+StreamArrow);
```

```
                        m_iStreamType = 1;
                        m_iStreamDir = 1;
                        m_iStreamOrientation = 0;
                        m_iStreamNumber = pDoc-
>GetNewStreamNumber(m_iStreamType, m_iStreamDir);

                        x[0] = m_point.x;
                        x[1] = m_point.y;
                        x[2] = m_iStreamType;
                        x[3] = m_iStreamDir;
                        x[4] = m_iStreamOrientation;
                        x[5] = m_iStreamNumber;

                        pDoc->NewIgiPrimitive(m_iItem, x);
                        m_wndStatusBar.SetPaneText( 0, "left button for
another one, right button to end add stream" , TRUE );
                        InvalidateRect(NULL, TRUE);
                    }
                    break;

                case IGI_PRIM_INPUT_ANALOG_STREAM:
                    {
                        CPen pen (PS_DASHDOT, 1, RGB (255, 0, 0));
                        dc.SelectObject (&pen);
                        dc.SetBkColor(cBkColor);

                        dc.MoveTo(m_point);
                        dc.LineTo(m_point.x+StreamLength, m_point.y);
                        dc.MoveTo(m_point);
                        dc.LineTo(m_point.x+StreamArrow,
m_point.y+StreamArrow);
                        dc.MoveTo(m_point);
                        dc.LineTo(m_point.x+StreamArrow, m_point.y-
StreamArrow);

                        m_iStreamType = 0;
                        m_iStreamDir = 0;
                        m_iStreamOrientation = 0;
                        m_iStreamNumber = pDoc-
>GetNewStreamNumber(m_iStreamType, m_iStreamDir);

                        x[0] = m_point.x;
                        x[1] = m_point.y;
                        x[2] = m_iStreamType;
                        x[3] = m_iStreamDir;
                        x[4] = m_iStreamOrientation;
                        x[5] = m_iStreamNumber;

                        pDoc->NewIgiPrimitive(m_iItem, x);

                        m_wndStatusBar.SetPaneText( 0, "left button for
another one, right button to end add stream" , TRUE );

                        InvalidateRect(NULL, TRUE);

                    }
                    break;

                case IGI_PRIM_OUTPUT_ANALOG_STREAM:
                    {
                        CPen pen (PS_DASHDOT, 1, RGB (255, 0, 0));
                        dc.SelectObject (&pen);
                        dc.SetBkColor(cBkColor);

                        dc.MoveTo(m_point);
                        dc.LineTo(m_point.x+StreamLength, m_point.y);
                        dc.MoveTo(m_point.x+StreamLength, m_point.y);
                        dc.LineTo(m_point.x-StreamArrow+StreamLength,
m_point.y-StreamArrow);
                        dc.MoveTo(m_point.x+StreamLength, m_point.y);
                        dc.LineTo(m_point.x-StreamArrow+StreamLength,
m_point.y+StreamArrow);

                        m_iStreamType = 0;
                        m_iStreamDir = 1;
                        m_iStreamOrientation = 0;
                        m_iStreamNumber = pDoc-
>GetNewStreamNumber(m_iStreamType, m_iStreamDir);

                        x[0] = m_point.x;
                        x[1] = m_point.y;
                        x[2] = m_iStreamType;
                        x[3] = m_iStreamDir;
                        x[4] = m_iStreamOrientation;
                        x[5] = m_iStreamNumber;

                        pDoc->NewIgiPrimitive(m_iItem, x);
                        m_wndStatusBar.SetPaneText( 0, "left button for
another one, right button to end add stream" , TRUE );
                        InvalidateRect(NULL, TRUE);
```

```
                                }
                                break;

                        default:
                                break;
                        } // end switch (m_iItem)
                }
        }
}
else if (pMsg->message == WM_MOUSEMOVE && m_iEditMode == ADD_PRIMITIVE_MODE)
{
        CWnd* pWnd = CWnd::FromHandlePermanent(pMsg->hwnd);
        CControlBar* pBar = DYNAMIC_DOWNCAST(CControlBar, pWnd);
        if (pBar == NULL)
        {
                if (m_bTracking)
                {
                        m_point.x = LOWORD(pMsg->lParam);
                        m_point.y = HIWORD(pMsg->lParam);
                        CClientDC dc (this);
                        dc.DPtoLP (&m_point);

                        switch (m_iItem)
                        {
                        case IGI_PRIM_BOX:
                                InvertBox (&dc, m_ptFrom, m_ptTo);
                                InvertBox (&dc, m_ptFrom, m_point);

                                m_ptTo = m_point;
                                break;

                        case IGI_PRIM_ELLIPSE:
                                InvertEllipse (&dc, m_ptFrom, m_ptTo);

                                InvertEllipse (&dc, m_ptFrom, m_point);

                                m_ptTo = m_point;
                                break;

                        case IGI_PRIM_LINE:
                                InvertLine (&dc, m_ptFrom, m_ptTo);
                        InvertLine (&dc, m_ptFrom, m_point);
                                m_ptTo = m_point;
                                break;
                case IGI_PRIM_TRIANGLE:
                    if (m_iSelectOrder == 2)
                    {
                        InvertLine (&dc, m_ptPoly[0], m_ptTo);
                    InvertLine (&dc, m_ptPoly[0], m_point);
                        m_ptTo = m_point;
                    }
                    else if (m_iSelectOrder == 3)
                    {

                        InvertLine (&dc, m_ptPoly[0], m_ptTo);
                        InvertLine (&dc, m_ptPoly[0], m_point);
                        InvertLine (&dc, m_ptPoly[1], m_ptTo);
                    InvertLine (&dc, m_ptPoly[1], m_point);

                        m_ptTo = m_point;
                    }
                    break;

                default:
                    break;
                } // end switch (m_iItem)
            }
        }
    }
    return CView::PreTranslateMessage(pMsg);
}



void CIgieditorView::OnCaptureChanged(CWnd *pWnd)
{
    // TODO: Add your message handler code here

    CView::OnCaptureChanged(pWnd);
}

void CIgieditorView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    // TODO: Add your message handler code here

}
```

```
void CIgieditorView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        // TODO: Add your message handler code here and/or call default

        CView::OnChar(nChar, nRepCnt, nFlags);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This command moves the selected primitive objects with the direction key:
//      VK_UP, VK_DOWN, VK_LEFT, and VK_RIGHT.
//      This command also deletes the selected primitive objects when user press
//      key "Delete".
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        // TODO: Add your message handler code here and/or call default
        CIgieditorDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        switch (nChar)
        {
        case VK_UP:
                pDoc->MoveIGIPrimitiveList(0, -1);
                break;

        case VK_DOWN:
                pDoc->MoveIGIPrimitiveList(0, 1);
                break;

        case VK_LEFT:
                pDoc->MoveIGIPrimitiveList(-1, 0);
                break;

        case VK_RIGHT:
                pDoc->MoveIGIPrimitiveList(1, 0);
                break;

        case VK_DELETE:
                pDoc->OnKeyDelete();
                break;

        default:
```

```
        break;
    } // end switch (nChar)
    InvalidateRect(NULL, TRUE); // Refrash window background color
}

void CIgieditorView::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default

    CView::OnKeyUp(nChar, nRepCnt, nFlags);
}

void CIgieditorView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default

    CView::OnLButtonDblClk(nFlags, point);
}

///////////////////////////////////////////////////////////////////////////
//
//      This function returns the pointer of a primitive object when the mouse is
//      clicked on. If user modifies the object, the function also returns the mark
//      index to show which point will be modified.
//
///////////////////////////////////////////////////////////////////////////
void CIgieditorView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CClientDC dc (this);

    CIgieditorDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // check mouse is clicked on a object
    if(m_iEditMode == MOVE_MODE &&
        (m_pCurrent = pDoc->SelectIGIprimitive(point)) != NULL)
    {
        m_bInMove = TRUE;
        m_ptTo = point;
        m_ptFrom = point;
        // it is modification?
        m_bSelectMark = m_pCurrent->GetSelectMark();
        // get mark index
        m_iMarkIndex = m_pCurrent->GetMarkIndex();
        // get the point that is not changed during object modification
        // according to which point is selected.
        if (m_bSelectMark)
        {
            m_bInModify = TRUE;
            m_ptModifyFrom = point;
            switch (m_pCurrent->GetPrimitiveType())
            {
            case IGI_BOX:
                m_ptFrom = m_pCurrent->GetModifyPoint(m_iMarkIndex, 0);

                break;

            case IGI_ELLIPSE:
                m_ptFrom = m_pCurrent->GetModifyPoint(m_iMarkIndex, 0);
                break;

            case IGI_LINE:
                m_ptFrom = m_pCurrent->GetModifyPoint(m_iMarkIndex, 0);
                break;

            case IGI_TRIANGLE:
                m_ptPoly[0] = m_pCurrent->GetModifyPoint(m_iMarkIndex, 0);
                m_ptPoly[1] = m_pCurrent->GetModifyPoint(m_iMarkIndex, 1);
                break;

            default:
                break;
            }  // end switch (m_pCurrent->ObjType)
        }      // if (SelectMark)
    }
    CView::OnLButtonDown(nFlags, point);
}

void CIgieditorView::OnMove(int x, int y)
{
    CView::OnMove(x, y);

    // TODO: Add your message handler code here

}
```

```
void CIgieditorView::OnShowWindow(BOOL bShow, UINT nStatus)
{
	CView::OnShowWindow(bShow, nStatus);

	// TODO: Add your message handler code here
}

/////////////////////////////////////////////////////////////////////////////
//
//	This function calls "Modify" or "Move" functions in document class to
//	update each primitive object if user modifies or moves the objects. If mouse
//	is not clicked on a object, the function sets all object to unselected.
//
/////////////////////////////////////////////////////////////////////////////
void CIgieditorView::OnLButtonUp(UINT nFlags, CPoint point)
{
	// TODO: Add your message handler code here and/or call default
	if (m_bInMove && m_pCurrent != NULL)
	{
		CClientDC dc (this);
		dc.DPtoLP (&point);
		if (m_bSelectMark)	// modify object
		{
			int dx = point.x - m_ptModifyFrom.x;
			int dy = point.y - m_ptModifyFrom.y;
			GetDocument()->ModifyIGIPrimitive(m_pCurrent,dx, dy, m_iMarkIndex);
			InvalidateRect(NULL, TRUE); // Refrash window background color
		}
		else	// move object
		{
			int dx = point.x - m_ptFrom.x;
			int dy = point.y - m_ptFrom.y;

			GetDocument()->MoveIGIPrimitiveList(dx, dy);
			InvalidateRect(NULL, TRUE); // Refrash window background color
		}
		m_bInMove = FALSE;
		m_bInModify = FALSE;
	}
	if (m_pCurrent == NULL)		// unselect all objects if mouse is not clicked on the
object
	{
		CClientDC dc (this);
		GetDocument()->SetIGIObjectSelectList(FALSE);
```

```
                InvalidateRect(NULL, TRUE);
                if (m_iEditMode != ADD_PRIMITIVE_MODE)
                        m_wndStatusBar.SetPaneText( 0, "Ready" , TRUE );
        }
        CView::OnLButtonUp(nFlags, point);
}

////////////////////////////////////////////////////////////////////////////
//
//      This function calls "IncreaseOrientation" functions in document class to
//      update the orientation of a stream object when user rotates it.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorView::OnRButtonUp (UINT nFlags, CPoint point)
{
        CClientDC dc (this);
        // rotate stream
        m_pCurrent = GetDocument()->GetSelectIGIstream();
        if (m_pCurrent != NULL)
        {
                GetDocument()->IncreaseOrientation(m_pCurrent);
                InvalidateRect(NULL, TRUE);
                m_wndStatusBar.SetPaneText( 0, "right button to rotate again, left button to end
rotate" , TRUE );
        }
}

////////////////////////////////////////////////////////////////////////////
//
//      This function updates the objects display during user modifies or moves
//      the selected objects. If a stream object has been selected, it could be
//      rotated and proper message is displayed on the status bar.
//
////////////////////////////////////////////////////////////////////////////
void CIgieditorView::OnMouseMove(UINT nFlags, CPoint point)
{
        // TODO: Add your message handler code here and/or call default
        if (m_bInMove && m_pCurrent != NULL)
        {
                if (m_bSelectMark)    // object is modified
                {
                        ModifyPrimitive(point);
                        m_wndStatusBar.SetPaneText( 0, "modify selected point to desired
position" , TRUE );
```

```
                }
                else
                {
                        if ( MovePrimitive(point) )
                        {
                                if (m_pCurrent->GetPrimitiveType() == IGI_STREAM)       // if a
stream object is selected
                                {
                                        m_wndStatusBar.SetPaneText( 0, "drag selected primitive(s)
to desired position, or right button to rotate the stream", TRUE );

                                }
                                else    // only message for moving is displayed
                                {
                                        m_wndStatusBar.SetPaneText( 0, "drag selected primitive(s)
to desired position", TRUE );
                                }
                        }
                }
        }
        CView::OnMouseMove(nFlags, point);
}

BOOL CIgieditorView::OnEraseBkgnd(CDC* pDC)
{
        // TODO: Add your message handler code here and/or call default

        return CView::OnEraseBkgnd(pDC);
}


///////////////////////////////////////////////////////////////////////////////
//
//      This function inverts a primitive object during the object modified.
//
///////////////////////////////////////////////////////////////////////////////
void CIgieditorView::ModifyPrimitive(CPoint point)
{
        CClientDC dc (this);
        dc.DPtoLP (&point);

        if (m_pCurrent == NULL)
                return;
```

```
switch (m_pCurrent->GetPrimitiveType())
{
        case IGI_BOX:
                if (m_bInModify)
                {
                        InvertBox (&dc, m_ptFrom, m_ptModifyFrom);
                        m_bInModify = FALSE;
                }
                InvertBox (&dc, m_ptFrom, m_ptTo);
                InvertBox (&dc, m_ptFrom, point);
                m_ptTo = point;
                break;



        case IGI_ELLIPSE:
                if (m_bInModify)
                {
                        InvertEllipse (&dc, m_ptFrom, m_ptModifyFrom);
                        m_bInModify = FALSE;
                }
                InvertEllipse (&dc, m_ptFrom, m_ptTo);
                InvertEllipse (&dc, m_ptFrom, point);
                m_ptTo = point;
                break;

        case IGI_LINE:
                if (m_bInModify)
                {
                        InvertLine (&dc, m_ptFrom, m_ptModifyFrom);
                        m_bInModify = FALSE;
                }
                InvertLine (&dc, m_ptFrom, m_ptTo);
                InvertLine (&dc, m_ptFrom, point);
                m_ptTo = point;
                break;

        case IGI_TRIANGLE:
                if (m_bInModify)
                {
                        InvertLine (&dc, m_ptPoly[0], m_ptModifyFrom);
                        InvertLine (&dc, m_ptPoly[1], m_ptModifyFrom);
                        m_bInModify = FALSE;
                }
```

```
                        InvertLine (&dc, m_ptPoly[0], m_ptTo);
                        InvertLine (&dc, m_ptPoly[0], point);
                        InvertLine (&dc, m_ptPoly[1], m_ptTo);
                        InvertLine (&dc, m_ptPoly[1], point);
                        m_ptTo = point;
                        break;

                default :
                        break;
        }   // end switch
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function inverts the primitive objects during the objects moved.
//
//////////////////////////////////////////////////////////////////////////////
BOOL CIgieditorView::MovePrimitive(CPoint point)
{
        CClientDC dc (this);
        dc.DPtoLP (&point);
        int dx1, dy1;
        int dx = m_ptTo.x - m_ptFrom.x;
        int dy = m_ptTo.y - m_ptFrom.y;

        dx1 = point.x - m_ptFrom.x;
        dy1 = point.y - m_ptFrom.y;



        CIgieditorDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        // TODO: add draw code for native data here

        CTypedPtrList<CObList,CIGIprimitive*>& primitiveList = pDoc->m_primitiveList;
        POSITION pos = primitiveList.GetHeadPosition();
        while (pos != NULL)
        {
                CIGIprimitive* p = primitiveList.GetNext(pos);
                if (p->GetIGIObjectSelect())
                {
                        pDoc->InvertIgiPrimitive(&dc, p, dx, dy);
                        m_ptTo = point;
                        dx1 = m_ptTo.x - m_ptFrom.x;
```

```
                    dy1 = m_ptTo.y - m_ptFrom.y;
                    pDoc->InvertIgiPrimitive(&dc, p, dx1, dy1);
            }
        } //  end of while
        return TRUE;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function inverts a rectangle object during adding a
//      Rectangle/FillRectangle object.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorView::InvertBox(CDC *pDC, CPoint ptFrom, CPoint ptTo)
{
        int nOldMode = pDC->SetROP2 (R2_NOT);
        pDC->SelectObject(GetStockObject (NULL_BRUSH));
        pDC->Rectangle (ptFrom.x, ptFrom.y, ptTo.x, ptTo.y);
        pDC->SetROP2 (nOldMode);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function inverts a ellipse object during adding a
//      Ellipse/FillEllipse object.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorView::InvertEllipse(CDC *pDC, CPoint ptFrom, CPoint ptTo)
{
        int nOldMode = pDC->SetROP2 (R2_NOT);
        pDC->SelectObject(GetStockObject (NULL_BRUSH));
        pDC->Ellipse (ptFrom.x, ptFrom.y, ptTo.x, ptTo.y);
        pDC->SetROP2 (nOldMode);
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function inverts a line object during adding a line or triangle
//      object.
//
//////////////////////////////////////////////////////////////////////////////
void CIgieditorView::InvertLine(CDC *pDC, CPoint ptFrom, CPoint ptTo)
{
       int nOldMode = pDC->SetROP2 (R2_NOT);
    pDC->MoveTo (ptFrom);
    pDC->LineTo (ptTo);
    pDC->SetROP2 (nOldMode);
}
```

4.2.5 IgiBox.cpp

```
// IgiBox.cpp: implementation of the CIgiBox class.
//
///////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "igieditor.h"
#include "IgiBox.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
//
//     This constructor sets the object not selected when it is created.
//
//////////////////////////////////////////////////////////////////////////////
CIgiBox::CIgiBox()
{
       m_bObjectSelect = FALSE;
}

//////////////////////////////////////////////////////////////////////////////
//
//     This constructor creates an object that position is decided by x1, y1,x2
//     and y2.
//
//////////////////////////////////////////////////////////////////////////////
CIgiBox::CIgiBox(int x1, int y1, int x2, int y2)
```

```
{
        m_start.x = x1;
        m_start.y = y1;
        m_end.x = x2;
        m_end.y = y2;
}

CIgiBox::~CIgiBox()
{

}

////////////////////////////////////////////////////////////////////////////
//
//      This function returns TURE if the (x, y) is on the object. It also sets
//      the mark index if the mouse position is on the select mark. If the mouse
//      position is not on the object, it returns FALSE.
//
////////////////////////////////////////////////////////////////////////////
BOOL CIgiBox::SelectPrimitive(int x, int y)
{
        BOOL Is_In = FALSE;

        POINT Point;
        Point.x = x;
        Point.y = y;

        CRect crect(m_start.x, m_start.y, m_end.x, m_end.y);

        int x1, y1, x2, y2, x3, y3, x4, y4;  // coordinator for four conner

        x1 = min(m_start.x,  m_end.x);
        y1 = min(m_start.y,  m_end.y);

        x2 = max(m_start.x,  m_end.x);
        y2 = max(m_start.y,  m_end.y);

        x3 = max(m_start.x,  m_end.x);
        y3 = min(m_start.y,  m_end.y);

        x4 = min(m_start.x,  m_end.x);
        y4 = max(m_start.y,  m_end.y);
```

```
if ( (x <= x1+m_iSquareSize && x >= x1-m_iSquareSize) &&
        (y <= y1+m_iSquareSize && y >= y1-m_iSquareSize) )
{
       m_iMarkIndex = 1;
       m_bSelectMark = TRUE;
       Is_In = TRUE;
}
else if ( (x <= x2+m_iSquareSize && x >= x2-m_iSquareSize) &&
                (y <= y2+m_iSquareSize && y >= y2-m_iSquareSize) )
{
       m_iMarkIndex = 2;
       m_bSelectMark = TRUE;
       Is_In = TRUE;
}
else if ( (x <= x3+m_iSquareSize && x >= x3-m_iSquareSize) &&
     (y <= y3+m_iSquareSize && y >= y3-m_iSquareSize) )
{
       m_iMarkIndex = 3;
       m_bSelectMark = TRUE;
       Is_In = TRUE;
}
else if ( (x <= x4+m_iSquareSize && x >= x4-m_iSquareSize) &&
                (y <= y4+m_iSquareSize && y >= y4-m_iSquareSize) )
{
       m_iMarkIndex = 4;
       m_bSelectMark = TRUE;
       Is_In = TRUE;
}
else if (!m_bFill)  // if no fill box
{
       if ( x > x1 - m_iSquareSize && x < x2 + m_iSquareSize &&
            y > y1 - m_iSquareSize && y < y2 + m_iSquareSize )
       {
              if ( ((x <= x1 + m_iSquareSize) && (x >= x1 - m_iSquareSize)) ||
                      ((x <= x2 + m_iSquareSize) && (x >= x2 - m_iSquareSize)) ||
                      ((y <= y1 + m_iSquareSize) && (y >= y1 - m_iSquareSize)) ||
                      ((y <= y2 + m_iSquareSize) && (y >= y2 - m_iSquareSize)) )
              {
                     m_iMarkIndex = 0;
                     m_bSelectMark = FALSE;
                     Is_In = TRUE;
              }
       }
```

```
        }
        else if ( crect.PtInRect(Point) )
        {
                m_iMarkIndex = 0;
                m_bSelectMark = FALSE;
                Is_In = TRUE;
        }
        return(Is_In);
}

////////////////////////////////////////////////////////////////////////////////
//
//      This function updates the position of the object according to the modification
//      value (dx, dy) and which point the mouse has been clicked on (index).
//
////////////////////////////////////////////////////////////////////////////////
void CIgiBox::ModifyPrimitive(int dx, int dy, int index)
{
        int x1, y1, x2, y2;  // data for four mark conner

        x1 = m_start.x;
        y1 = m_start.y;

        x2 = m_end.x;
        y2 = m_end.y;

        switch (index)
        {
        case 1 : // modify from topleft conner
                x1 = x1 + dx;
                y1 = y1 + dy;
                break;

        case 2 : // modify from lowright conner
                x2 = x2 + dx;      // lowright conner
                y2 = y2 + dy;
                break;

        case 3 : // modify from topright conner
                x2 = x2 + dx;    // topright conner
                y1 = y1 + dy;
                break;
        case 4 : // modify from lowleft conner
                x1 = x1 + dx;     // lowleft conner
                y2 = y2 + dy;
                break;

        default :
                break;
        }   // end switch (index)

        m_start.x = min(x1, x2);
        m_start.y = min(y1, y2);
        m_end.x = max(x1, x2);
        m_end.y = max(y1, y2);
}

/////////////////////////////////////////////////////////////////////////////
//
//      This function draws the box according to its fill mode. If the object is
//      selected, the function draws a small square as the select mark.
//
/////////////////////////////////////////////////////////////////////////////
void CIgiBox::DrawPrimitive(CDC *pDC)
{
        if (m_bObjectSelect)  // draw select mark
        {
                CIgiDrawMark drawMark;

                drawMark.DrawMarkSetup(pDC, m_start.x, m_start.y, 0, 0);
                drawMark.DrawSquare(TRUE);

                drawMark.DrawMarkSetup(pDC, m_end.x, m_end.y, 0, 0);
                drawMark.DrawSquare(TRUE);

                drawMark.DrawMarkSetup(pDC, m_end.x, m_start.y, 0, 0);
                drawMark.DrawSquare(TRUE);

                drawMark.DrawMarkSetup(pDC, m_start.x, m_end.y, 0, 0);
                drawMark.DrawSquare(TRUE);
        }

        if(m_bFill)
        {
                CBrush pBrush(m_cPenColor);
                CPen pen (PS_SOLID,1,m_cPenColor);
                pDC->SelectObject(&pBrush);
                pDC->SelectObject(&pen);
                pDC->Rectangle( m_start.x, m_start.y, m_end.x, m_end.y);
        }
        else
        {
                pDC->SelectObject(GetStockObject (NULL_BRUSH)) ;
                CPen pen (PS_SOLID,1,m_cPenColor);
                pDC->SelectObject(&pen);
                pDC->Rectangle( m_start.x, m_start.y, m_end.x, m_end.y);
        }
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function updates the object position according to its moving distance
//      (dx, dy)
//
//////////////////////////////////////////////////////////////////////////////
void CIgiBox::MovePrimitive(int dx, int dy)
{
        m_start.x = m_start.x + dx;
        m_start.y = m_start.y + dy;
        m_end.x = m_end.x + dx;
        m_end.y = m_end.y + dy;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function inverts the object during it is moving.
//
//////////////////////////////////////////////////////////////////////////////
void CIgiBox::InvertIgiPrimitive(CDC *pDC, int dx, int dy)
{
        int nOldMode = pDC->SetROP2 (R2_NOT);
        pDC->SelectObject(GetStockObject (NULL_BRUSH)) ;
        pDC->Rectangle( m_start.x + dx, m_start.y + dy,
                                        m_end.x + dx, m_end.y + dy );
        pDC->SetROP2 (nOldMode);
}

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

```
//
//      This function reads/writes the object from/to hard disk.
//
///////////////////////////////////////////////////////////////////////////
void CIgiBox::Serialize(CArchive &ar)
{
        if (ar.IsStoring())
        {
                ar << m_cPenColor;
                ar << m_bFill;
                ar << m_start;
                ar << m_end;

        }
        else    // loading
        {
                ar >> m_cPenColor;
                ar >> m_bFill;
                ar >> m_start;
                ar >> m_end;
        }
}

///////////////////////////////////////////////////////////////////////////
//
//      This function returns the color that the object is displayed.
//
///////////////////////////////////////////////////////////////////////////
COLORREF CIgiBox::GetPenColor()
{
        return m_cPenColor;
}

///////////////////////////////////////////////////////////////////////////
//
//      This function returns the status that the object is selected.
//
///////////////////////////////////////////////////////////////////////////
BOOL CIgiBox::GetObjectSelect()
{
        return m_bObjectSelect;
}
```

```
//////////////////////////////////////////////////////////////////////////////
//
//      This function sets the status that the selection of the object.
//
//////////////////////////////////////////////////////////////////////////////
void CIgiBox::SetObjectSelect(BOOL select)
{
        m_bObjectSelect = select;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function sets the color that the object is displayed.
//
//////////////////////////////////////////////////////////////////////////////
void CIgiBox::SetPenColor(COLORREF currentColor)
{
        m_cPenColor = currentColor;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function gets the fill mode for the object.
//
//////////////////////////////////////////////////////////////////////////////
BOOL CIgiBox::GetFillMode()
{
        return m_bFill;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function sets the fill mode for the object.
//
//////////////////////////////////////////////////////////////////////////////
void CIgiBox::SetFillMode(BOOL bFillMode)
{
        m_bFill = bFillMode;
}

//////////////////////////////////////////////////////////////////////////////
//
//      This function returns the point that is not changed when the object is
//      modified.
//
////////////////////////////////////////////////////////////////////////////////
CPoint CIgiBox::GetModifyPoint(int iMarkIndex)
{
        CPoint ptFrom;
        switch (iMarkIndex)
        {
        case 1:
                ptFrom = m_end;
                break;
        case 2:
                ptFrom = m_start;
                break;
        case 3:
                ptFrom.x = m_start.x;
                ptFrom.y = m_end.y;
                break;
        case 4:
                ptFrom.x = m_end.x;
                ptFrom.y = m_start.y;
                break;
        default: break;
        }
        return ptFrom;
}

////////////////////////////////////////////////////////////////////////////////
//
//      This function returns the position of the object
//
////////////////////////////////////////////////////////////////////////////////
void CIgiBox::GetParameter(int *x, int dx)
{
        x[0] = m_start.x + dx;
        x[1] = m_start.y + dx;
        x[2] = m_end.x + dx;
        x[3] = m_end.y + dx;
        x[4] = 0;
        x[5] = 0;
}

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

```
//
//      This function returns the selection status of the object. It returns TRUE
//      if the mouse is clicked on the select mark.
//
////////////////////////////////////////////////////////////////////////////////
BOOL CIgiBox::GetSelectMark()
{
        return m_bSelectMark;
}

////////////////////////////////////////////////////////////////////////////////
//
//      This function returns the mark index when the mouse is clicked on the
//      select mark.
//
////////////////////////////////////////////////////////////////////////////////
int CIgiBox::GetMarkIndex()
{
        return m_iMarkIndex;
}
```