INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
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

© Jiazhong Tu, 2003
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-77722-7
ABSTRACT

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

Jiazhong Tu

This major report focuses 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 processes are followed in the software analysis and design. Software Requirements Specifications 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

1. SOFTWARE REQUIREMENTS SPECIFICATION OF IGI EDITOR ......................... 1

1.1 INTRODUCTION .................................................................................. 2
  1.1.1 Purpose ....................................................................................... 2
  1.1.2 Problem Statement ....................................................................... 2
  1.1.3 References .................................................................................. 3
  1.1.4 Overview ..................................................................................... 3

1.2 OVERALL DESCRIPTION ...................................................................... 4
  1.2.1 Product Perspective ........................................................................ 4
    1.2.1.1 System Interfaces .................................................................. 6
    1.2.1.2 User Interfaces ....................................................................... 6
    1.2.1.3 Hardware Interfaces ................................................................ 9
    1.2.1.4 Software Interfaces ................................................................ 9
    1.2.1.5 Communications Interfaces .................................................. 10
    1.2.1.6 Memory Constraints ............................................................... 10
    1.2.1.7 Operations ............................................................................ 10
    1.2.1.8 Site Adaptation Requirements .............................................. 10

  1.2.2 Product Functions ........................................................................ 10
    1.2.2.1 IGI document management .................................................. 11
    1.2.2.2 IGI document edit ................................................................. 11

  1.2.3 User Characteristics ...................................................................... 12

  1.2.4 Constraints ................................................................................... 12

  1.2.5 Assumptions and Dependencies .................................................. 12

1.3 SPECIFIC REQUIREMENTS .................................................................. 13
  1.3.1 External Interface Requirements ................................................ 13
  1.3.2 Software Product Features .......................................................... 13
    1.3.2.1 Use case 1: User creates a new IGI document .......................... 13
    1.3.2.2 Use case 2: User opens an existing IGI document .................. 15
    1.3.2.3 Use case 3: User saves an active IGI document ...................... 17
    1.3.2.4 Use case 4: User saves an active IGI document with a new name 18
    1.3.2.5 Use case 5: User adds a primitive object to an IGI ................. 20
    1.3.2.6 Use case 6: User selects a primitive object ............................... 25
    1.3.2.7 Use case 7: User moves the selected primitive object(s) .......... 27
    1.3.2.8 Use case 8: User selects all primitive objects .......................... 29
    1.3.2.9 Use case 9: User unselects all selected primitive object(s) ....... 30
    1.3.2.10 Use case 10: User cuts any selected primitive objects ............. 32
    1.3.2.11 Use case 11: User cuts all primitive objects .......................... 34
    1.3.2.12 Use case 12: User undoes the last edit action ....................... 36
    1.3.2.13 Use case 13: User pushes primitive object down ..................... 37
    1.3.2.14 Use case 14: User pushes primitive object down to bottom ...... 40
    1.3.2.15 Use case 15: User pops a primitive object up ........................ 42
    1.3.2.16 Use case 16: User pops primitive object up to top ............... 44
1.3.2.17 Use case 17: User copies any selected primitive object(s) .......... 46
1.3.2.18 Use case 18: User changes any selected primitive object(s) color... 48
1.3.2.19 Use case 19: User rotates a selected stream object.................. 50
1.3.2.20 Use case 20: User modifies a selected primitive object............... 52
1.3.2.21 Global Collaboration Diagram ......................................... 55
1.3.2.22 Class Diagram .................................................................. 56
1.3.3 Performance Requirements ..................................................... 56
1.3.4 Design Constraints ............................................................... 56
1.3.5 Software System Attributes .................................................. 57
  1.3.5.1 Reliability .......................................................................... 57
  1.3.5.2 Availability ........................................................................ 57
  1.3.5.3 Security .............................................................................. 57
  1.3.5.4 Maintainability ................................................................. 57
  1.3.5.5 Portability .......................................................................... 57
1.3.6 Logical Database Requirements .............................................. 57
1.3.7 Other Requirements .............................................................. 57
1.4 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS .................... 58

2. SOFTWARE DESIGN DESCRIPTIONS OF IGI EDITOR ..................... 60

2.1 INTRODUCTION ...................................................................... 61
  2.1.1 Purpose ................................................................................ 61
  2.1.2 Scope .................................................................................. 61
  2.1.3 Document Overview ............................................................ 61
  2.1.4 Definitions and Acronyms .................................................... 61
  2.2 DESIGN METHODOLOGY ...................................................... 64

2.3 ARCHITECTURAL DESIGN (AD) ............................................... 65
  2.3.1 Rational .............................................................................. 65
    2.3.1.1 Application Framework .................................................. 65
    2.3.1.2 The Document – View Architecture ............................... 65
    2.3.1.3 Single Document Interface ............................................ 65
    2.3.1.4 Reading and Writing Document .................................... 66
    2.3.1.5 IGI Document Data Structure ................................... 68
  2.3.2 Architecture ..................................................................... 69
  2.3.3 Object Model ................................................................. 72
    2.3.3.1 Class CMainFrame ....................................................... 73
    2.3.3.2 Class CigliEditorView .................................................. 74
    2.3.3.3 Class CigliEditorDoc .................................................... 75
    2.3.3.4 Class CIGlprimitive ...................................................... 78
    2.3.3.5 Class CPrimitive ........................................................ 80
    2.3.3.6 Class CgliBox .............................................................. 80
    2.3.3.7 Class CgliEllipse ......................................................... 81
    2.3.3.8 Class CgliLine ............................................................. 82
    2.3.3.9 Class CgliTriangle ....................................................... 83
    2.3.3.10 Class CgliStream ....................................................... 84
### Class CgiText

2.3.3.11

Class CgiText

| 84 |

### Class CgiTagName

2.3.3.12

Class CgiTagName

| 85 |

### Class CgiDrawMark

2.3.3.13

Class CgiDrawMark

| 86 |

### Class CTextInputDlg

2.3.3.14

Class CTextInputDlg

| 86 |

### Class CPaletteBar

2.3.3.15

Class CPaletteBar

| 87 |

### Class CColorBar

2.3.3.16

Class CColorBar

| 87 |

### Class CToolDlg

2.3.3.17

Class CToolDlg

| 88 |

### Dynamic Model

2.3.4

User creates a new IGI document

| 89 |

User opens an existing IGI document

| 90 |

User saves an active IGI document

| 91 |

User saves an active IGI document with a new name

| 92 |

User adds a primitive to an IGI

| 93 |

User selects a primitive object

| 99 |

User moves the selected primitive(s)

| 100 |

User selects all primitive objects

| 101 |

User unselects all selected primitive object(s)

| 102 |

User cuts any selected primitive object(s)

| 103 |

User cuts all primitive object(s)

| 104 |

User undoes the last edit action

| 105 |

User pushes primitive object down

| 106 |

User pushes primitive object down to bottom level

| 107 |

User pops a primitive object up one level

| 108 |

User pops primitive object up to top level

| 109 |

User copies any selected primitive object(s)

| 110 |

User changes any selected primitive object(s) color

| 111 |

User rotates a selected stream object

| 112 |

User modifies a selected primitive object

| 113 |

### Functional Model

2.3.5

User creates a new IGI document

| 114 |

### INTERFACE SPECIFICATION (IS)

2.4

Exported Types

| 118 |

Specify Operations

| 119 |

#### Operations for class CigieditorDoc

2.4.2.1

Operations for class CigieditorDoc

| 119 |

#### Operations for class CigieditorView

2.4.2.2

Operations for class CigieditorView

| 133 |

#### Operations for class CIGIprimitve

2.4.2.3

Operations for class CIGIprimitve

| 134 |

#### Operations for class CigiBox

2.4.2.4

Operations for class CigiBox

| 137 |

#### Operations for class CigiElliptse

2.4.2.5

Operations for class CigiElliptse

| 139 |

#### Operations for class CigiLine

2.4.2.6

Operations for class CigiLine

| 141 |

#### Operations for class CigiTriangle

2.4.2.7

Operations for class CigiTriangle

| 142 |

#### Operations for class CigiText

2.4.2.8

Operations for class CigiText

| 144 |

#### Operations for class CigiTagName

2.4.2.9

Operations for class CigiTagName

| 145 |

#### Operations for class CigiStream

2.4.2.10

Operations for class CigiStream

| 146 |

### REFERENCES

3.

| 149 |

### APPENDICES

4.

| 150 |
4.1 Sample of Icon Graphic Image (IGI) ................................................................. 151
4.2 Sample of Source Code .................................................................................. 152
   4.2.1 IGIprimitive.h .......................................................................................... 152
   4.2.2 IGIprimitive.cpp ....................................................................................... 155
   4.2.3 igieditorDoc.cpp ....................................................................................... 178
   4.2.4 igieditorView.cpp ..................................................................................... 229
   4.2.5 IgiBox.cpp ............................................................................................... 254
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>User Interface Layout</td>
<td>7</td>
</tr>
<tr>
<td>1-2</td>
<td>Use Case Diagram</td>
<td>10</td>
</tr>
<tr>
<td>1-3</td>
<td>Collaboration diagram for use case 1</td>
<td>14</td>
</tr>
<tr>
<td>1-4</td>
<td>Sequence diagram for use case 1</td>
<td>14</td>
</tr>
<tr>
<td>1-5</td>
<td>Collaboration diagram for use case 2</td>
<td>16</td>
</tr>
<tr>
<td>1-6</td>
<td>Sequence diagram for use case 2</td>
<td>16</td>
</tr>
<tr>
<td>1-7</td>
<td>Collaboration diagram for use case 3</td>
<td>18</td>
</tr>
<tr>
<td>1-8</td>
<td>Sequence diagram for use case 3</td>
<td>18</td>
</tr>
<tr>
<td>1-9</td>
<td>Collaboration diagram for use case 4</td>
<td>19</td>
</tr>
<tr>
<td>1-10</td>
<td>Sequence diagram for use case 4</td>
<td>20</td>
</tr>
<tr>
<td>1-11</td>
<td>Collaboration diagram for use case 5</td>
<td>24</td>
</tr>
<tr>
<td>1-12</td>
<td>Sequence diagram for use case 5</td>
<td>25</td>
</tr>
<tr>
<td>1-13</td>
<td>Collaboration diagram for use case 6</td>
<td>26</td>
</tr>
<tr>
<td>1-14</td>
<td>Sequence diagram for use case 6</td>
<td>26</td>
</tr>
<tr>
<td>1-15</td>
<td>Collaboration diagram for use case 7</td>
<td>28</td>
</tr>
<tr>
<td>1-16</td>
<td>Sequence diagram for use case 7</td>
<td>28</td>
</tr>
<tr>
<td>1-17</td>
<td>Collaboration diagram for use case 8</td>
<td>29</td>
</tr>
<tr>
<td>1-18</td>
<td>Sequence diagram for use case 8</td>
<td>30</td>
</tr>
<tr>
<td>1-19</td>
<td>Collaboration diagram for use case 9</td>
<td>31</td>
</tr>
<tr>
<td>1-20</td>
<td>Sequence diagram for use case 9</td>
<td>32</td>
</tr>
<tr>
<td>1-21</td>
<td>Collaboration diagram for use case 10</td>
<td>33</td>
</tr>
<tr>
<td>1-22</td>
<td>Sequence diagram for use case 10</td>
<td>34</td>
</tr>
<tr>
<td>1-23</td>
<td>Collaboration diagram for use case 11</td>
<td>35</td>
</tr>
<tr>
<td>1-24</td>
<td>Sequence diagram for use case 11</td>
<td>35</td>
</tr>
<tr>
<td>1-25</td>
<td>Collaboration diagram for use case 12</td>
<td>36</td>
</tr>
<tr>
<td>1-26</td>
<td>Sequence diagram for use case 12</td>
<td>37</td>
</tr>
<tr>
<td>1-27</td>
<td>Before an object push down</td>
<td>38</td>
</tr>
<tr>
<td>1-28</td>
<td>After an object push down</td>
<td>38</td>
</tr>
<tr>
<td>1-29</td>
<td>Collaboration diagram for use case 13</td>
<td>39</td>
</tr>
<tr>
<td>1-30</td>
<td>Sequence diagram for use case 13</td>
<td>40</td>
</tr>
<tr>
<td>1-31</td>
<td>Collaboration diagram for use case 14</td>
<td>41</td>
</tr>
<tr>
<td>1-32</td>
<td>Sequence diagram for use case 14</td>
<td>42</td>
</tr>
<tr>
<td>1-33</td>
<td>Collaboration diagram for use case 15</td>
<td>43</td>
</tr>
<tr>
<td>1-34</td>
<td>Sequence diagram for use case 15</td>
<td>44</td>
</tr>
<tr>
<td>1-35</td>
<td>Collaboration diagram for use case 16</td>
<td>45</td>
</tr>
<tr>
<td>1-36</td>
<td>Sequence diagram for use case 16</td>
<td>46</td>
</tr>
<tr>
<td>1-37</td>
<td>Collaboration diagram for use case 17</td>
<td>47</td>
</tr>
<tr>
<td>1-38</td>
<td>Sequence diagram for use case 17</td>
<td>48</td>
</tr>
<tr>
<td>1-39</td>
<td>Collaboration diagram for use case 18</td>
<td>49</td>
</tr>
<tr>
<td>1-40</td>
<td>Sequence diagram for use case 18</td>
<td>50</td>
</tr>
<tr>
<td>1-41</td>
<td>Collaboration diagram for use case 19</td>
<td>51</td>
</tr>
<tr>
<td>1-42</td>
<td>Sequence diagram for use case 19</td>
<td>52</td>
</tr>
<tr>
<td>1-43</td>
<td>Collaboration diagram for use case 20</td>
<td>53</td>
</tr>
<tr>
<td>Figure 1-44</td>
<td>Sequence diagram for use case 20</td>
<td>54</td>
</tr>
<tr>
<td>Figure 1-45</td>
<td>Collaboration Diagram</td>
<td>55</td>
</tr>
<tr>
<td>Figure 1-46</td>
<td>Domain Model</td>
<td>56</td>
</tr>
<tr>
<td>Figure 2-1</td>
<td>Class relationships</td>
<td>67</td>
</tr>
<tr>
<td>Figure 2-2</td>
<td>Object relationships</td>
<td>67</td>
</tr>
<tr>
<td>Figure 2-3</td>
<td>The serialization process</td>
<td>68</td>
</tr>
<tr>
<td>Figure 2-4</td>
<td>IGI document data structure</td>
<td>68</td>
</tr>
<tr>
<td>Figure 2-5</td>
<td>Main Class and Relationship Diagram</td>
<td>70</td>
</tr>
<tr>
<td>Figure 2-6</td>
<td>The Model-View-Controller Design pattern</td>
<td>71</td>
</tr>
<tr>
<td>Figure 2-7</td>
<td>Main Class and Relationship Diagram</td>
<td>72</td>
</tr>
<tr>
<td>Figure 2-8</td>
<td>Active diagram of creating a new IGI document</td>
<td>89</td>
</tr>
<tr>
<td>Figure 2-9</td>
<td>Active diagram of opening an existing IGI document</td>
<td>90</td>
</tr>
<tr>
<td>Figure 2-10</td>
<td>Active diagram of saving an active IGI document</td>
<td>91</td>
</tr>
<tr>
<td>Figure 2-11</td>
<td>Active diagram of saving an active IGI document with a new name</td>
<td>92</td>
</tr>
<tr>
<td>Figure 2-12</td>
<td>Active diagram of adding a Rectangle/Ellipse object</td>
<td>93</td>
</tr>
<tr>
<td>Figure 2-13</td>
<td>Active Diagram of adding a Line object</td>
<td>94</td>
</tr>
<tr>
<td>Figure 2-14</td>
<td>Active Diagram of adding a Triangle object</td>
<td>95</td>
</tr>
<tr>
<td>Figure 2-15</td>
<td>Active Diagram of adding a Text object</td>
<td>96</td>
</tr>
<tr>
<td>Figure 2-16</td>
<td>Active Diagram of adding a TagName object</td>
<td>97</td>
</tr>
<tr>
<td>Figure 2-17</td>
<td>Active Diagram of adding a Stream object</td>
<td>98</td>
</tr>
<tr>
<td>Figure 2-18</td>
<td>Active Diagram of selecting a primitive object</td>
<td>99</td>
</tr>
<tr>
<td>Figure 2-19</td>
<td>Active Diagram of moving the selected primitive(s)</td>
<td>100</td>
</tr>
<tr>
<td>Figure 2-20</td>
<td>Active Diagram of selecting all primitive objects</td>
<td>101</td>
</tr>
<tr>
<td>Figure 2-21</td>
<td>Active Diagram of unselecting all selected primitive object(s)</td>
<td>102</td>
</tr>
<tr>
<td>Figure 2-22</td>
<td>Active Diagram of cutting selected primitive object(s)</td>
<td>103</td>
</tr>
<tr>
<td>Figure 2-23</td>
<td>Active Diagram of cutting all primitive object(s)</td>
<td>104</td>
</tr>
<tr>
<td>Figure 2-24</td>
<td>Active Diagram of undoing the last edit action</td>
<td>105</td>
</tr>
<tr>
<td>Figure 2-25</td>
<td>Active Diagram of pushing primitive object down</td>
<td>106</td>
</tr>
<tr>
<td>Figure 2-26</td>
<td>Active Diagram of pushing an object down to bottom level</td>
<td>107</td>
</tr>
<tr>
<td>Figure 2-27</td>
<td>Active Diagram of popping an object up one level</td>
<td>108</td>
</tr>
<tr>
<td>Figure 2-28</td>
<td>Active Diagram of popping an object up to top level</td>
<td>109</td>
</tr>
<tr>
<td>Figure 2-29</td>
<td>Active Diagram of copying selected primitive object(s)</td>
<td>110</td>
</tr>
<tr>
<td>Figure 2-30</td>
<td>Active Diagram of changing selected object(s) color</td>
<td>111</td>
</tr>
<tr>
<td>Figure 2-31</td>
<td>Active Diagram of rotating a selected stream object</td>
<td>112</td>
</tr>
<tr>
<td>Figure 2-32</td>
<td>Active Diagram of modifying a selected object</td>
<td>113</td>
</tr>
<tr>
<td>Figure 2-33</td>
<td>Data Flow Diagram</td>
<td>114</td>
</tr>
<tr>
<td>Figure 4-1</td>
<td>A sample of Icon Graphic Image (IGI)</td>
<td>151</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1-1</td>
<td>IGI Primitives</td>
<td>4</td>
</tr>
<tr>
<td>Table 1-2</td>
<td>IGI Primitive color</td>
<td>5</td>
</tr>
<tr>
<td>Table 1-3</td>
<td>Menu configuration and functionality</td>
<td>8</td>
</tr>
<tr>
<td>Table 1-4</td>
<td>Typical course of events of use case 1</td>
<td>13</td>
</tr>
<tr>
<td>Table 1-5</td>
<td>Typical course of events of use case 2</td>
<td>15</td>
</tr>
<tr>
<td>Table 1-6</td>
<td>Typical course of events of use case 3</td>
<td>17</td>
</tr>
<tr>
<td>Table 1-7</td>
<td>Typical course of events of use case 4</td>
<td>19</td>
</tr>
<tr>
<td>Table 1-8</td>
<td>Typical course of events of use case 5</td>
<td>20</td>
</tr>
<tr>
<td>Table 1-9</td>
<td>Typical course of events of use case 6</td>
<td>26</td>
</tr>
<tr>
<td>Table 1-10</td>
<td>Typical course of events of use case 7</td>
<td>27</td>
</tr>
<tr>
<td>Table 1-11</td>
<td>Typical course of events of use case 8</td>
<td>29</td>
</tr>
<tr>
<td>Table 1-12</td>
<td>Typical course of events of use case 9</td>
<td>31</td>
</tr>
<tr>
<td>Table 1-13</td>
<td>Typical course of events of use case 10</td>
<td>33</td>
</tr>
<tr>
<td>Table 1-14</td>
<td>Typical course of events of use case 11</td>
<td>34</td>
</tr>
<tr>
<td>Table 1-15</td>
<td>Typical course of events of use case 12</td>
<td>36</td>
</tr>
<tr>
<td>Table 1-16</td>
<td>Typical course of events of use case 13</td>
<td>39</td>
</tr>
<tr>
<td>Table 1-17</td>
<td>Typical course of events of use case 14</td>
<td>40</td>
</tr>
<tr>
<td>Table 1-18</td>
<td>Typical course of events of use case 15</td>
<td>42</td>
</tr>
<tr>
<td>Table 1-19</td>
<td>Typical course of events of use case 16</td>
<td>44</td>
</tr>
<tr>
<td>Table 1-20</td>
<td>Typical course of events of use case 17</td>
<td>46</td>
</tr>
<tr>
<td>Table 1-21</td>
<td>Typical course of events of use case 18</td>
<td>48</td>
</tr>
<tr>
<td>Table 1-22</td>
<td>Typical course of events of use case 19</td>
<td>51</td>
</tr>
<tr>
<td>Table 1-23</td>
<td>Typical course of events of use case 20</td>
<td>53</td>
</tr>
<tr>
<td>Table 2-1</td>
<td>Attributes and interfaces of class CMainFrame</td>
<td>73</td>
</tr>
<tr>
<td>Table 2-2</td>
<td>Attributes and interfaces of class CBigeditorView</td>
<td>74</td>
</tr>
<tr>
<td>Table 2-3</td>
<td>Attributes and interfaces of class CBigeditorDoc</td>
<td>75</td>
</tr>
<tr>
<td>Table 2-4</td>
<td>Attributes and interfaces of class CBiglprimitive</td>
<td>78</td>
</tr>
<tr>
<td>Table 2-5</td>
<td>Attributes and interfaces of class CPrimitive</td>
<td>80</td>
</tr>
<tr>
<td>Table 2-6</td>
<td>Attributes and interfaces of class CBigiBox</td>
<td>80</td>
</tr>
<tr>
<td>Table 2-7</td>
<td>Attributes and interfaces of class CBigiEllipse</td>
<td>81</td>
</tr>
<tr>
<td>Table 2-8</td>
<td>Attributes and interfaces of class CBigiLine</td>
<td>82</td>
</tr>
<tr>
<td>Table 2-9</td>
<td>Attributes and interfaces of class CBigiTriangle</td>
<td>83</td>
</tr>
<tr>
<td>Table 2-10</td>
<td>Attributes and interfaces of class CBigiStream</td>
<td>84</td>
</tr>
<tr>
<td>Table 2-11</td>
<td>Attributes and interfaces of class CBigiText</td>
<td>84</td>
</tr>
<tr>
<td>Table 2-12</td>
<td>Attributes and interfaces of class CBigiTagName</td>
<td>85</td>
</tr>
<tr>
<td>Table 2-13</td>
<td>Attributes and interfaces of class CBigiDrawMark</td>
<td>86</td>
</tr>
<tr>
<td>Table 2-14</td>
<td>Attributes and interfaces of class CTextField</td>
<td>86</td>
</tr>
<tr>
<td>Table 2-15</td>
<td>Attributes and interfaces of class CPaletteBar</td>
<td>87</td>
</tr>
<tr>
<td>Table 2-16</td>
<td>Attributes and interfaces of class CColorBar</td>
<td>87</td>
</tr>
<tr>
<td>Table 2-17</td>
<td>Attributes and interfaces of class CToolDlg</td>
<td>88</td>
</tr>
<tr>
<td>Table 2-18</td>
<td>Relationship of stream direction and its orientation</td>
<td>147</td>
</tr>
</tbody>
</table>
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 diagram, 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 verify 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 IGIE via adding a primitive object to IGIE edit area easily with graphic user interface. An IGIE includes the following primitives:

<table>
<thead>
<tr>
<th>Table 1-1</th>
<th>IGIE Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primitive</strong></td>
<td><strong>Graphic symbol</strong></td>
</tr>
<tr>
<td>Line</td>
<td>![Line Graphic]</td>
</tr>
<tr>
<td>Rectangle</td>
<td>![Rectangle Graphic]</td>
</tr>
<tr>
<td>Fill rectangle</td>
<td>![Fill Rectangle Graphic]</td>
</tr>
<tr>
<td>Triangle</td>
<td>![Triangle Graphic]</td>
</tr>
<tr>
<td>Fill triangle</td>
<td>![Fill Triangle Graphic]</td>
</tr>
<tr>
<td>Ellipse</td>
<td>![Ellipse Graphic]</td>
</tr>
<tr>
<td>Fill Ellipse</td>
<td>![Fill Ellipse Graphic]</td>
</tr>
<tr>
<td>Text</td>
<td>Text example</td>
</tr>
<tr>
<td>Tag name</td>
<td>Tag name</td>
</tr>
<tr>
<td>Process input stream</td>
<td>![Process Input Stream]</td>
</tr>
<tr>
<td>Process output stream</td>
<td>![Process Output Stream]</td>
</tr>
<tr>
<td>Logic input stream</td>
<td>![Logic Input Stream]</td>
</tr>
<tr>
<td>Logic output stream</td>
<td>![Logic Output Stream]</td>
</tr>
<tr>
<td>Primitive</td>
<td>Graphic symbol</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Analog input stream</td>
<td>←</td>
</tr>
<tr>
<td>Analog output stream</td>
<td>→</td>
</tr>
</tbody>
</table>

An IGI has only one tag name that has default name “xxxxxxxxxxxxxxx”. 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>
<thead>
<tr>
<th>Color</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Cyan</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Magenta</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Yellow</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>White</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Dark blue</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Dark green</td>
<td>0</td>
<td>128</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Dark cyan</td>
<td>0</td>
<td>128</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Dark red</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Dark magenta</td>
<td>128</td>
<td>0</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Dark yellow</td>
<td>128</td>
<td>128</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Dark gray</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Light gray</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td></td>
</tr>
</tbody>
</table>
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
- IGIEdit area

The layout of the user interface is shown below in Figure 1-1.
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>
<thead>
<tr>
<th>Main Menu</th>
<th>Submenu</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>New</td>
<td>Create a new document</td>
</tr>
<tr>
<td>File</td>
<td>Open</td>
<td>Open an existing document</td>
</tr>
<tr>
<td>File</td>
<td>Save</td>
<td>Save the active document</td>
</tr>
<tr>
<td>File</td>
<td>Save as</td>
<td>Save the active document with a new name</td>
</tr>
<tr>
<td>File</td>
<td>Exit</td>
<td>Quit the application; prompts to save documents</td>
</tr>
<tr>
<td>Edit</td>
<td>Cut</td>
<td>Cut the selected primitive(s)</td>
</tr>
<tr>
<td>Edit</td>
<td>Cut All</td>
<td>Cut all primitives</td>
</tr>
<tr>
<td>Edit</td>
<td>Copy</td>
<td>Copy the selected primitive(s)</td>
</tr>
<tr>
<td>Edit</td>
<td>Undo</td>
<td>Undo the last action</td>
</tr>
<tr>
<td>Edit</td>
<td>Redo</td>
<td>Redo the previously undone action</td>
</tr>
<tr>
<td>Edit</td>
<td>Select All</td>
<td>Select all the primitives</td>
</tr>
<tr>
<td>Edit</td>
<td>Unselect All</td>
<td>Unselect all the primitives</td>
</tr>
<tr>
<td>Edit</td>
<td>Push Down</td>
<td>Push the selected primitive one layer down</td>
</tr>
<tr>
<td>Edit</td>
<td>Push To Bottom</td>
<td>Push a selected primitive to bottom</td>
</tr>
<tr>
<td>Edit</td>
<td>Pop Up</td>
<td>Pop the selected primitive one layer up</td>
</tr>
<tr>
<td>Edit</td>
<td>Pop To Top</td>
<td>Pop a selected primitive to top</td>
</tr>
<tr>
<td>View</td>
<td>Toolbars</td>
<td>Configure all the Toolbars</td>
</tr>
<tr>
<td>View</td>
<td>Status Bar</td>
<td>Enable/disable Status Bar</td>
</tr>
<tr>
<td>Primitives</td>
<td>Rectangle</td>
<td>Create a new rectangle</td>
</tr>
<tr>
<td>Primitives</td>
<td>Fill Rectangle</td>
<td>Create a new fill rectangle</td>
</tr>
<tr>
<td>Primitives</td>
<td>Triangle</td>
<td>Create a new triangle</td>
</tr>
<tr>
<td>Primitives</td>
<td>Fill Triangle</td>
<td>Create a new fill triangle</td>
</tr>
<tr>
<td>Primitives</td>
<td>Ellipse</td>
<td>Create a new ellipse</td>
</tr>
<tr>
<td>Primitives</td>
<td>Fill Ellipse</td>
<td>Create a new fill ellipse</td>
</tr>
<tr>
<td>Primitives</td>
<td>Process input stream</td>
<td>Create a new process input stream</td>
</tr>
<tr>
<td>Primitives</td>
<td>Process output stream</td>
<td>Create a new process output stream</td>
</tr>
<tr>
<td>Primitives</td>
<td>Logic input stream</td>
<td>Create a new logic input stream</td>
</tr>
<tr>
<td>Primitives</td>
<td>Logic output stream</td>
<td>Create a new logic output stream</td>
</tr>
<tr>
<td>Primitives</td>
<td>Analog input stream</td>
<td>Create a new analog input stream</td>
</tr>
<tr>
<td>Primitives</td>
<td>Analog output stream</td>
<td>Create a new analog output stream</td>
</tr>
<tr>
<td>Primitives</td>
<td>Text</td>
<td>Create a new text</td>
</tr>
<tr>
<td>Primitives</td>
<td>Tag Name</td>
<td>Create a new tag name</td>
</tr>
<tr>
<td>Main Menu</td>
<td>Submenu</td>
<td>Functionality</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>Primitives</td>
<td>Line</td>
<td>Create a new line</td>
</tr>
<tr>
<td>Help</td>
<td>About IGIEditor</td>
<td>Display program information, version number and copyright</td>
</tr>
</tbody>
</table>

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.

![Use Case Diagram](image)

**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>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “New” menu item, or click “New” toolbar button, or press “Ctrl+N” on keyboard.</td>
<td></td>
</tr>
<tr>
<td>2. The edit area becomes empty.</td>
<td></td>
</tr>
<tr>
<td>3. The title text changes to “untitled”.</td>
<td></td>
</tr>
</tbody>
</table>
1.3.2.1.5 Interaction Diagram for the main scenario

1.3.2.1.5.1 Collaboration diagram

![Collaboration Diagram](image)

Figure 1-3  Collaboration diagram for use case 1

1.3.2.1.5.2 Sequence diagram

![Sequence Diagram](image)

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

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>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Save” menu item, or click “Save” toolbar button, or presses “Ctrl+S” on keyboard.</td>
<td>2. If the active IGI is untitled, pop up “Save As” window.</td>
</tr>
<tr>
<td>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.</td>
<td>4. An IGI file is saved on the hard disk.</td>
</tr>
<tr>
<td></td>
<td>5. If the active IGI is untitled, the title text changes to new IGI name.</td>
</tr>
</tbody>
</table>

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
1.3.2.3.5.2 Sequence diagram

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>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Save As” menu item.</td>
<td>2. Pop up “Save As” window.</td>
</tr>
<tr>
<td>3. Select a folder and type a new IGI name. Click on “Save” button or type “Enter” key.</td>
<td>4. Create a new IGI file and save the active IGI to this new file.</td>
</tr>
<tr>
<td></td>
<td>5. The title text changes to new IGI name.</td>
</tr>
</tbody>
</table>

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

![Collaboration diagram for use case 4](image)

Figure 1-9  Collaboration diagram for use case 4

19
1.3.2.4.5.2 Sequence diagram

![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

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select a menu item from main menu “Primitives”.</td>
<td></td>
</tr>
<tr>
<td>Actor Action</td>
<td>System Response</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>or select a primitive from primitive palette.</td>
<td>2. Create a previous primitive list for undo purpose.</td>
</tr>
<tr>
<td></td>
<td>3. Set the selected primitive as current primitive.</td>
</tr>
<tr>
<td></td>
<td>4. Check the selected primitive palette button on the palette bar.</td>
</tr>
<tr>
<td>5. Select a desired position in the edit area.</td>
<td>7. Create a new primitive object.</td>
</tr>
<tr>
<td>6. Drag the primitive object to the desired position.</td>
<td>8. Set the new primitive object parameters according to the primitive object type, color, and position.</td>
</tr>
<tr>
<td></td>
<td>9. Display the primitive object at the desired position.</td>
</tr>
</tbody>
</table>

**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

![Collaboration Diagram]

Figure 1-11  Collaboration diagram for use case 5
1.3.2.5.5.2 Sequence diagram

![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>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Move mouse over a primitive object.</td>
<td></td>
</tr>
<tr>
<td>2. Click left mouse button.</td>
<td>3. Show a small square on the control points of the primitive object.</td>
</tr>
<tr>
<td>4. The primitive object is ready for any edit action.</td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Move mouse over a selected primitive object.</td>
<td></td>
</tr>
<tr>
<td>2. Click left mouse button.</td>
<td></td>
</tr>
<tr>
<td>3. Hold left mouse button down.</td>
<td></td>
</tr>
<tr>
<td>4. Move the mouse to the desired position.</td>
<td></td>
</tr>
<tr>
<td>5. Release left mouse button.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. Update IGI document.</td>
</tr>
<tr>
<td></td>
<td>7. The primitive object is displayed on the desired position.</td>
</tr>
</tbody>
</table>
1.3.2.7.5 Interaction Diagram for the main scenario

1.3.2.7.5.1 Collaboration diagram

![Collaboration Diagram](image)

Figure 1-15  Collaboration diagram for use case 7

1.3.2.7.5.2 Sequence diagram

![Sequence Diagram](image)

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 IG1 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>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Select All” menu item, or click “Select All” toolbar button.</td>
<td>2. All primitive objects are selected. Display small squares on the control points of all primitive objects.</td>
</tr>
</tbody>
</table>

1.3.2.8.5 Interaction Diagram for the main scenario

1.3.2.8.5.1 Collaboration diagram

![Collaboration diagram]

Figure 1-17 Collaboration diagram for use case 8
1.3.2.8.5.2 Sequence diagram

![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>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>2. All primitive objects are unselected. The small squares on the control points of the selected primitive objects are removed.</td>
</tr>
</tbody>
</table>

### 1.3.2.9.5 Interaction Diagram for the main scenario

#### 1.3.2.9.5.1 Collaboration diagram

![Collaboration diagram](image_url)  

**Figure 1-19** Collaboration 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**

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Cut” menu item, or clicks “Cut” toolbar button, or presses “Delete” on keyboard.</td>
<td>2. Create a previous primitive object list.</td>
</tr>
<tr>
<td></td>
<td>3. All the selected primitive objects are removed from the primitive object list.</td>
</tr>
<tr>
<td></td>
<td>4. Redraw the primitive object list.</td>
</tr>
</tbody>
</table>

### 1.3.2.10.5 Interaction Diagram for the main scenario

#### 1.3.2.10.5.1 Collaboration diagram

![Collaboration diagram for use case 10](image)

**Figure 1-21 Collaboration diagram for use case 10**
1.3.2.10.5.2 Sequence diagram

![Sequence diagram for use case 10](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 | 34 |


<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Cut All” menu item, or clicks “Cut All” toolbar button.</td>
<td>2. Create a previous primitive object list.</td>
</tr>
<tr>
<td></td>
<td>3. All the added primitive objects are deleted from</td>
</tr>
<tr>
<td></td>
<td>the primitive object list.</td>
</tr>
<tr>
<td></td>
<td>4. The edit area is refreshed.</td>
</tr>
</tbody>
</table>

1.3.2.11.5 Interaction Diagram for the main scenario

1.3.2.11.5.1 Collaboration diagram

![Collaboration diagram](image)

Figure 1-23  Collaboration diagram for use case 11

1.3.2.11.5.2 Sequence diagram

![Sequence diagram](image)

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>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Undo” menu item, or clicks “Undo” toolbar button.</td>
<td>2. Check the IGI document is modified?</td>
</tr>
<tr>
<td></td>
<td>3. Swap the current primitive object list with the previous primitive object list.</td>
</tr>
<tr>
<td></td>
<td>4. Redraw the primitive object list on the edit area.</td>
</tr>
</tbody>
</table>

1.3.2.12.5 Interaction Diagram for the main scenario

1.3.2.12.5.1 Collaboration diagram

![Collaboration diagram for use case 12](image)

Figure 1-25  Collaboration diagram for use case 12
1.3.2.12.5.2 Sequence diagram

![Sequence diagram for use case 12]

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 Purpose

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

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

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Push Down” menu item, or clicks “Push Down” toolbar button.</td>
<td>2. Get the selected primitive object position in the primitive object list.</td>
</tr>
<tr>
<td></td>
<td>3. Insert the selected primitive object before its previous object.</td>
</tr>
<tr>
<td></td>
<td>4. Remove the selected primitive object at its current position.</td>
</tr>
<tr>
<td></td>
<td>5. Redraw the primitive object list.</td>
</tr>
</tbody>
</table>

**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

![Collaboration diagram for use case 13](image)

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

<p>| Table 1-17  Typical course of events of use case 14 |
|----------------------------------------|----------------------------------------|
| <strong>Actor Action</strong>                       | <strong>System Response</strong>                    |
| 1. Select “Push To Bottom” menu item, or clicks “Push To Bottom” toolbar button. | |</p>
<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Get the selected primitive object position in the primitive object list.</td>
<td></td>
</tr>
<tr>
<td>3. Add the selected primitive object at the head of primitive object list.</td>
<td></td>
</tr>
<tr>
<td>4. Remove the selected primitive object at its current position</td>
<td></td>
</tr>
<tr>
<td>5. Redraw the primitive object list.</td>
<td></td>
</tr>
</tbody>
</table>

**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

![Collaboration Diagram](image)

Figure 1-31  Collaboration diagram for use case 14
1.3.2.14.5.2 Sequence diagram

![Sequence diagram](image)

**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
<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Pop Up” menu item, or clicks “Pop Up” toolbar button.</td>
<td></td>
</tr>
<tr>
<td>Actor Action</td>
<td>System Response</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>2.</td>
<td>Get the selected primitive object position in the primitive object list.</td>
</tr>
<tr>
<td>3.</td>
<td>Insert the selected primitive object after its next primitive object.</td>
</tr>
<tr>
<td>4.</td>
<td>Remove the selected primitive object at its current position</td>
</tr>
<tr>
<td>5.</td>
<td>Redraw the primitive object list.</td>
</tr>
</tbody>
</table>

### 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

![Collaboration diagram](image)

**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 IG1 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

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Pop To top” menu item, or clicks “Pop to Top” toolbar button.</td>
<td></td>
</tr>
<tr>
<td>Actor Action</td>
<td>System Response</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td>2.</td>
<td>Get the selected primitive object position in the primitive object list.</td>
</tr>
<tr>
<td>3.</td>
<td>Add the selected primitive object at the tail of the primitive object list.</td>
</tr>
<tr>
<td>4.</td>
<td>Remove the selected primitive object at its current position</td>
</tr>
<tr>
<td>5.</td>
<td>Redraw the primitive object list.</td>
</tr>
</tbody>
</table>

**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**

![Collaboration Diagram](image)

**Figure 1-35** Collaboration diagram for use case 16
1.3.2.16.5.2 Sequence diagram

![Sequence diagram for use case 16](image)

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

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select “Copy” menu item, or clicks “Copy” toolbar button</td>
<td></td>
</tr>
<tr>
<td>Actor Action</td>
<td>System Response</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>2. Create the previous primitive object list.</td>
<td></td>
</tr>
<tr>
<td>3. Get the selected primitive object position in the primitive object list.</td>
<td></td>
</tr>
<tr>
<td>4. Create a new primitive object.</td>
<td></td>
</tr>
<tr>
<td>5. Copy the parameters of the selected primitive object to the new primitive object.</td>
<td></td>
</tr>
<tr>
<td>6. Add the new primitive object to the tail of the primitive object list.</td>
<td></td>
</tr>
<tr>
<td>7. Redraw the primitive object list.</td>
<td></td>
</tr>
</tbody>
</table>

**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

![Collaboration diagram for use case 17](image)

**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 Purpose

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
<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select one desired color on the color palette.</td>
<td>2. Create a previous primitive object list.</td>
</tr>
<tr>
<td></td>
<td>3. Set current color to the selected color.</td>
</tr>
<tr>
<td></td>
<td>4. Check the selected color button on the color bar.</td>
</tr>
<tr>
<td></td>
<td>5. Update the selected primitive object color to the</td>
</tr>
<tr>
<td></td>
<td>current color.</td>
</tr>
<tr>
<td></td>
<td>6. Redraw the primitive object list.</td>
</tr>
</tbody>
</table>

1.3.2.18.5 Interaction Diagram for the main scenario

1.3.2.18.5.1 Collaboration diagram

![Collaboration Diagram](image)

Figure 1-39  Collaboration diagram for use case 18
1.3.2.18.5.2 Sequence diagram

![Sequence diagram for use case 18]

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 Purpose
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.

1.3.2.19.5.2 Sequence diagram

![Sequence Diagram](image)

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 Purpose

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

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

1.3.2.20.5 Interaction Diagram for the main scenario

1.3.2.20.5.1 Collaboration diagram

![Collaboration Diagram](image)

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

![Class Diagram Image]

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 disk 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 describes each component 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 describing 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 Editor’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
Figure 2-3  The serialization process


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: CLgieditorApp, CMainFrame, CLgieditorDoc, CLgieditorView.

The CLgieditorApp class – An object of class CLgieditorApp represents an application. The program defines a single global CLgieditorApp 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 CLgieditorApp object.

The CLgieditorDoc class – An object of class CLgieditorDoc holds the data of the application. The different objects of the primitive classes represent the application data. The primitive classes include the following classes: CLgiBox, CLgiEllipse, CLgiLine, CLgiTriangle, CLgiStream, CLgiText, CLgiTagName. 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 CLgiprimitive as an adapter of these primitive classes. An object of class CLgiprimitive is a node of the link list. In the class CLgiprimitive, a PrimitivePointer is used to point to an object of a primitive class according to the type of a primitive.

The CLgieditorView class – An object of class CLgieditorView displays the data of the application. It has a pointer of CLgiprimitive 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.
The architectural design follows the Model/View/Controller (MVC) architectural pattern. MVC consists of three kinds of objects. 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 *CIGlprimitive* 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 coordinates the actions of the model and the view by sending appropriate messages to them. It passes the command to *CIGlprimitive* 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 relationship 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.
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 `CBorderColor` is created when the function `CMainFrame::OnCreate()` is called. From the object of class `CPaletterBar` and `CBorderColor`, 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 `CLgieditorDoc::OnPaletteText()`.

An object of class `CLgiDrawMark` 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

<table>
<thead>
<tr>
<th>Class CMainFrame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attribute:</strong></td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>\texttt{BOOL m_bToolTips;}</td>
</tr>
<tr>
<td>\texttt{BOOL m_bColor;}</td>
</tr>
<tr>
<td>protected:</td>
</tr>
<tr>
<td>\texttt{CToolBar m_wndMainBar;}</td>
</tr>
</tbody>
</table>
### Class CMainFrame

<table>
<thead>
<tr>
<th>CPaletteBar</th>
<th>m_wndPaletteBar;</th>
</tr>
</thead>
<tbody>
<tr>
<td>CColorBar</td>
<td>m_wndColorBar;</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Class CigieditorView</th>
</tr>
</thead>
</table>

**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 OnMouseMove(UINT nFlags, CPoint point);

afx_msg BOOL OnEraseBkgnd(CDC* pDC);

**private:**

BOOL MovePrimitive(CPoint point);
2.3.3.3 Class CigieditorDoc

Table 2-3 Attributes and interfaces of class CigieditorDoc

<table>
<thead>
<tr>
<th>Class CigieditorDoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CTypedPtrList&lt;CObList, CIGIprimitive*&gt; m_primitiveList;</td>
</tr>
<tr>
<td>BOOL m_bColor;</td>
</tr>
<tr>
<td>protected:</td>
</tr>
<tr>
<td>CTypedPtrList&lt;CObList, CIGIprimitive*&gt; m_primitivePrevList;</td>
</tr>
<tr>
<td>int m_iColorPalette[16];</td>
</tr>
<tr>
<td>int m_iPrimitivePalette[16];</td>
</tr>
<tr>
<td>int m_iSelectOrder;</td>
</tr>
<tr>
<td>int m_iCountSelect;</td>
</tr>
<tr>
<td>int m_iTagNameFlag;</td>
</tr>
<tr>
<td>BOOL m_bFillMode;</td>
</tr>
<tr>
<td>EditModelIndex m_iEditMode;</td>
</tr>
<tr>
<td>PalettePrimitive m_iCurrentPrimitive;</td>
</tr>
<tr>
<td>COLORREF m_cCurrentColor;</td>
</tr>
<tr>
<td>CString m_cTextInput;</td>
</tr>
<tr>
<td>Method():</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>void OnKeyDelete();</td>
</tr>
<tr>
<td>void InvertGIPrimitive(CDC* pDC, CIGIprimitive* p, int dx, int dy);</td>
</tr>
<tr>
<td>CIGIprimitive* GetSelectGIStream();</td>
</tr>
<tr>
<td>void IncreaseOrientation(CIGIprimitive* p);</td>
</tr>
<tr>
<td>int GetNewStreamNumber(int iType, int iDir);</td>
</tr>
<tr>
<td>CString GetTextInput();</td>
</tr>
<tr>
<td>void SetTagNameFlag(int tagNameFlag);</td>
</tr>
<tr>
<td>int GetTagNameFlag();</td>
</tr>
<tr>
<td>void SetSelectOrder(int selectOrder);</td>
</tr>
<tr>
<td>void ModifyGIPrimitive(CIGIprimitive* p, int dx, int dy, int iIndex);</td>
</tr>
</tbody>
</table>
class CdlgeditorDoc

void SetEditMode(EditModeIndex editMode);
void MoveGIPrimitiveList(int dx, int dy);
void SetGIObjectSelectList(BOOL select);
void DrawGIPrimitiveList(CDC *pDC);
void UnSetPaletteBar();

int GetSelectOrder();
BOOL GetFillMode();

CGIprimitive* SelectGIPrimitive(CPoint point);
CGIprimitive* NewGIPrimitive(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:
CGIprimitive* CopyGIPrimitive(CGIprimitive* pGIprimitive, int dx);
void SwapList();
void CreatePreviousList();
void SetPaletteBar(int index);
void SetColorBar(int index);
void InitDocument();
void UpdateDocument();
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 OnPoupOneUp();
afx_msg void OnPoupToTop();

afx_msg void OnColorBlack();
afx_msg void OnColorYellow();
afx_msg void OnColorRed();
Class CLgieditorDoc

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

```cpp
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);
```

```cpp
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

<table>
<thead>
<tr>
<th>Class CIGIprimitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>private:</td>
</tr>
</tbody>
</table>

78
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 GetGltextinput();
void SetGltextinput(CString cTextInput);
BOOL GetFillMode();
COLORREF GetPenColor();
PalettePrimitive GetPrimitiveItem();
void GetParameter(int* x, int dx);
BOOL GetGIObjectSelect();
void UpdateColor(COLORREF currentColor);
void MoveGIPrimitive(int dx, int dy);
void SetGIObjectSelect(BOOL select);
void InvertGIPrimitive(CDC* pDC, int dx, int dy);
void ModifyPrimitive(int dx, int dy, int index);
int GetMarkIndex();
BOOL GetSelectMark();
BOOL SelectGIPrimitive(CPoint point);
void SetPenColor(COLORREF currentColor);
void NewPrimitive();
void SetParameter(int x[6]);
void SetPrimitiveType(PalettePrimitive item);
BOOL DrawGIprimitive(CDC* pDC);
CPoint GetModifyPoint(int iMarkIndex, int iFromIndex);

virtual void Serialize(CArchive& ar);
2.3.3.5 Class CPrimitive

<table>
<thead>
<tr>
<th>Table 2-5</th>
<th>Attributes and interfaces of class CPrimitive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Class CPrimitive</strong></td>
</tr>
<tr>
<td>Attribute:</td>
<td></td>
</tr>
<tr>
<td>protected:</td>
<td>int m_iSquareSize;</td>
</tr>
<tr>
<td></td>
<td>COLORREF m_cPenColor;</td>
</tr>
<tr>
<td></td>
<td>BOOL m_bObjectSelect;</td>
</tr>
<tr>
<td></td>
<td>BOOL m_bSelectMark;</td>
</tr>
<tr>
<td></td>
<td>int m_iMarkIndex;</td>
</tr>
<tr>
<td>Method():</td>
<td></td>
</tr>
<tr>
<td>public:</td>
<td>virtual void DrawPrimitive(CDC* pDC);</td>
</tr>
<tr>
<td></td>
<td>virtual void Serialize(CArchive &amp;ar);</td>
</tr>
<tr>
<td></td>
<td>virtual BOOL SelectPrimitive(int x, int y);</td>
</tr>
<tr>
<td></td>
<td>virtual void ModifyPrimitive(int dx, int dy, int index);</td>
</tr>
<tr>
<td></td>
<td>virtual void MovePrimitive(int DX, int DY);</td>
</tr>
<tr>
<td></td>
<td>virtual void InvertIgiPrimitive(CDC* pDC, int dx, int dy);</td>
</tr>
<tr>
<td></td>
<td>virtual void GetParameter(int* x, int dx);</td>
</tr>
<tr>
<td></td>
<td>virtual void SetPenColor(COLORREF currentColor);</td>
</tr>
<tr>
<td></td>
<td>virtual void SetObjectSelect(BOOL select);</td>
</tr>
<tr>
<td></td>
<td>virtual BOOL GetObjectSelect();</td>
</tr>
<tr>
<td></td>
<td>virtual COLORREF GetPenColor();</td>
</tr>
</tbody>
</table>

2.3.3.6 Class CIgiBox

<table>
<thead>
<tr>
<th>Table 2-6</th>
<th>Attributes and interfaces of class CIgiBox</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Class CIgiBox</strong></td>
</tr>
<tr>
<td>Attribute:</td>
<td></td>
</tr>
<tr>
<td>private:</td>
<td>POINT m_end;</td>
</tr>
<tr>
<td></td>
<td>POINT m_start;</td>
</tr>
<tr>
<td></td>
<td>BOOL m_bFill;</td>
</tr>
<tr>
<td>Method():</td>
<td></td>
</tr>
<tr>
<td>public:</td>
<td></td>
</tr>
</tbody>
</table>

80
Class CgiBox

CgiBox();
CgiBox(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 InvertGigiPrimitive(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 CgiEllipse

Table 2-7 Attributes and interfaces of class CgiEllipse

<table>
<thead>
<tr>
<th>Class CgiEllipse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>private:</td>
</tr>
<tr>
<td>POINT m_lowright;</td>
</tr>
<tr>
<td>POINT m_topleft;</td>
</tr>
<tr>
<td>BOOL m_bFill;</td>
</tr>
<tr>
<td>Method():</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CgiEllipse();</td>
</tr>
<tr>
<td>CgiEllipse(int x1, int y1, int x2, int y2);</td>
</tr>
<tr>
<td>int GetMarkIndex();</td>
</tr>
<tr>
<td>BOOL GetSelectMark();</td>
</tr>
</tbody>
</table>
### 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

<table>
<thead>
<tr>
<th>Class CIGiLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>private:</td>
</tr>
<tr>
<td>POINT m_start;</td>
</tr>
<tr>
<td>POINT m_end;</td>
</tr>
<tr>
<td>Method():</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CIGiLine();</td>
</tr>
<tr>
<td>CIGiLine(int x1, int y1, int x2, int y2);</td>
</tr>
<tr>
<td>int GetMarkIndex();</td>
</tr>
<tr>
<td>BOOL GetSelectMark();</td>
</tr>
<tr>
<td>void GetParameter(int* x, int dx);</td>
</tr>
<tr>
<td>CPoint GetModifyPoint(int iMarkIndex);</td>
</tr>
<tr>
<td>void SetPenColor(COLORREF currentColor);</td>
</tr>
<tr>
<td>void SetObjectSelect(BOOL select);</td>
</tr>
<tr>
<td>BOOL GetObjectSelect();</td>
</tr>
<tr>
<td>COLORREF GetPenColor();</td>
</tr>
<tr>
<td>void Serialize(CArchive &amp;ar);</td>
</tr>
<tr>
<td>void InvertIgiPrimitive(CDC* pDC, int dx, int dy);</td>
</tr>
<tr>
<td>void MovePrimitive (int dx, int dy);</td>
</tr>
</tbody>
</table>
### Class CgiLine

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

### 2.3.3.9 Class CgiTriangle

#### Table 2-9 Attributes and interfaces of class CgiTriangle

<table>
<thead>
<tr>
<th>Class CgiTriangle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attribute:</strong></td>
</tr>
<tr>
<td>private:</td>
</tr>
<tr>
<td>BOOL m_bFill;</td>
</tr>
<tr>
<td>POINT m_first;</td>
</tr>
<tr>
<td>POINT m_second;</td>
</tr>
<tr>
<td>POINT m_third;</td>
</tr>
<tr>
<td><strong>Method():</strong></td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CgiTriangle();</td>
</tr>
<tr>
<td>CgiTriangle(int x1, int y1, int x2, int y2, int x3, int y3);</td>
</tr>
<tr>
<td>int GetMarkIndex();</td>
</tr>
<tr>
<td>BOOL GetSelectMark();</td>
</tr>
<tr>
<td>void GetParameter(int* x, int dx);</td>
</tr>
<tr>
<td>CPoint GetModifyPoint(int iMarkIndex, int iFromIndex);</td>
</tr>
<tr>
<td>void SetFillMode(BOOL bFillMode);</td>
</tr>
<tr>
<td>BOOL GetFillMode();</td>
</tr>
<tr>
<td>void SetPenColor(COLORREF currentColor);</td>
</tr>
<tr>
<td>void SetObjectSelect(BOOL select);</td>
</tr>
<tr>
<td>BOOL GetObjectSelect();</td>
</tr>
<tr>
<td>COLORREF GetPenColor();</td>
</tr>
<tr>
<td>void Serialize(CArchive &amp;ar);</td>
</tr>
<tr>
<td>void InvertGigiPrimitive(CDC* pDC, int dx, int dy); //, int Off);</td>
</tr>
<tr>
<td>void MovePrimitive (int dx, int dy);</td>
</tr>
<tr>
<td>void DrawPrimitive(CDC* pDC);</td>
</tr>
<tr>
<td>void ModifyPrimitive (int dx, int dy, int index);</td>
</tr>
<tr>
<td>BOOL SelectPrimitive (int x, int y);</td>
</tr>
<tr>
<td>private:</td>
</tr>
<tr>
<td>BOOL SelectOnLine(int x, int y, POINT start, POINT end);</td>
</tr>
</tbody>
</table>
2.3.3.10 Class CgiStream

Table 2-10 Attributes and interfaces of class CgiStream

<table>
<thead>
<tr>
<th>Class CgiStream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>private:</td>
</tr>
<tr>
<td>int StreamArrow;</td>
</tr>
<tr>
<td>int StreamLength;</td>
</tr>
<tr>
<td>int m_iNoStream;</td>
</tr>
<tr>
<td>int m_iOrientation;</td>
</tr>
<tr>
<td>int m_iDir;</td>
</tr>
<tr>
<td>int m_iType;</td>
</tr>
<tr>
<td>POINT m_start;</td>
</tr>
<tr>
<td>Method():</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CgiStream();</td>
</tr>
<tr>
<td>CgiStream(int x, int y, int iType, int iDir, int iOrientation, int iNoStream);</td>
</tr>
<tr>
<td>void GetParameter(int* x, int dx);</td>
</tr>
<tr>
<td>int GetStreamOrientation();</td>
</tr>
<tr>
<td>int GetStreamDir();</td>
</tr>
<tr>
<td>int GetStreamType();</td>
</tr>
<tr>
<td>void SetStreamNumber(int iNoStream);</td>
</tr>
<tr>
<td>int GetStreamNumber();</td>
</tr>
<tr>
<td>void SetObjectSelect(BOOL select);</td>
</tr>
<tr>
<td>BOOL GetObjectSelect();</td>
</tr>
<tr>
<td>void IncreaseOrientation();</td>
</tr>
<tr>
<td>void Serialize(CArchive &amp;ar);</td>
</tr>
<tr>
<td>void InvertGigiPrimitive(CDC* pDC, int dx, int dy);</td>
</tr>
<tr>
<td>void MovePrimitive (int dx, int dy);</td>
</tr>
<tr>
<td>void DrawPrimitive(CDC* pDC);</td>
</tr>
<tr>
<td>BOOL SelectPrimitive (int x, int y);</td>
</tr>
</tbody>
</table>

2.3.3.11 Class CgiText

Table 2-11 Attributes and interfaces of class CgiText

<table>
<thead>
<tr>
<th>Class CgiText</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>private:</td>
</tr>
<tr>
<td>int m_iMarkHigh;</td>
</tr>
</tbody>
</table>
### Class CgiText

| Method():
|---|
| public:
| CgiText();
| CgiText(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 InvertGiglPrimitive(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 CgiTagName

#### Table 2-12 Attributes and interfaces of class CgiTagName

<table>
<thead>
<tr>
<th>Attribute:</th>
</tr>
</thead>
<tbody>
<tr>
<td>private:</td>
</tr>
<tr>
<td>int m_iMarkHigh;</td>
</tr>
<tr>
<td>int m_iMarkLen;</td>
</tr>
<tr>
<td>POINT m_start;</td>
</tr>
<tr>
<td>CString m_cString;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method():</th>
</tr>
</thead>
<tbody>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CgiTagName();</td>
</tr>
<tr>
<td>CgiTagName(int x, int y, int len, int high);</td>
</tr>
<tr>
<td>void Serialize(CArchive &amp;ar);</td>
</tr>
<tr>
<td>void InvertGiglPrimitive(CDC* pDC, int dx, int dy);</td>
</tr>
<tr>
<td>void MovePrimitive (int dx, int dy);</td>
</tr>
<tr>
<td>void DrawPrimitive(CDC* pDC);</td>
</tr>
</tbody>
</table>
### Class CgiTagName

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL SelectPrimitive (int x, int y);</td>
</tr>
<tr>
<td>void GetParameter(int* x, int dx);</td>
</tr>
<tr>
<td>BOOL GetObjectSelect();</td>
</tr>
<tr>
<td>COLORREF GetPenColor();</td>
</tr>
<tr>
<td>void SetObjectSelect(BOOL select);</td>
</tr>
<tr>
<td>void SetPenColor(COLORREF currentColor);</td>
</tr>
</tbody>
</table>

#### 2.3.3.13 Class CgiDrawMark

**Table 2-13 Attributes and interfaces of class CgiDrawMark**

<table>
<thead>
<tr>
<th>Attribute:</th>
</tr>
</thead>
<tbody>
<tr>
<td>private:</td>
</tr>
<tr>
<td>int m_iSquareSize;</td>
</tr>
<tr>
<td>int m_iY2;</td>
</tr>
<tr>
<td>int m_iX2;</td>
</tr>
<tr>
<td>int m_iY1;</td>
</tr>
<tr>
<td>int m_iX1;</td>
</tr>
<tr>
<td>CDC* m_pDC;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method():</th>
</tr>
</thead>
<tbody>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CgiDrawMark();</td>
</tr>
<tr>
<td>void DrawMarkSetup(CDC *p, int X1, int Y1, int X2, int Y2);</td>
</tr>
<tr>
<td>void DrawSquare(BOOL yes);</td>
</tr>
<tr>
<td>void DrawTextMark(BOOL yes);</td>
</tr>
</tbody>
</table>

#### 2.3.3.14 Class CTextInputDlg

**Table 2-14 Attributes and interfaces of class CTextInputDlg**

<table>
<thead>
<tr>
<th>Attribute:</th>
</tr>
</thead>
<tbody>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CEdit m_TextInput;</td>
</tr>
<tr>
<td>CString m_cTextInput;</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Class CPaletteBar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>protected:</td>
</tr>
<tr>
<td>BOOL m_bRectangle;</td>
</tr>
<tr>
<td>UINT m_nColumns;</td>
</tr>
<tr>
<td>Method:</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CPaletteBar();</td>
</tr>
<tr>
<td>UINT GetColumns();</td>
</tr>
<tr>
<td>void SetColumns(UINT nColumns);</td>
</tr>
</tbody>
</table>

2.3.3.16 Class CColorBar

Table 2-16 Attributes and interfaces of class CColorBar

<table>
<thead>
<tr>
<th>Class CColorBar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>protected:</td>
</tr>
<tr>
<td>UINT m_nColorColumns;</td>
</tr>
<tr>
<td>Method:</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CColorBar();</td>
</tr>
<tr>
<td>UINT GetColumns();</td>
</tr>
</tbody>
</table>
### Class CColorBar

```cpp
void SetColumns(UINT nColorColumns);
```

#### 2.3.3.17 Class CToolDlg

**Table 2-17 Attributes and interfaces of class CToolDlg**

<table>
<thead>
<tr>
<th>Class CToolDlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute:</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>int m_nToolTips;</td>
</tr>
<tr>
<td>int m_nColumns;</td>
</tr>
<tr>
<td>int m_nColor;</td>
</tr>
<tr>
<td>BOOL m_bPalette;</td>
</tr>
<tr>
<td>BOOL m_bMain;</td>
</tr>
<tr>
<td>BOOL m_bColorbar;</td>
</tr>
<tr>
<td>Method():</td>
</tr>
<tr>
<td>public:</td>
</tr>
<tr>
<td>CToolDlg(CWnd* pParent = NULL); // standard constructor</td>
</tr>
<tr>
<td>protected:</td>
</tr>
<tr>
<td>virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support</td>
</tr>
</tbody>
</table>
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

![Active diagram of creating a new IGI document](image)

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)

![Diagram showing the process of moving selected primitives](image)

**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

![Diagram of pushing an object down to bottom level]

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

107
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 IGIDocument 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_primitiveList” 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_primitiveList”.
   - **Output:** The objects in the “m_primitiveList” 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_primitiveList” 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 IGIDocument node is created and is added to the object list “m_primitiveList”.

115
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_primitiveList” for “Cut”. All objects are removed and “m_primitiveList” 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_primitiveList” 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 Primitive List” 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_primitiveList”. OnPopOneUp() will swap the selected object node with its next node in the “m_primitiveList”. OnPushBottom() will move the selected object node to the head of the “m_primitiveList”. OnPopToTop() will move the selected object node to the tail of the “m_primitiveList”.
Output: The objects in the “m Primitive List” 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_primitiveList”.
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_primitiveList”.
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,
} EditModelIndex;

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
{
  ClgiEllipse *plGIELLIPSE;  // pointer to ellipse object
  ClgiBox      *plGIBOX;     // pointer to box object
  ClgiTriangle *plGITRIANGLE; // pointer to triangle object
  ClgiLine     *plGILINE;    // pointer to line object
  ClgiText     *plGITEXT;     // pointer to text object
  ClgiTagName  *plGITagName;  // pointer to tagname object
  ClgiStream   *plGISTREAM;   // pointer to stream object
} ObjectPointer;

2.4.2 Specify Operations

2.4.2.1 Operations for class ClgieeditorDoc

1. BOOL ClgieeditorDoc::OnNewDocument()

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

2. BOOL ClgieeditorDoc::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 primitive 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 CImgeditorDoc:: 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 CImgeditorDoc:: 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 CImgeditorDoc:: 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 CImgeditorDoc:: 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 CImgeditorDoc:: 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 CImgeditorDoc:: 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 CImgeditorDoc:: 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 CImgeditorDoc:: 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 CImgeditorDoc:: 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 CIgeditorDoc:: 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 CIgeditorDoc:: 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 CIgeditorDoc:: 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 CIgeditorDoc:: 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 CIgeditorDoc:: 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 CIgeditorDoc:: 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 CIgeditorDoc:: 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 CIgeditorDoc:: 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 CIgeditorDoc:: OnPaletteRectangle ()

125
Description:
This command sets Rectangle as default primitive for adding new primitive object. It checks the Rectangle button on the primitive bar.

57. void ClgieditorDoc:: 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 ClgieditorDoc:: 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 ClgieditorDoc:: 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 ClgieditorDoc:: 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 ClgieditorDoc:: 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 ClgieditorDoc:: 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 ClgieditorDoc:: 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 ClgieditorDoc:: 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 ClgieditorDoc:: 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 CIgiedReaderDoc::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 CIgiedReaderDoc::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 CIgiedReaderDoc::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 CIgiedReaderDoc::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 CIgiedReaderDoc::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 CIgiedReaderDoc::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 CIgiedReaderDoc::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 CIgiedReaderDoc::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. CIGieditorView
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 IG1 document to empty.

100. void ClgieditorDoc::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 ClgieditorDoc::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 ClgieditorDoc::SetGIOBJECTSelectList(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 ClgieditorDoc::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 ClgieditorDoc::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 ClgieditorDoc::UpdateColor()

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

106. void ClgieditorDoc::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 ClgieditorDoc::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 kepted 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::GetSelectGIstream()

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
CIGiPrimitive::DrawGIPrimitive(). 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 ClgieditorView::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 ClgieditorView::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 ClgieditorView::ModifyPrimitive(CPoint point)

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

9.    BOOL ClgieditorView::MovePrimitive(CPoint point)

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

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

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

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

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

12.   void ClgieditorView::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::InvertGIPrimitive(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::SetGIObjectSelect(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::GetGIObjectSelect()
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::SetGItextinput(CString cTextInput)
Description:
This function sets input text string for the TEXT primitive.
21. CString CIGIPrimitive::GetGItextinput()
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 TRUE 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 CLgiBox::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 CLgiBox::MovePrimitive(int dx, int dy)
Description:
This function updates the object position according to its moving distance (dx, dy).

7. void CLgiBox::InvertIgiPrimitive(CDC *pDC, int dx, int dy)
Description:
This function inverts the object during it is moving.

8. void CLgiBox::Serialize(CArchive &ar)
Description:
This function reads/writes the object from/to a file on the hard disk.

9. COLORREF CLgiBox::GetPenColor()
Description:
This function returns the color that the object is displayed.

10. BOOL CLgiBox::GetObjectSelect()
Description:
This function returns the status that the object is selected.

11. void CLgiBox::SetObjectSelect(BOOL select)
Description:
This function sets the status that the selection of the object.

12. void CLgiBox::SetPenColor(COLORREF currentColor)
Description:
This function sets the color that the object is displayed.

13. BOOL CLgiBox::GetFillMode()
Description:
This function gets the fill mode for the object.

14. void CLgiBox::SetFillMode(BOOL bFillMode)
Description:
This function sets the fill mode for the object.

15. CPoint CLgiBox::GetModifyPoint(int iMarkIndex)
Description:
This function returns the point that is not changed when the object is modified.

16. void CgiBox::GetParameter(int *x, int dx)
Description:
This function returns the position of the object.

17. BOOL CgiBox::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 CgiBox::GetMarkIndex()
Description:
This function returns the mark index when the mouse is clicked on the select mark.

2.4.2.5 Operations for class CgiEllipse

1. CgiEllipse::CgiEllipse ()
Description:
This constructor sets the object not selected when it is created.

2. CgiEllipse::CgiEllipse(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 CgiEllipse::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 CgiEllipse::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 CgiEllipse::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 CgiEllipse::MovePrimitive(int dx, int dy)
Description:
This function updates the object position according to its moving distance (dx, dy).
7. void CLgiEllipse::InvertIGIPrimitive(CDC *pDC, int dx, int dy)
   Description:
   This function inverts the object during it is moving.

8. void CLgiEllipse::Serialize(CArchive &ar)
   Description:
   This function reads/writes the object from/to a file on the hard disk.

9. COLORREF CLgiEllipse::GetPenColor()
   Description:
   This function returns the color that the object is displayed.

10. BOOL CLgiEllipse::GetObjectSelect()
    Description:
    This function returns the status that the object is selected.

11. void CLgiEllipse::SetObjectSelect(BOOL select)
    Description:
    This function sets the status that the selection of the object.

12. void CLgiEllipse::SetPenColor(COLORREF currentColor)
    Description:
    This function sets the color that the object is displayed.

13. BOOL CLgiEllipse::GetFillMode()
    Description:
    This function gets the fill mode for the object.

14. void CLgiEllipse::SetFillMode(BOOL bFillMode)
    Description:
    This function sets the fill mode for the object.

15. CPoint CLgiEllipse::GetModifyPoint(int iMarkIndex)
    Description:
    This function returns the point that is not changed when the object is modified.

16. void CLgiEllipse::GetParameter(int *x, int dx)
    Description:
    This function returns the position of the object.

17. BOOL CLgiEllipse::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 CLgiEllipse::GetMarkIndex()
    Description:
This function returns the mark index when the mouse is clicked on the select mark.

2.4.2.6 Operations for class CLine

1. CLine::CLine()
   Description:
   This constructor sets the object not selected when it is created.

2. CLine::CLine(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 CLine::SelectPrimitive(int x, int y)
   Description:
   This function returns TRUE 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 CLine::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 CLine::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 CLine::MovePrimitive(int dx, int dy)
   Description:
   This function updates the object position according to its moving distance (dx, dy).

7. void CLine::InvertPrimitive(CDC *pDC, int dx, int dy)
   Description:
   This function inverts the object during it is moving.

8. void CLine::Serialize(CArchive &ar)
   Description:
   This function reads/writes the object from/to hard disk.

9. COLORREF CLine::GetPenColor()
   Description:
   This function returns the color that the object is displayed.

10. BOOL CLine::GetObjectSelect()
Description:
This function returns the status that the object is selected.

11. void ClgiLine::SetObjectSelect(BOOL select)

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

12. void ClgiLine::SetPenColor(COLORREF currentColor)

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

13. CPoint ClgiLine::GetModifyPoint(int iMarkIndex)

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

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

Description:
This function returns the position of the object.

15. BOOL ClgiLine::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 ClgiLine::GetMarkIndex()

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

### 2.4.2.7 Operations for class ClgiTriangle

1. ClgiTriangle::ClgiTriangle()

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

2. ClgiTriangle::ClgiTriangle(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 ClgiTriangle::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 ClgiTriangle::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 CgiTriangle::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 CgiTriangle::MovePrimitive(int dx, int dy)`

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

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

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

8. `void CgiTriangle::Serialize(CArchive &ar)`

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

9. `COLORREF CgiTriangle::GetPenColor()`

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

10. `BOOL CgiTriangle::GetObjectSelect()`

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

11. `void CgiTriangle::SetObjectSelect(BOOL select)`

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

12. `void CgiTriangle::SetPenColor(COLORREF currentColor)`

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

13. `BOOL CgiTriangle::GetFillMode()`

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

14. `void CgiTriangle::SetFillMode(BOOL bFillMode)`

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

15. `CPoint CgiTriangle::GetModifyPoint(int iMarkIndex, int iFromIndex)`

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

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

Description:
This function returns the position of the object.

17. BOOL CgiTriangle::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 CgiTriangle::GetMarkIndex()

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

2.4.2.8 Operations for class CgiText

1. CgiText::CgiText()

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

2. CgiText::CgiText(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 CgiText::SelectPrimitive(int x, int y)

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

4. void CgiText::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 CgiText::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 CgiText::InvertGgiPrimitive(CDC *pDC, int dx, int dy)

Description:
This function inverts the object during it is moving.
7. void CglText::Serialize(CArchive &ar)
   Description:
   This function reads/writes the object from/to hard disk.
8. COLORREF CglText::GetPenColor()
   Description:
   This function returns the color that the object is displayed.
9. BOOL CglText::GetObjectSelect()
   Description:
   This function returns the status that the object is selected.
10. void CglText::SetObjectSelect(BOOL select)
    Description:
    This function sets the status that the selection of the object.
11. void CglText::SetPenColor(COLORREF currentColor)
    Description:
    This function sets the color that the object is displayed.
12. void CglText::GetParameter(int *x, int dx)
    Description:
    This function returns the position and the size of the object.
13. void CglText::SetTextInput(CString cTextInput)
    Description:
    This function set the text string for the object.
14. CString CglText::GetTextInput()
    Description:
    This function returns the text string of the object.

2.4.2.9 Operations for class CgliTagName

1. CgliTagName::CgliTagName()
   Description:
   This constructor sets the object not selected when it is created.
2. CgliTagName::CgliTagName(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 CgliTagName::SelectPrimitive(int x, int y)
   Description:
This function returns TRUE if the (x, y) is on the object. If the mouse position is not on the object, it returns FALSE.

4. **void ClgiTagName::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 ClgiTagName::MovePrimitive(int dx, int dy)**
   
   *Description:*
   This function updates the object position according to its moving distance (dx, dy).

6. **void ClgiTagName::InvertIgiPrimitive(CDC *pDC, int dx, int dy)**
   
   *Description:*
   This function inverts the object during it is moving.

7. **void ClgiTagName::Serialize(CArchive &ar)**
   
   *Description:*
   This function reads/writes the object from/to hard disk.

8. **COLORREF ClgiTagName::GetPenColor()**
   
   *Description:*
   This function returns the color that the object is displayed.

9. **BOOL ClgiTagName::GetObjectSelect()**
   
   *Description:*
   This function returns the status that the object is selected.

10. **void ClgiTagName::SetObjectSelect(BOOL select)**
    
    *Description:*
    This function sets the status that the selection of the object.

11. **void ClgiTagName::SetPenColor(COLORREF currentColor)**
    
    *Description:*
    This function sets the color that the object is displayed.

12. **void ClgiTagName::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 iOrientation, int iNoStream)

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

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

Description:
This function returns TRUE 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::InvertGIPrimitive(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>
<thead>
<tr>
<th>Stream direction</th>
<th>Orientation value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point to east (→)</td>
<td>0</td>
</tr>
<tr>
<td>Point to north (↑)</td>
<td>1</td>
</tr>
<tr>
<td>Point to west (←)</td>
<td>2</td>
</tr>
<tr>
<td>Point to south (↓)</td>
<td>3</td>
</tr>
</tbody>
</table>

9. BOOL CIGiStream::GetObjectSelect()
Description:
This function returns the status that the object is selected.

10. void ClgiStream::SetObjectSelect(BOOL select)

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

11. int ClgiStream::GetStreamNumber()

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

12. void ClgiStream::SetStreamNumber(int iNoStream)

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

13. int ClgiStream::GetStreamType()

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

14. int ClgiStream::GetStreamDir()

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

15. int ClgiStream::GetStreamOrientation()

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

16. void ClgiStream::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

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

#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,
```
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  *pDIGITAGNAME; // pointer to tagname
    CigiStream   *pDIGISTREAM;   // pointer to stream
} ObjectPointer;

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

class CIGI_primitive : public CObject
{
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"

#if defined _DEBUG
#define new DEBUG_NEW
#endif

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

CIGIprimitive::~CIGIprimitive()
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 CgiBox();
            break;

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

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

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

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

        case IGI_TEXT:
            m_pObject.pIGITEXT = new CgiText();
            break;
        }
    }
}
case IGI_STREAM:
    m_pObject.pIGISTREAM = new CIGiStream();
    break;
    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;
}
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;
}

void CIGIprimitive::setParameter(int x[])
{
    for (int i = 0; i < 6; i++)
        m_iVar[i] = x[i];
}

void CIGIprimitive::CreateObject() {
    // 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);
    
    161
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;
}


void CIGIprimitive::SetPenColor(COLORREF currentColor)
{
    m_cPenColor = currentColor;
}


void CIGIprimitive::SetFillMode(BOOL bFillMode)
{
    m_bFill = bFillMode;
}

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
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();
        break;

    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.pIGITRIANGLE->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.pDIGITRIANGLE->ModifyPrimitive( dx, dy, index );
    break;

default : break;
} // end of switch

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.pDIGITRIANGLE->InvertIgiPrimitive(pDC, dx, dy);
        break;
    } // end of switch
break;

case IGI_TAGNAME:
    m_pObject.pIGITAGNAME->InvertIGiPrimitive(pDC, dx, dy);
    break;

case IGI_TEXT:
    m_pObject.pGITEXT->InvertIGiPrimitive(pDC, dx, dy);
    break;

case IGI_STREAM:
    m_pObject.pGISTREAM->InvertIGiPrimitive(pDC, dx, dy);
    break;

default:
    break;
} // end of switch

/////////End of switch\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n
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;
    } // end of switch

167
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.
 //
z
*********************************************************************************/
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.pITEMPLATETAGNAME->MovePrimitive( dx, dy );
    break;

case IGI_TEXT:
    m_pObject.pITYPENAME->MovePrimitive( dx, dy );
    break;

case IGI_STREAM:
    m_pObject.pITEMPLATESTREAM->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.pITYPENAME->SetPenColor(currentColor);
        break;

    case IGI_ELLIPSE:
        m_pObject.pITYPENAME->SetPenColor(currentColor);
        break;

    case IGI_LINE:
        m_pObject.pITYPENAME->SetPenColor(currentColor);
        break;

    case IGI_TRIANGLE:
        m_pObject.pITYPENAME->SetPenColor(currentColor);
        break;
    } // end of switch

169
case IGI_TAGNAME:
    m_pObject.pGITAGNAME->SetPenColor(currentColor);
    break;

case IGI_TEXT:
    m_pObject.pGITEXT->SetPenColor(currentColor);
    break;

default:
    break;
} // end of switch

Thông tin liên quan đến trạng thái của các phần tử được chọn.

BOOL CIGIprimitive::GetIGIObjectSelect()
{
    BOOL bSelect;
    switch ( m_iType )
    {
    case IGI_BOX:
        bSelect = m_pObject.pGIBOX->GetObjectSelect();
        break;

    case IGI_ELLIPSE:
        bSelect = m_pObject.pGIELLIPSE->GetObjectSelect();
        break;

    case IGI_LINE:
        bSelect = m_pObject.pGILINE->GetObjectSelect();
        break;

    case IGI_TRIANGLE:
        bSelect = m_pObject.pGITRIANGLE->GetObjectSelect();
        break;

    case IGI_TAGNAME:
        bSelect = m_pObject.pGITAGNAME->GetObjectSelect();
        break;
    } // end of switch

170
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.pIGIBILITY->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.pIGIEXT->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.pOBJBOX->GetPenColor();
    break;

case IGI_ELLIPSE:
    currentColor = m_pObject.pOBJELLIPSE->GetPenColor();
    break;

case IGI_LINE:
    currentColor = m_pObject.pOBJLINE->GetPenColor();
    break;

case IGI_TRIANGLE:
    currentColor = m_pObject.pOBJTRIANGLE->GetPenColor();
    break;

    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;


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;
}
}

void CIGILicense::SetStreamNumber(int iNoStream)
{
    n_pObject.pIGISTREAM->SetStreamNumber(iNoStream);
}

void CIGILicense::IncreaseOrientation()
{
    n_pObject.pIGISTREAM->IncreaseOrientation();
}

int CIGILicense::GetStreamType()
{
    return n_pObject.pIGISTREAM->GetStreamType();
}

int CIGILicense::GetStreamDir()
\{ 
    return m_pObject.pIGISTREAM-&gt;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-&gt;GetModifyPoint(iMarkIndex);
        break;

    case IGI_ELLIPSE:
        ptFrom = m_pObject.pIGIELLIPSE-&gt;GetModifyPoint(iMarkIndex);
        break;

    case IGI_LINE:
        ptFrom = m_pObject.pIGILINE-&gt;GetModifyPoint(iMarkIndex);
        break;

    case IGI_TRIANGLE:
        ptFrom = m_pObject.pIGITRIANGLE-&gt;GetModifyPoint(iMarkIndex, iFromIndex);
        break;

    default:
        break;
    }
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)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
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, OnPaletteFillColor)
ON_COMMAND(ID_PALETTE_TRIANGLE, OnPaletteTriangle)
ON_COMMAND(ID_PALETTE_FILLTRIANGLE, OnPaletteFillColor)
ON_COMMAND(ID_PALETTE_ELLIPSE, OnPaletteEllipse)
ON_COMMAND(ID_PALETTE_FILLELLIPSE, OnPaletteFillColor)
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_ICICUT_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(LPCSTR 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);
}

这几行代码是用于存储和加载对象的。

在`CIGeditorDoc`类中，定义了`SelectAll`命令。

这行代码用于在编辑区域中选择所有原始对象。

```cpp
void CIGeditorDoc::OnSelectAll()
{
    SetIGIOBJECTSELECTLIST(TRUE);
    UpdateAllViews(NULL);
}
```
void CIGeditorDoc::OnUnSelectAll()
{
    SetIGIOBJECTSelectList(FALSE);
    UpdateAllViews(NULL);
}

void CIGeditorDoc::OnDeleteAll()
{
    CreatePreviousList();      // Create a backup list of the primitives for purposes
                                 // of the undo command.
    DeleteContents();          // Mark the document as having been modified, for
    SetModifiedFlag();         // purposes of confirming file close.
    UpdateAllViews(NULL);
}

void CIGeditorDoc::OnDeleteSome()
{
    if (m_iCountSelect == 0)    // if there is no primitive selected, return.
        return;

    CreatePreviousList();      // for the 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
            {
                CIGIprimitive* pIGIprimitive = CopyIGIprimitive(pIGIprimitive,
m_primitiveList.AddTail(pIgiPrimitive);

}

}

UpdateAllViews(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 primitive 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;
   }
}
void CIGEditorDoc::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->GIObjectSelect() // the object is selected
        {
            m_primitiveList.InsertAfter(pos, pIGIprimitive);
            m_primitiveList.RemoveAt(posPop);
            UpdateAllViews(NULL);
            return;
        }
    }
}

void CIGeditorDoc::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 CIGeditorDoc::OnUpdateUnSelectAll(CCmdUI* pCmdUI)
{
    if (n_iCountSelect == 0)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

void CIGeditorDoc::OnUpdateDeleteAll(CCmdUI* pCmdUI)
{
    if (n_primitiveList.GetCount() == 0)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

void CIGeditorDoc::OnUpdateDeleteSome(CCmdUI* pCmdUI)
{
    if (n_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);
void C sildenafilDoc::OnUpdatePushBottom(CCmdUI* pCmdUI)
{
    if (m_iCountSelect != 1)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

void C sildenafilDoc::OnUpdatePopOneUp(CCmdUI* pCmdUI)
{
    if (m_iCountSelect != 1)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

void C sildenafilDoc::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 object's 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);
}
void CigieditorDoc::OnColorRed()
{
    CreatePreviousList();
    m_cCurrentColor = RGB(255, 0, 0);
    SetColorBar(RED);
    UpdateColor();
    UpdateAllViews(NULL);
}

void CigieditorDoc::OnColorBlue()
{
    CreatePreviousList();
    m_cCurrentColor = RGB(0, 0, 255);
    SetColorBar(BLUE);
    UpdateColor();
    UpdateAllViews(NULL);
}

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 CGieditorDoc::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 CGieditorDoc::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 CGieditorDoc::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 CGieditorDoc::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 CGieditorDoc::OnColorDarkgreen()
{
    CreatePreviousList();
    m_cCurrentColor = RGB(0, 128, 0);
SetColorBar(Darkgreen);
UpdateColor();
UpdateAllViews(NULL);
}

_tAColor m_cCurrentColor = RGB(0, 128, 128);
SetColorBar(Darkcyan);
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 CGieditorDoc::OnColorDarkmagenta()
{
    CreatePreviousList();
    m_cCurrentColor = RGB(128, 0, 128);
    SetColorBar(Darkmagenta);
    UpdateColor();
    UpdateAllViews(NULL);
}

// 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 CGieditorDoc::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 CGieditorDoc::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 CGieditorDoc::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 CGieditorDoc::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 CGieditorDoc::OnUpdateColorWhite( CCmdUI* pCmdUI )
{
    pCmdUI->SetCheck( m_iColorPalette[White] );
}

This command updates the darkblue color button on the color bar.
void CIGieditorDoc::OnUpdateColorDarkblue(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_iColorPalette[Darkblue]);
}

void CIGieditorDoc::OnUpdateColorDarkgreen(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_iColorPalette[Darkgreen]);
}

void CIGieditorDoc::OnUpdateColorDarkcyan(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_iColorPalette[Darkcyan]);
}

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.
 //
 202
void CGieditorDoc::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 CGieditorDoc::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 CGieditorDoc::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 CGiEditorDoc::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 CGiEditorDoc::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 CGiEditorDoc::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 CIGeditorDoc::OnPaletteFillEllipse()
{
    CreatePreviousList();
    SetPaletteBar(FillEllipse);
    m_iEditMode = ADD_PRIMITIVE_MODE;
    m_iCurrentPrimitive = IGI_PRIM_ELLIPSE;
    m_bFillMode = TRUE;
}

void CIGeditorDoc::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.

205
void CIGeditorDoc::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 CIGeditorDoc::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 CIGeditorDoc::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 CIGeditorDoc::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_PRIMITIVEMODE;
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_PRIMITIVEMODE;
    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.

209
void CIGieditorDoc::OnUpdatePaletteEllipse(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_iPrimitivePalette[IgiEllipse]);
}

void CIGieditorDoc::OnUpdatePaletteFillEllipse(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_iPrimitivePalette[FillEllipse]);
}

void CIGieditorDoc::OnUpdatePaletteText(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_iPrimitivePalette[IgiText]);
}

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 CImageEditorDoc::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 CImageEditorDoc::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 CImageEditorDoc::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 CGieditorDoc::OnUpdatePaletteLogicOut(CTCmdUI* 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 CGieditorDoc::OnUpdatePaletteAnalogIn(CTCmdUI* 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 CGieditorDoc::OnUpdatePaletteAnalogOut(CTCmdUI* 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 CGieditorDoc::OnUpdatePaletteLine(CTCmdUI* 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.

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.

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.
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;
    }
}

/***********************************************************************************/

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 gets the current color that user clicks on the color bar

COLORREF CQigieditorDoc::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 CQigieditorDoc::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 CQigieditorDoc::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;
}

financial
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.

financial

int CIGieditorDoc::GetSelectOrder()
{
    return m_iSelectOrder;
}

financial
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.

financial

void CIGieditorDoc::SetSelectOrder(int selectOrder)
{
    m_iSelectOrder = selectOrder;
}

financial
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. It is called by IgieeditorView class after user
selected the desired position on the edit area.

financial

CIGIprimitive* CIGieditorDoc::NewIGIPrimitive(PalettePrimitive item, int x[6])
{
    CIGIprimitive* pIGIPrimitive = new CIGIprimitive();  // create a new node of
    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.

220
void CIGieditorDoc::SetEditMode(EditModeIndex editMode)
{
    m_iEditMode = editMode;
}


void CIGieditorDoc::UpdateColor()
{
    POSITION pos = m_primitiveList.GetHeadPosition();
    while (pos != NULL)
    {
        CIGIprimitive* pIGIprimitive = mPrimitiveList.GetNext(pos);
        if (pIGIprimitive->GetIGIObjectSelect())
        {
            pIGIprimitive->UpdateColor(m_cCurrentColor);
        }
    }
    SetModifiedFlag(); // Mark the document as having been modified, for
                       // purposes of confirming File Close.
}

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 IGIPrimitive 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* pPrimitive = CopyIGIPrimitive(pIGIPrimitive, 0);  
    m_primitivePrevList.AddTail(pPrimitive);  
}
// 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  
// kept 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  

223
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;
}
int CIGieditorDoc::GetTagNameFlag()
{
    return m_iTagNameFlag;
}

void CIGieditorDoc::SetTagNameFlag(int tagNameFlag)
{
    m_iTagNameFlag = tagNameFlag;
}

CString CIGieditorDoc::GetTextInput()
{
    return m_cTextInput;
}

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 maximum 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.
//
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
CIPIprimitive* 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, CIPIprimitive *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"

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

#define StreamLength 18
#define StreamArrow 4

extern CStatusBar m_wndStatusBar;

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_LBUTTONDOWN()
    ON_WM_LBUTTONDOWN()
//}}AFX_MSG_MAP

229
ON_WM_RBUTTONDOWN ()
ON_WM_MOVE ()
ON_WM_DESTROY ()
ON_WM_SHOWWINDOW ()
ON_WM_LBUTTONDOWN ()
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
 // CIGIPrimitive::DrawIGIPrimitive (). It is called when the view window
 // updated.

 void CIGieditorView::OnDraw (CDC * pDC)
 {

CIGeditorDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
// TODO: add draw code for native data here

// The view delegates the drawing of individual primitive to
// CIGIPrimitive::DrawIGIPrimitive().
pDoc->DrawIGIPrimitiveList(pDC);
}

////////////////////////////////////////////////////////////////////
// CIGeditorView diagnostics

#ifdef _DEBUG
void CIGeditorView::AssertValid() const
{
    CView::AssertValid();
}

void CIGeditorView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CIGeditorDoc* CIGeditorView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CIGeditorDoc)));
    return (CIGeditorDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////////////////////////////////////
// This function is CIGeditorView 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 CIGeditorView::PreTranslateMessage(MSG* pMsg)
{
    CIGeditorDoc* 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_RBUTTONDOWN)
{
    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_LBUTTONDOWN && 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);
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_hFill)
  {
    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);
  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
        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
        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
        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;

pDoc->NewIgiPrimitive(m_iItem, x);
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,
    CLIP_CHARACTER_PRECIS,
    FF_DONTCARE,"Arial");
    dc.SelectObject( &m_fontMain );
    CString cString = "xxxxxxxxxxxxxxxxx";
    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 (LOGPIXELS) *8) /
    78);

    m_fontMain.CreateFont(nHeight, 0, 0, 0, FW_NORMAL,
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        DEFAULT_QUALITY, DEFAULT_PITCH |
        CLIP_CHARACTER_PRECIS,
        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);

another one, right button to end add stream". TRUE);
InvalidOperationException(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);

240
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);

another one, right button to end add stream

m_wndStatusBar.SetPaneText( 0, "left button for

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);

another one, right button to end add stream

m_wndStatusBar.SetPaneText( 0, "left button for

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_TEXT:
                InvertText (

            case IGI_PRIM_TRIANGLE:
                break;
            case IGI_PRIM_ARROW:
                break;
            }
        }
    }

243
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 CGieditorView::OnCaptureChanged(CWnd *pwnd)
{
    // TODO: Add your message handler code here

    CView::OnCaptureChanged(pwnd);
}

void CGieditorView::OnContextMenu(CWnd* pwnd, CPoint point)
{
    // TODO: Add your message handler code here
}
void CigeditorView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default

    CView::OnChar(nChar, nRepCnt, nFlags);
}

void CigeditorView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default
    CigeditorDoc* pDoc = GetDocument();
    ASSERT_VALID(pDX);
    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); // Refresh window background color

void CgieditorView::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default
    Cview::OnKeyUp(nChar, nRepCnt, nFlags);
}

void CgieditorView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    Cview::OnLButtonDown(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 CgieditorView::OnLButtonDBLCLK(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CClientDC dc (this);

    CGieditorDoc* 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->ObjectType)
} // if (SelectMark)

CView::OnLButtonDown(nFlags, point);

void CView::OnMove(int x, int y)
{
    CView::OnMove(x, y);

    // TODO: Add your message handler code here
}
void CIGeditorView::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 CIGeditorView::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); // Refresh 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); // Refresh window background color
        }
        m_bInMove = FALSE;
        m_bInModify = FALSE;
    }
    else if (m_pCurrent == NULL) // unselect all objects if mouse is not clicked on the object
    {
        CClientDC dc (this);
        GetDocument() -> SetIGIOBJECTSelectList(FALSE);
    }
InvalidRect(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 CVisualEditorView::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 CVisualEditorView::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);
            }
    }
}

249
}  
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::OnEraseBkgnDCDC* pDC)
{
    // TODO: Add your message handler code here and/or call default
    return CView::OnEraseBkgnDCDC* 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;

250
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 CGieditorView::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;

CGieditorDoc* 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) {

```cpp
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"

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

// Construction/Destruction

CIgiBox::CIgiBox()
{
    m_bObjectSelect = FALSE;
}

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()
{
}

 wereld

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 corner
  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 CgiBox::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 corner
    x1 = x1 + dx;    // lowleft corner
    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 CGiBox::DrawPrimitive(CDC *pDC)
{
    if (m_bObjectSelect)  // draw select mark
    {
        CGiDrawMark 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::InvertGigiPrimitive(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 CGiBox::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;
    }
}

COLORREF CGiBox::GetPenColor()
{
    return m_cPenColor;
}

BOOL CGiBox::GetObjectSelect()
{
    return m_bObjectSelect;
}
This function sets the fill mode for the object.

```cpp
void CgBox::setFillMode(BOOL bFillMode)
{
    m_bFill = bFillMode;
}
```

This function gets the fill mode for the object.

```cpp
BOOL CgBox::getFillMode()
{
    return m_bFill;
}
```

This function sets the color that the object is displayed.

```cpp
COLORREF currentColor;
```

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

```cpp
m_objectSel = select;
```
CPoint CgiBox::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;
}

void CgiBox::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;
  //}
  
  //int CIGiBox::GetMarkIndex()
  //{
  //  return m_iMarkIndex;
  //}