Praise for Clayberg and Rubel’s *Eclipse Plug-ins, Third Edition*

“Dan Rubel and Eric Clayberg are the authors of one of the most highly regarded books in the history of Eclipse. Their *Eclipse Plug-ins* is generally considered the seminal book on how to extend the Eclipse platform.”

— *Mike Milinkovich*
Executive Director, Eclipse Foundation

“I’m often asked, ‘What are the best books about Eclipse?’ Number one on my list, every time, is *Eclipse Plug-ins*. I find it to be the clearest and most relevant book about Eclipse for the real-world software developer. Other Eclipse books focus on the internal Eclipse architecture or on repeating the Eclipse documentation, whereas this book is laser focused on the issues and concepts that matter when you’re trying to build a product.”

— *Bjorn Freeman-Benson*
Former Director, Open Source Process, Eclipse Foundation

“As the title suggests, this massive tome is intended as a guide to best practices for writing Eclipse plug-ins. I think in that respect it succeeds handily. Before you even think about distributing a plug-in you’ve written, read this book.”

— *Ernest Friedman-Hill*
Marshall, JavaRanch.com

“If you’re looking for just one Eclipse plug-in development book that will be your guide, this is the one. While there are other books available on Eclipse, few dive as deep as *Eclipse Plug-ins*.”

— *Simon Archer*

“*Eclipse Plug-ins* was an invaluable training aid for all of our team members. In fact, training our team without the use of this book as a base would have been virtually impossible. It is now required reading for all our developers and helped us deliver a brand-new, very complex product on time and on budget thanks to the great job this book does of explaining the process of building plug-ins for Eclipse.”

— *Bruce Gruenbaum*

“The authors of this seminal book have decades of proven experience with the most productive and robust software engineering technologies ever developed. Their experiences have now been well applied to the use of Eclipse for more effective Java development. A must-have for any serious software engineering professional!”

— *Ed Klimas*
“This is easily one of the most useful books I own. If you are new to developing Eclipse plug-ins, it is a ‘must-have’ that will save you lots of time and effort. You will find lots of good advice in here, especially things that will help add a whole layer of professionalism and completeness to any plug-in. The book is very focused, well-structured, thorough, clearly written, and doesn’t contain a single page of ‘waffly page filler.’ The diagrams explaining the relationships between the different components and manifest sections are excellent and aid in understanding how everything fits together. This book goes well beyond Actions, Views, and Editors, and I think everyone will benefit from the authors’ experience. I certainly have.”

— Tony Saveski

“Just wanted to also let you know this is an excellent book! Thanks for putting forth the effort to create a book that is easy to read and technical at the same time!”

— Brooke Hedrick

“The key to developing great plug-ins for Eclipse is understanding where and how to extend the IDE, and that’s what this book gives you. It is a must for serious plug-in developers, especially those building commercial applications. I wouldn’t be without it.”

— Brian Wilkerson
The Eclipse Graphical Editing Framework (GEF)
Eclipse is a universal, multilanguage software development environment—an open, extensible, integrated development environment (IDE)—that can be used for anything. Eclipse represents one of the most exciting initiatives to come from the world of application development, and it has the support of leading companies and organizations in the technology sector. Eclipse is gaining widespread acceptance in both commercial and academic arenas.

The Eclipse Series is the definitive collection of publications dedicated to the Eclipse platform. Books in this series bring you key technical information, critical insight, and the practical advice you need to build tools to support this revolutionary open-source platform.
The Eclipse Graphical Editing Framework (GEF)

Dan Rubel
Jaime Wren
Eric Clayberg
To the women we love,
Kathy, Helene, and Karen
Contents

Foreword by Mike Milinkovich xix

Preface xxi

Chapter 1  What Is GEF? 1
  1.1  GEF Overview 1

  1.2  GEF Example Applications  2
    1.2.1  Shapes Example  2
    1.2.2  Flow Example  3
    1.2.3  Logic Example  4
    1.2.4  Text Example  4
    1.2.5  XMind  5
    1.2.6  WindowBuilder  5

  1.3  Summary 6

Chapter 2  A Simple Draw2D Example 7
  2.1  Draw2D Installation 7
  2.2  Draw2D Project 8
  2.3  Draw2D Application 9
## Contents

2.4 Draw2D View  15  
2.5 Draw2D Events  17  
2.6 Book Samples  20  
2.7 Summary  20  

Chapter 3 Draw2D Infrastructure  21  
3.1 Architecture  21  
3.2 Drawing  23  
3.3 Processing Events  24  
3.4 Summary  25  

Chapter 4 Figures  27  
4.1 IFigure  27  
4.2 Common Figures  29  
4.3 Custom Figures  33  
  4.3.1 Extending Existing Figures  33  
  4.3.2 Adding Nested Figures  35  
4.4 Painting  37  
  4.4.1 Bounds and Client Area  37  
  4.4.2 Paint Methods  38  
  4.4.3 Graphics  39  
  4.4.4 Z-Order  40  
  4.4.5 Clipping  40  
  4.4.6 Custom Painting  41  
4.5 Borders  42  
  4.5.1 Common Borders  43  
  4.5.2 Custom Borders  45  
4.6 Summary  53
Chapter 5  Layout Managers  55
  5.1  List Constraints  55
  5.2  Minimum, Maximum, and Preferred Size  56
  5.3  Common Layout Managers  57
    5.3.1  BorderLayout  57
    5.3.2  DelegatingLayout  58
    5.3.3  FlowLayout  59
    5.3.4  GridLayout  60
    5.3.5  StackLayout  61
    5.3.6  ToolbarLayout  62
    5.3.7  XYLayout  63
  5.4  Using Layout Managers  63
  5.5  Summary  67

Chapter 6  Connections  69
  6.1  Common Anchors  70
    6.1.1  ChopboxAnchor  70
    6.1.2  EllipseAnchor  71
    6.1.3  LabelAnchor  71
    6.1.4  XYAnchor  71
  6.2  Custom Anchors  72
    6.2.1  CenterAnchor  72
    6.2.2  MarriageAnchor  73
  6.3  Decorations  76
    6.3.1  Default Decorations  77
    6.3.2  Custom Decorations  78
6.4 Routing Connections  80
  6.4.1 BendpointConnectionRouter  81
  6.4.2 FanRouter  84
  6.4.3 ManhattanConnectionRouter  85
  6.4.4 NullConnectionRouter  85
  6.4.5 ShortestPathConnectionRouter  85

6.5 Connection Labels  86
  6.5.1 BendpointLocator  87
  6.5.2 ConnectionEndpointLocator  88
  6.5.3 ConnectionLocator  88
  6.5.4 MidpointLocator  89

6.6 Summary  90

Chapter 7 Layers and Viewports 91

7.1 Layers  91
  7.1.1 LayeredPane  92
  7.1.2 ConnectionLayer  93
  7.1.3 Hit Testing  95

7.2 Scrolling  96
  7.2.1 FigureCanvas  97
  7.2.2 Viewport  98
  7.2.3 FreeformFigure  98
  7.2.4 FreeformLayer  99
  7.2.5 FreeformLayeredPane  100
  7.2.6 FreeformViewport  100

7.3 Coordinates  101
7.4 Scaling 104
  7.4.1 ScalableFigure 104
  7.4.2 ScalableFreeformLayeredPane 104
  7.4.3 Zoom Menu 105
  7.4.4 Scaling Dimensions 107
  7.4.5 PrecisionPoint and PrecisionDimension 109

7.5 Summary 112

Chapter 8 GEF Models 113

  8.1 Genealogy Model 113
     8.1.1 Domain Information versus Presentation Information 115
     8.1.2 Listeners 115

  8.2 Populating the Diagram 116
     8.2.1 Reading the Model 116
     8.2.2 Hooking Model to Diagram 118
     8.2.3 Hooking Diagram to Model 124
     8.2.4 Reading from a File 125

  8.3 Storing the Diagram 126
     8.3.1 Serializing Model Information 126
     8.3.2 Writing to a File 127

  8.4 Summary 128

Chapter 9 Zest 129

  9.1 Setup 129
     9.1.1 Installation 129
     9.1.2 Plug-in Dependencies 130
     9.1.3 Creating GenealogyZestView 130

  9.2 GraphViewer 131
10.2 EditPartViewer 178
   10.2.1 EditPartFactory 179
   10.2.2 RootEditPart 179
   10.2.3 EditPartViewer setContents 180
   10.2.4 EditDomain 180

10.3 Tools, Actions, Policies, Requests, and Commands 180
   10.3.1 Tools 181
   10.3.2 Actions 182
   10.3.3 Requests 182
   10.3.4 EditPolicy 182
   10.3.5 Commands 183

10.4 Summary 183

Chapter 11 GEF View 185

11.1 Setup 185
   11.1.1 Installation 185
   11.1.2 Plug-in Dependencies 185

11.2 GEF Viewer 186
   11.2.1 Standalone GEF View 187
   11.2.2 Viewer setContents 187

11.3 EditPartFactory 188
   11.3.1 GenealogyGraphEditPart 188
   11.3.2 PersonEditPart 189

11.4 Connections 193

11.5 Summary 200
Chapter 12  GEF Editor  201

12.1  Setup  201

12.2  GenealogyGraphEditor  201
  12.2.1  Reading and Displaying the Model  203
  12.2.2  Saving the Model  205

12.3  Selection  207
  12.3.1  Making the Selection Visible  207
  12.3.2  Selection EditPolicy  209
  12.3.3  SelectionChangeListener  212
  12.3.4  SelectionManager  214
  12.3.5  Synchronizing the Selection in Multiple Editors  217
  12.3.6  Accessibility  217

12.4  Summary  218

Chapter 13  Commands and Tools  219

13.1  Listening for Model Changes  219
  13.1.1  Adding and Removing EditParts  220
  13.1.2  Updating Figures  221
  13.1.3  Updating Connections  223
  13.1.4  Adding and Removing Nested EditParts  226

13.2  Commands  226
  13.2.1  Create Command  227
  13.2.2  Move and Resize Command  228
  13.2.3  Reorder Command  229
  13.2.4  Reparent Command  230
  13.2.5  Delete Command  231
  13.2.6  Composite Commands  232
13.3 EditPolicies  233
  13.3.1 Creating Components  233
  13.3.2 Moving and Resizing Components  235
  13.3.3 Reordering Components  236
  13.3.4 Reparenting Components  238
  13.3.5 Deleting Components  240
  13.3.6 Creating Connections  240
  13.3.7 Modifying Connections  244
  13.3.8 Deleting Connections  247
  13.3.9 Deleting the Graph  248

13.4 Global Edit Menu Actions  248

13.5 Palette and Tools  249
  13.5.1 Palette Creation  250
  13.5.2 Selection Tools  250
  13.5.3 Component Creation Tools  251
  13.5.4 Connection Creation Tools  252
  13.5.5 Creation Drag and Drop  252

13.6 Summary  253

Index  255
Foreword

The Eclipse Graphical Editor Framework (GEF) project supports the creation of rich graphical editors and views for Eclipse-based tools and Rich Client Platform (RCP) applications. GEF’s three frameworks—Draw2D, Zest, and GEF—are amongst the most widely used within the Eclipse community and ecosystem.

“Mighty oaks from little acorns grow” is the story of the GEF project. In the context of the Eclipse community, GEF is a relatively small project. But the tools, applications, and products that have been enabled by GEF form a very long list indeed. Everything from mission planning for the Mars Rovers to most of the world’s commercial modeling tools make use of GEF. GEF is also widely re-used within the Eclipse community itself, and is leveraged by Eclipse projects such as GMF, Graphiti, AMP, Sphinx and Papyrus. It is a testament to the idea that a small, powerful, and open source framework can make an enormous impact on the industry.

A big part of the success of the GEF project and its three frameworks has been its long-term focus on being a platform. Although there has been a steady flow of innovative new features, the quality, stability, and backwards compatibility of the GEF project APIs have been a big part of its success. That level of commitment to the “platformness” (to coin a phrase) of a framework is the hallmark of a great project at Eclipse. It requires a great deal of commitment and discipline by the project team to accomplish.

Eclipse projects are powered by people, so I would like to recognize the contributions of the present GEF project leader Anthony Hunter, and the past leaders Randy Hudson and Steven Shaw, all of IBM. I would also like to recognize the many contributions of the projects committers past and present: Nick Boldt, Alex Boyko, Ian Bull, Marc Gobeil, Alexander Nyssen, Cherie
Revells, Pratik Shah, and Fabian Steeg. I would also like to recognize the contributions and investments of IBM, itemis AG, EclipseSource, and Tasktop in supporting the team working on GEF.

Dan Rubel and Eric Clayberg are the authors of one of the most highly regarded books in the history of Eclipse. Their *Eclipse Plug-ins* is generally considered the seminal book on how to extend the Eclipse platform. Dan and Eric, this time joined by their colleague Jaime Wren, have brought their clear prose, deep knowledge, and focus on the issues that matter to developers using the Eclipse GEF framework to this new book. I know that you will find it a useful addition to your Eclipse library.

—Mike Milinkovich  
Executive Director  
Eclipse Foundation, Inc.
When we were first exposed to Eclipse back in late 1999, we were struck by the magnitude of the problem IBM was trying to solve. IBM wanted to unify all its development environments on a single code base. At the time, the company was using a mix of technology composed of a hodgepodge of C/C++, Java, and Smalltalk.

Many of IBM’s most important tools, including the award-winning VisualAge for Java IDE, were actually written in Smalltalk—a wonderful language for building sophisticated tools, but one that was rapidly losing market share to languages like Java. While IBM had one of the world’s largest collections of Smalltalk developers, there wasn’t a great deal of industry support for it outside of IBM, and very few independent software vendors (ISVs) were qualified to create Smalltalk-based add-ons.

Meanwhile, Java was winning the hearts and minds of developers worldwide with its promise of easy portability across a wide range of platforms, while providing the rich application programming interface (API) needed to build the latest generation of Web-based business applications. More important, Java was an object-oriented (OO) language, which meant that IBM could leverage the large body of highly skilled object-oriented developers it had built up over the years of creating Smalltalk-based tools. In fact, IBM took its premier Object Technology International (OTI) group, which had been responsible for creating IBM’s VisualAge Smalltalk and VisualAge Java environments (VisualAge Smalltalk was the first of the VisualAge brand family, and VisualAge Java was built using it), and tasked the group with creating
a highly extensible integrated development environment (IDE) construction set based in Java. Eclipse was the happy result.

OTI was able to apply its highly evolved OO skills to produce an IDE unmatched in power, flexibility, and extensibility. The group was able to replicate most of the features that had made Smalltalk-based IDEs so popular the decade before, while simultaneously pushing the state of the art in IDE development ahead by an order of magnitude.

The Java world had never seen anything as powerful or as compelling as Eclipse, and it now stands, with Microsoft’s .NET, as one of the world’s premier development environments. That alone makes Eclipse a perfect platform for developers wishing to get their tools out to as wide an audience as possible. The fact that Eclipse is completely free and open source is icing on the cake. An open, extensible IDE base that is available for free to anyone with a computer is a powerful motivator to the prospective tool developer.

It certainly was to us. At Instantiations and earlier at ObjectShare, we had spent the better part of a decade as entrepreneurs focused on building add-on tools for various IDEs. We had started with building add-ons for Digitalk’s Smalltalk/V, migrated to developing tools for IBM’s VisualAge Smalltalk, and eventually ended up creating tools for IBM’s VisualAge Java (including our award-winning VA Assist product and our jFactor product, one of the world’s first Java refactoring tools). Every one of these environments provided a means to extend the IDE, but they were generally not well documented and certainly not standardized in any way. Small market shares (relative to tools such as VisualBasic) and an eclectic user base also afflicted these environments and, by extension, us.

As an Advanced IBM Business Partner, we were fortunate to have built a long and trusting relationship with the folks at IBM responsible for the creation of Eclipse. That relationship meant that we were in a unique position to be briefed on the technology and to start using it on a daily basis nearly a year-and-a-half before the rest of the world even heard about it. When IBM finally announced Eclipse to the world in mid-2001, our team at Instantiations had built some of the first demo applications IBM had to show. Later that year, when IBM released its first Eclipse-based commercial tool, WebSphere Studio Application Developer v4.0 (v4.0 so that it synchronized with its then-current VisualAge for Java v4.0), our CodePro product became the very first commercial add-on available for it (and for Eclipse in general) on the same day. Two years later, we introduced our first GEF-based tool, WindowBuilder Pro, a powerful graphical user interface (GUI) development tool.

Developing WindowBuilder over the last several years has provided us with an opportunity to learn the details of Eclipse GEF development at a level matched by very few others. WindowBuilder has also served as a testbed for
many of the ideas and techniques presented in this book, providing us with a
unique perspective from which to write.

WindowBuilder’s product suite (especially GWT Designer) caught the
attention of Google, which acquired Instantiations in August of 2010. Since
the acquisition Google has donated the WindowBuilder architecture and the
two projects, SWT Designer and Swing Designer, to the Eclipse Foundation.
The GWT Designer product has been folded into the Google Plug-in for
Eclipse (GPE).

Goals of the Book

This book provides an in-depth description of the process involved in building
Eclipse GEF-based tools and editors. This book has several complementary
goals:

- To provide a quick introduction to GEF for new users
- To provide a reference for experienced Eclipse GEF users wishing to
  expand their knowledge and improve the quality of their GEF-based
  products
- To provide a detailed tutorial on creating sophisticated GEF tools suit-
  able for new and experienced users

The first chapter introduces GEF, Draw2D, and Zest and includes exam-
pies of what has been built using GEF. The next two chapters outline the pro-
cess of building a simple Draw2D example. The intention of these chapters is
to help developers new to GEF quickly understand and pull together an exam-
ple they can use to experiment with.

The next five chapters progressively introduce the reader to more and
more of the Draw2D framework that forms the foundation of GEF. The
fourth chapter introduces figures, which are the building blocks for the rest of
the book. Chapters 5 through 8 bring the user through the complete Draw2D
Genealogy example, introducing concepts such as layout managers, connec-
tions, layers, and viewports.

The ninth chapter presents Zest, a graph visualization project part of GEF.
The remaining chapters present the non-Draw2D portions of the GEF
project, including EditParts, EditPolicies, tools, commands, and actions.
These chapters walk the user through the development of a GEF Editor for a
genealogy model.

Each chapter focuses on a different aspect of the topic and includes an
overview, a detailed description, a discussion of challenges and solutions, dia-
grams, screenshots, cookbook-style code examples, relevant API listings, and
a summary.
Sometimes a developer needs a quick solution, while at other times that same developer needs to gain in-depth knowledge about a particular aspect of development. The intent is to provide several different ways for the reader to absorb and use the information so that both needs can be addressed. Relevant APIs are included in several of the chapters so that the book can be used as a standalone reference during development without requiring the reader to look up those APIs in the IDE. Most API descriptions are copied or paraphrased from the Eclipse platform Javadoc.

The examples provided in the chapters describe building various aspects of a concrete Eclipse GEF-based plug-in that will evolve over the course of the book. When you use the book as a reference rather than read it cover to cover, you will typically start to look in one chapter for issues that are covered in another. To facilitate this type of searching, every chapter contains numerous cross-references to related material that appears in other chapters.

**Intended Audience**

The audience for this book includes Java tool developers wishing to build graphical editing products that integrate with Eclipse and other Eclipse-based products, relatively advanced Eclipse users wishing to build their own graphical tools, or anyone who is curious about what makes Eclipse GEF tick. You should be a moderately experienced Eclipse developer to take full advantage of this book. If you are new to Eclipse or Eclipse plug-in development, we recommend starting with our companion book, *Eclipse Plug-ins*. We also anticipate that the reader is a fairly seasoned developer with a good grasp of Java and at least a cursory knowledge of extensible markup language (XML).

**Conventions Used in This Book**

The following formatting conventions are used throughout the book.

- **Bold**—the names of UI elements such as menus, buttons, field labels, tabs, and window titles
- **Italic**—emphasize new terms
- **Courier**—code examples, references to class and method names, and filenames
- **Courier Bold**—emphasize code fragments
- “**Quoted text**”—indicates words to be entered by the user
Acknowledgments

The authors would like to thank all those who have had a hand in putting this book together or who gave us their support and encouragement throughout the many months it took to create.

To our comrades at Instantiations and Google, who gave us the time and encouragement to work on this book: Rick Abbott, Brad Abrams, Brent Caldwell, Devon Carew, David Carlson, David Chandler, Jim Christensen, Taylor Corey, Rajeev Dayal, Dianne Engles, Marta George, Nick Gilman, Seth Hollyman, Alex Humesky, Bruce Johnson, Mark Johnson, Ed Klimas, Tina Kvavle, Florin Malita, Warren Martin, Miguel Méndez, Steve Messick, Alexander Mitin, Gina Nebling, John O’Keefe, Keerti Parthasarathy, Phil Quitslund, Chris Ramsdale, Mark Russell, Rob Ryan, Andrey Sablin, Konstantin Scheglov, Chuck Shawan, Bryan Shepherd, Julie Taylor, Mike Taylor, Solveig Viste, Andrew Wegley, and Brian Wilkerson.

To our editor, Greg Doench, our production editor, Elizabeth Ryan, our copy editor, Barbara Wood, our editorial assistant, Michelle Housley, our marketing manager, Stephane Nakib, and the staff at Pearson, for their encouragement and tremendous efforts in preparing this book for production.

To the series editors, Erich Gamma, Lee Nackman, and John Wiegand, for their thoughtful comments and for their ongoing efforts to make Eclipse the best development environment in the world.

We would also like to thank our wives, Kathy, Helene, and Karen, for their endless patience, and our children, Beth, Lauren, Lee, and David, for their endless inspiration.

About the Authors

Dan Rubel is Senior Software Engineer for Google. He is an entrepreneur and an expert in the design and application of OO technologies with more than 17 years of commercial software development experience, including 15 years of experience with Java and 11 years with Eclipse. He is the architect and product manager for several successful commercial products, including RCP Developer, Window-Tester, jFactor, and jKit, and has played key design and leadership roles in other commercial products such as VA Assist, and CodePro. He has a B.S. from Bucknell and was a cofounder of Instantiations.
Jaime Wren is Software Engineer for Google. He has worked with object-oriented technologies for the last nine years, and Eclipse tools for the past six years, gaining extensive expertise in developing commercial Eclipse-based tools. At Instantiations, Jaime made significant contributions as a developer on the CodePro and WindowBuilder product lines. After the acquisition of Instantiations by Google, he continues to work on the WindowBuilder product on the Google Web Toolkit (GWT) team. Jaime holds a double B.S. in Mathematics and Computer Science from the University of Oregon.

Eric Clayberg is Software Engineering Manager for Google. Eric is a seasoned software technologist, product developer, entrepreneur, and manager with more than 19 years of commercial software development experience, including 14 years of experience with Java and 11 years with Eclipse. He is the primary author and architect of more than a dozen commercial Java and Smalltalk add-on products, including the popular WindowBuilder, CodePro, and the award-winning VA Assist product lines. He has a B.S. from MIT, and an M.B.A. from Harvard, and has cofounded two successful software companies—ObjectShare and Instantiations.

Google is a multinational public corporation invested in Internet search, cloud computing, and advertising technologies. Google hosts and develops a number of Internet-based services and products, and its mission statement from the beginning has been “to organize the world’s information and make it universally accessible and useful.”

How to Contact Us

While we have made every effort to make sure that the material in this book is timely and accurate, Eclipse is a rapidly moving target, and it is quite possible that you may encounter differences between what we present here and what you experience using Eclipse. The Eclipse UI has evolved considerably over the years, and the latest 3.7 release is no exception. While we have targeted it at Eclipse 3.7 and used it for all of our examples, this book was completed after Eclipse 3.6 was finished and during the final phases of development of Eclipse 3.7. If you are using an older or newer version of Eclipse, this means that you may encounter various views, dialogs, and wizards that are subtly different from the screenshots herein.
• Questions about the book’s technical content should be addressed to info@qualityeclipse.com
• Sales questions should be addressed to Addison-Wesley at www.informit.com/store/sales.aspx
• Source code for the projects presented can be found at www.qualityeclipse.com/projects
• Errata can be found at www.qualityeclipse.com/errata
• Tools used and described can be found at www.qualityeclipse.com/tools
CHAPTER 2

A Simple Draw2D Example

Before covering the Draw2D infrastructure (see Chapter 3 on page 21) and each area of building a Draw2D diagram in depth, it is useful to create a simple example on which discussion can be based. This chapter takes a step-by-step approach to creating a simple Draw2D diagram representing the relationship between two people and their offspring. To start, we take an unsophisticated “brute force” approach, which we will refactor and refine in later chapters as we introduce more concepts. This process provides valuable first-hand experience using the Draw2D API.

2.1 Draw2D Installation

Select the Help > Install New Software... menu to install the GEF framework into Eclipse. When the Install wizard opens, select the Eclipse release update site (e.g., Indigo - http://download.eclipse.org/releases/indigo). Once the wizard refreshes, expand the Modeling category and select Graphical Editing Framework GEF SDK (see Figure 2–1). Alternatively, if you would like to install a different version of GEF, enter the GEF specific update site (http://download.eclipse.org/tools/gef/releases/releases) in the Install wizard and select the GEF features you wish to install. After you click Finish and restart Eclipse, the GEF framework installation is complete.
2.2 Draw2D Project

The full Eclipse RCP framework is not needed to use the Draw2D framework, so if you are creating a simple Java application, you can create a simple Java project in Eclipse and modify its build path to include the following Eclipse JAR files:

- ECLIPSE_HOME/plugins/
  - org.eclipse.swt_3.7.X.vXXXX.jar
- ECLIPSE_HOME/plugins/
  - org.eclipse.swt.win32.win32.x86_3.7.X.vXXXX.jar
- ECLIPSE_HOME/plugins/
  - org.eclipse.draw2d_3.7.X.vXXXX.jar

![Figure 2-1 Install wizard.](image-url)
Alternatively, if you are creating a diagram as part of a larger Eclipse RCP application, then create a Plug-in project with the following plug-in dependencies (see Chapter 2 in the *Eclipse Plug-ins* book for more about Plug-in projects):

- org.eclipse.ui
- org.eclipse.core.runtime
- org.eclipse.draw2d

Since the second half of this book describes techniques that require the Eclipse RCP framework, we use a Plug-in project rather than a simple Java project for all of the samples in this book.

### 2.3 Draw2D Application

Since the full Eclipse RCP framework is not needed to use the Draw2D framework, we create a simple Java class containing a `main(...)` method.

```java
class GenealogyView {
    public static void main(String[] args) {
        new GenealogyView().run();
    }
}
```

**Tip:** All of this source can be downloaded from www.qualityeclipse.com.

The `main(...)` method calls a `run()` method to initialize the shell, create the diagram, and show the shell. The `run()` method is not interesting with respect to Draw2D and is included here only for completeness. For more information on SWT and shells, please see the *Eclipse Plug-ins* book.
private void run() {
    Shell shell = new Shell(new Display());
    shell.setSize(365, 280);
    shell.setText("Genealogy");
    shell.setLayout(new GridLayout());

    Canvas canvas = createDiagram(shell);
    canvas.setLayoutData(new GridData(GridData.FILL_BOTH));

    Display display = shell.getDisplay();
    shell.open();
    while (!shell.isDisposed()) {
        while (!display.readAndDispatch()) {
            display.sleep();
        }
    }
}

The run() method calls the createDiagram(...) method to create and populate the diagram. This method creates a root figure to contain all of the other figures in the diagram (see Chapter 4 on page 27 for more about figures). A simple layout manager (see Chapter 5 on page 55 for more about layout managers) is used to statically lay out the figures that are added later in this section. Finally, the last bit of code creates a Canvas on which the diagram is displayed and a LightweightSystem used to display the diagram (see Section 3.1 on page 21 for more about LightweightSystem).

private Canvas createDiagram(Composite parent) {

    // Create a root figure and simple layout to contain
    // all other figures
    Figure root = new Figure();
    root.setFont(parent.getFont());
    XYLayout layout = new XYLayout();
    root.setLayoutManager(layout);

    // Create a canvas to display the root figure
    Canvas canvas = new Canvas(parent, SWT.DOUBLE_BUFFERED);
    canvas.setBackground(ColorConstants.white);
    LightweightSystem lws = new LightweightSystem(canvas);
    lws.setContents(root);
    return canvas;
}

Tip: Always set the font for the root figure

    root.setFont(parent.getFont());

so that each Label’s preferred size will be correctly calculated.
If you run the `main(...)` method, an empty window will appear (see Figure 2–2).

![Empty Genealogy window.](image)

Next, we want to add figures to the diagram representing a man, a woman, and their one child. Add the following to the `createDiagram(...)` method so that these figures are created and displayed.

```java
private Canvas createDiagram(Composite parent) {
    // Add the father "Andy"
    IFigure andy = createPersonFigure("Andy");
    root.add(andy);
    layout.setConstraint(andy,
        new Rectangle(new Point(10, 10), andy.getPreferredSize()));

    // Add the mother "Betty"
    IFigure betty = createPersonFigure("Betty");
    root.add(betty);
    layout.setConstraint(betty,
        new Rectangle(new Point(230, 10), betty.getPreferredSize()));

    // Add the son "Carl"
    IFigure carl = createPersonFigure("Carl");
    root.add(carl);
    layout.setConstraint(carl,
        new Rectangle(new Point(120, 120), carl.getPreferredSize()));

    // ... existing code here ...
}
```
The createDiagram(...) method now calls a new createPersonFigure(...) method to do the work of instantiating and initializing the figure representing a person. This person figure contains a nested Label figure to display the person’s name (see Section 4.3.2 on page 35 for more on nested figures).

```java
private IFigure createPersonFigure(String name) {
    RectangleFigure rectangleFigure = new RectangleFigure();
    rectangleFigure.setBackgroundColor(ColorConstants.lightGray);
    rectangleFigure.setLayoutManager(new ToolbarLayout());
    rectangleFigure.setPreferredSize(100, 100);
    rectangleFigure.add(new Label(name));
    return rectangleFigure;
}
```

**Tip:** Rather than adding the figure and then separately setting the layout constraint:

```java
    root.add(andy);
    layout.setConstraint(andy,
        new Rectangle(new Point(10, 10),
        andy.getPreferredSize()));
```

combine this into a single statement using the IFigure.add(child, constraint) method:

```java
    root.add(andy,
        new Rectangle(new Point(10, 10),
        andy.getPreferredSize()));
```

Refactor the createDiagram(...) method above to use this more compact form, and inline the layout as we do not need to refer to it.

```java
    root.setLayoutManager(new XYLayout());
```

The createDiagram(...) method now calls a new createPersonFigure(...) method to do the work of instantiating and initializing the figure representing a person. This person figure contains a nested Label figure to display the person’s name (see Section 4.3.2 on page 35 for more on nested figures).
Now, when the `main(...) method is run, the following window appears (see Figure 2–3).

![Genealogy window with three people.](image)

**Figure 2–3** Genealogy window with three people.

Next, add more code to the `createDiagram(...) method to create a “marriage” figure representing the relationship among the three people. This additional code calls a new `createMarriageFigure(...)` method to instantiate and initialize the marriage figure. This marriage figure is displayed using a `PolygonShape` (see Section 4.2 on page 29 for more about shapes) in the form of a diamond.

```java
private Canvas createDiagram(Composite parent) {
    ... existing figure creation for people here ...

    IFigure marriage = createMarriageFigure();
    root.add(marriage,
             new Rectangle(new Point(145, 35),
                           marriage.getPreferredSize()));

    ... prior code here ...
}
```
private IFigure createMarriageFigure() {
    Rectangle r = new Rectangle(0, 0, 50, 50);
    PolygonShape polygonShape = new PolygonShape();
    polygonShape.setStart(r.getTop());
    polygonShape.addPoint(r.getTop());
    polygonShape.addPoint(r.getLeft());
    polygonShape.addPoint(r.getBottom());
    polygonShape.addPoint(r.getRight());
    polygonShape.addPoint(r.getTop());
    polygonShape.setEnd(r.getTop());
    polygonShape.setFill(true);
    polygonShape.setBackgroundColor(ColorConstants.lightGray);
    polygonShape.setPreferredSize(r.getSize());
    return polygonShape;
}

Now the marriage figure is displayed when the main(...) method is run (see Figure 2–4).

![Genealogy window showing marriage figure.](image)

Finally, connect each of the people to the marriage (see Figure 2–5), showing their relationship to one another by modifying the createDiagram(...) method again. This is accomplished by calling a connect(...) method to create the line connecting the center of one figure to the center of another (see Chapter 6 on page 69 for more about connections).
private Canvas createDiagram(Composite parent) {

    ... existing figure creation for marriage here ...

    root.add(connect(andy, marriage));
    root.add(connect(betty, marriage));
    root.add(connect(carl, marriage));

    ... prior code here ...
}

private Connection connect(IFigure figure1, IFigure figure2) {
    PolylineConnection connection = new PolylineConnection();
    connection.setSourceAnchor(new ChopboxAnchor(figure1));
    connection.setTargetAnchor(new ChopboxAnchor(figure2));
    return connection;
}

Figure 2–5  Genealogy window showing connections.

2.4 Draw2D View

The above example diagram can also be displayed in a view that is part of an Eclipse RCP application (see Chapter 7 in the Eclipse Plug-ins book for more about views). Start by adding the following extension to the plugin.xml:
Now modify the `GenealogyView` class to be a subclass of `org.eclipse.ui.part.ViewPart`, and add the following methods:

```java
package com.qualityeclipse.genealogy.view;

import org.eclipse.ui.part.ViewPart;

public class GenealogyView extends ViewPart {
    public void createPartControl(Composite parent) {
        createDiagram(parent);
    }
    public void setFocus() {
    }
    ...
}
```

When you launch the runtime workbench and open the `Genealogy` view, you’ll see something like this (see Figure 2–6).
2.5 Draw2D Events

We would like the user to be able to drag the figures around the diagram. To accomplish this, we create a new Draw2D event listener to process mouse events, move figures, and update the diagram. Start by creating a new FigureMover class that implements the Draw2D MouseListener and MouseMotionListener interfaces. Add a constructor that hooks the listener to the specified figure and a concrete method that does nothing for each method specified in the interfaces.

```java
package com.qualityeclipse.genealogy.listener;

import org.eclipse.draw2d.*;
import org.eclipse.draw2d.geometry.*;

public class FigureMover
    implements MouseListener, MouseMotionListener
{
    private final IFigure figure;

    public FigureMover(IFigure figure) {
        this.figure = figure;
        figure.addMouseListener(this);
        figure.addMouseMotionListener(this);
    }

    ... stub methods here ...
}
```

When the user presses the mouse button, we need to record the location where the mouse down occurred by adding a field and implementing the `mousePressed(...)` method. In addition, this method must mark the event as “consumed” so that the Draw2D event dispatcher will send all mouse events to this listener’s figure until the mouse button is released.

Tip: When developing a Draw2D view with figures that don’t have dependencies on the Eclipse RCP framework, add a `main(...) method to your ViewPart as shown in Section 2.3 on page 9 so that you can quickly test your diagram in a shell rather than launching the entire Eclipse RCP application.
private Point location;
public void mousePressed(MouseEvent event) {
    location = event.getLocation();
    event.consume();
}

As the user moves the mouse around with the mouse button held down, we need to move the figure in the same direction and distance. The mouseDragged(...) method calculates the distance moved, moves the figure, and marks the event as consumed. To move the figure, we must update both the figure’s bounding box and the layout information. Both the figure’s original location and new location must be marked as “dirty” so that the update manager will redraw the diagram appropriately. The getBounds() method returns the actual rectangle object used by the figure to remember its bounds, so we cannot modify that object. Instead, we call getCopy() before calling translate(...) to prevent any undesired side effects.

public void mouseDragged(MouseEvent event) {
    if (location == null)
        return;
    Point newLocation = event.getLocation();
    if (newLocation == null)
        return;
    Dimension offset = newLocation.getDifference(location);
    if (offset.width == 0 && offset.height == 0)
        return;
    location = newLocation;

    UpdateManager updateMgr = figure.getUpdateManager();
    LayoutManager layoutMgr = figure.getParent().getLayoutManager();
    Rectangle bounds = figure.getBounds();
    updateMgr.addDirtyRegion(figure.getParent(), bounds);
    bounds = bounds.getCopy().translate(offset.width, offset.height);
    layoutMgr.setConstraint(figure, bounds);
    figure.translate(offset.width, offset.height);
    updateMgr.addDirtyRegion(figure.getParent(), bounds);
    event.consume();
}

Tip: To prevent undesired side effects, call getCopy(), then modify the copy rather than modifying the original rectangle.

When the mouse button is released, we clear the cached location and mark the event as consumed.
public void mouseReleased(MouseEvent event) {
    if (location == null)
        return;
    location = null;
    event.consume();
}

Finally, modify the GenealogyView class to import the FigureMover class and hook the listeners to each person figure and the marriage figure by modifying the createPerson(...) and createMarriage() methods. Once these steps are complete, the figures can be dragged around the window (see Figure 2–7). For more information on how the framework determines where figures are for the mouse events, see Chapter 7 on page 91.

private RectangleFigure createPersonFigure(String name) {
    ... existing code here ...
    new FigureMover(rectangleFigure);
    return rectangleFigure;
}

private PolygonShape createMarriageFigure() {
    ... existing code here ...
    new FigureMover(polygonShape);
    return polygonShape;
}
Implementing listeners such as these is useful when providing user interaction with pure Draw2D diagrams, but much of this functionality, such as dragging figures around a diagram, is already provided by the higher-level GEF framework.

### 2.6 Book Samples

Source code for each chapter can be downloaded and compiled into Eclipse for the reader to review, run, and modify. Go to www.qualityeclipse.com and click **Book Samples**, or go directly to the Quality Eclipse update site www.qualityeclipse.com/update to download the **QualityEclipse Book Samples** view into Eclipse. Once the samples are installed, open the view by selecting **Eclipse > QualityEclipse Book Samples**.

The view can be used to download the content for each chapter and compare the workspace content to the content in each chapter.

### 2.7 Summary

This chapter quickly brought the reader through a simple Draw2D example which includes a few figures that can be dragged and dropped on the canvas. The following chapters will walk through Draw2D content in more detail. All source code covered in this book can be downloaded from www.qualityeclipse.com.

### References

- Chapter source (see Section 2.6 on page 20).
INDEX

A
AbsoluteBendpoint 81–82
Absolute coordinates 102
AbstractBorder 46
AbstractConnectionAnchor 72–73, 75
AbstractConnectionEditPart 194
AbstractGraphicalEditPart 189–191
AbstractLayoutAlgorithm 167, 169–170
Accessibility 217
Action 182, 248
ActionBarContributor 202
ActionFactory 248
Actions 180, 182
addAncestorListener() 28
addChild() 75–76, 79, 94, 122, 220, 226
addCoordinateListener() 28
addDirtyRegion() 18, 23
addDisposeListener() 32
addDragSourceListener() 252
addDropTargetListener() 253
addFigureListener() 28, 124
addFocusListener() 27
addGenealogyGraphListener() 219
addKeyListener() 27
addMouseListener() 17, 27
addMouseMotionListener() 232
addNote() 125
addNotify() 222–223
addOffspring() 114
addParent() 75–76, 79, 94, 122
addPerspective() 227, 232
addPerspectiveListener() 119, 222
addPoint() 14, 31, 34, 78–80
addPrimaryFigure() 125
addRetargetAction() 248
addSelectionChangedListener() 212
addSelectionListener() 106, 125, 127, 151, 159
addSmallPolygonArrowheads() 77
addSmallPolylineArrowhead() 77
addSourceConnection() 224
addTargetConnection() 224
AlignmentAction 182
AncestorListener 28
anchor point 23
Anchors
Common 70
Custom 72
appendSelection() 215
applyLayout() 157, 159
applyLayoutInternal() 167–168, 171
ARROW 211
ArrowButton 32
ARROWHEAD 79, 195, 198, 243
Arrow Keys 217
B
BAR 105, 158
BasicAnchors 70
BasicBorders 43
BasicDecorations 77
BasicFigures 30
BasicRouters 80
Bendpoint 81
Absolute 82
Interface 82
Relative 82
Bendpoint 81
BendpointConnectionRouter 81–82
BendpointLocator 86–88
birthYearChanged() 116, 120, 222
Border 28
BorderLayout 57
Borders 37, 42
bounding box 22
bounds 37
Bucknell xxv
buildActions() 202, 248
Button 32
ByteArrayInputStream 206
C
CASCADE 105, 125, 158
category 16
CenterAnchor 72
chain() 232, 245–246
chained 232
ChangeBoundsRequest 235, 238
Checkbox 32
Click 250
Clickable 29
Clickables 29, 32
client area 37, 40
Clipping 40
CodePro xxv–xxvi
Color 28, 142–143, 152
ColorConstants 10, 12, 14, 30–34, 41, 47–49, 78–79, 97, 100, 141–142, 148, 154, 186, 195, 198, 208, 243
CombinedTemplateCreationEntry 249, 251
com.ibm.icu.text 186–187
Command
  Composite 232
  Create 227
  Delete 231
  Move and Resize 228
  Reorder 229
  Reparent 230
  Stack 226
Commands 180, 183, 219, 226
commandStackChanged() 227
COMPONENT_ROLE 240, 248
ComponentEditPolicy 240, 247
Components
  Creating 233
  Creation Tools 251
  Deleting 240
  Moving and Resizing 235
  Reordering 236
  Reparenting 238
composed figures 22
CompositeCommand 232
Composite Commands 232
CompositeLayoutAlgorithm 160–163, 167, 169, 172
CompoundBorder 43–44, 47, 50, 208
com.qualityeclipse.genealogy.editor 201
com.qualityeclipse.genealogy.parts 188–189
com.qualityeclipse.genealogy.view 186
configureGraphicalViewer() 203, 212, 214, 217
configurePaletteViewer() 252
connect 23
connect() 14–15, 76
Connection 15, 69–72, 121–122, 135–136, 146, 155, 243
ConnectionAnchor 84, 103, 196, 199, 242, 246–247
ConnectionCreationToolEntry 249, 252
ConnectionDragCreationTool 182, 244
ConnectionEditPart 198–199, 224–225, 242
ConnectionEditPolicy 247–248
ConnectionEndpointEditPolicy 210
ConnectionEndpointLocator 86, 88
ConnectionLayer 92–94, 100
ConnectionLocator 86, 88
ConnectionRouter 84–85
Connections 29, 69, 193–200
Anchors 70
Creating 240
Creation Tools 252
Decorations 76
Default Color 154
Default Width 154
Deleting 247
Fan Router 84
Labels 86
Manhattan Router 85
Modifying 244
Null Router 85
Routing 80
Shortest Path Router 85
Toolips 154
Undirected 153
Updating 223
CONNECTIONS_DASH 154
CONNECTIONS_DIRECTED 154
Constraints 55
consume() 18–19
Containers 29
containsAncestor() 216
containsPoint() 28, 86, 95–96
contributeToToolBar() 248
Control 131–132, 150, 169, 172
Controller 177
CoordinateListener 28
Coordinates 101
createAddCommand() 234, 238–239
createChangeConstraintCommand() 223, 233–235
createChild() 220
createChildEditPolicy() 236
Create Command 227
CreateConnectionCommand 225, 228, 241–242, 245–247
createConnectionCommand() 241
createConnectionRequest 241–242
createControl() 186
createDeleteCommand() 240
createDummyConnection() 243
createEditPart() 195
createElementsDrawer() 250–251
createFigure() 189–190, 195, 197–198, 243
createFilterMenuItem() 158
createFixedZoomMenuItem() 252
createHandle() 211
createMarriage() 19
Index

CreateMarriageCommand() 234
createMarriageFigure() 13–14, 19, 34
createMenuBar() 105, 118, 125, 127, 131, 158, 187
createMoveChildCommand() 235–237
CreateNoteCommand 234
CreateNoteCommand() 234
createOpenFileMenuItem() 125, 127, 187
createOrFindConnection() 224–225
createPalette() 250
createPaletteViewerProvider() 252
createPartControl() 124, 130, 186–188
createPerson() 19
CreatePersonCommand 221, 227
CreatePersonCommand() 234
createPersonFigure() 11–12, 19, 33
CreateRequest 234
createRotatedImageOfString() 29, 32
createSaveFileMenuItem() 127, 187
createScaleToFitMenuItem() 105–106
createSelectionHandles() 211, 236, 244
CreateSpouseConnectionCommand 245
CreateSpouseConnectionCommand() 241
createToolsGroup() 250–251
createView() 104
Creating Components 233
Creating Connections 240
Creation Drag and Drop 252
CreationRequest 233
CreationTool 181
CreationToolEntry 249
Creation tools 251
crop() 46
CustomFigureHighlightAdapter 150–151
Custom Figures 33

D
Dan Rubel xxv
deathYearChanged() 116, 120, 222
declareGlobalActionKeys() 202
Decorations 76
  Custom 78
  Default 77
  Rotatable 77
DefaultEditDomain 180, 203
DefaultHandler 117
DelegatingLayout 58, 86–88
DeleteAction 182
Delete Command 231
DeleteGenealogyConnectionCommand 245–246, 248
DeletePersonCommand 231–232, 240
DeleteRetargetAction 248
Deleting Components 240
Dimension 18, 51–52, 81, 83, 108–109
DirectedGraphLayoutAlgorithm 161–163, 167, 169–170, 172
DirectedGraphLayoutAlgorithm() 161
Display 10, 32, 41, 65, 126, 128, 131
dispose() 120, 130, 133–136, 142
DisposeEvent 32
Domain information 115
doSave() 203, 205–207
doSaveAs() 207
DOUBLE_BUFFERED 23, 97, 100, 122
DragEditPartsTracker 181, 236
DragTracker 211, 236
Draw2D xxiii, 1–2, 7, 118, 175, 177, 195
  Application 9
  Architecture 21
  Basic figures 30
  Diagrams 20
  Drawing 23
  Events 17
  Example 7
  Figures 53
  Graphics 39
  Infrastructure 21
  Installation 7
  Painting 37
  Processing Events 24
  Project 8
  View 15
drawImage() 39
drawLine() 39, 45–47
drawPolygon() 39
drawRectangle() 39
drawRoundRectangle() 39
drawText() 39
DROP_DOWN 105, 125, 158

E
EAST 211
EclipseCon 5–6
Eclipse Foundation xxiii, 6
Eclipse Modeling Framework, see EMF
Eclipse Plug-ins book xxiv, 9, 15
Edit > Delete menu 231, 240, 248
Edit > Redo menu 248
Edit > Undo menu 248
EditDomain 178, 180–181
EditFactory 195
Edit menu 246
  Adding and Removing 220
  Nested 226
EditPartFactory 178–179, 188, 203
EditPartViewer 178–181
Ellipse 30, 84, 86
EllipseAnchor 70–71
EMF 113, 176
EntityConnectionData 138–140
equals() 193
eraseSourceConnectionFeedback() 247
eraseTargetConnectionFeedback() 243, 247
eRCP 5
Eric Clayberg xxvi
ERROR 206
ErrorDialog 206
Event 24
EventDispatcher 22, 24–25
EventManager 22
EventObject 227
execute() 227–229, 231–232
expand() 34
extension 16
F
FanRouter 84
Figure 10, 33, 42, 57, 66, 86, 93, 95–96, 121, 189, 192, 207, 209, 211
FigureCanvas 97–100, 105, 122, 125, 127, 186
FigureListener 28, 124
figureMoved() 124
FigureMover 17, 19, 33–34, 36, 50, 64, 66, 95–96, 125
Figures 27, 177
Borders 42
Bounds 37
Child 28
Clickables 29
Client Area 37
Clipping 40
Common 29
Common Borders 43
Complex 27
Connections 29, 69
Containers 29
Custom 33
Custom Borders 45
Custom Painting 41
Extending Existing 33
Graphics 39
Layered 29
Layout Managers 55
Maximum 56
Minimum 56
Nested 27, 35
Painting 37
Paint Methods 38
Preferred Size 56
Sample Code 30
Shapes 29
Z-Order 27, 40
File 128
File > Save As menu 207
File > Save menu 205
FileDialog 126, 128
FileEditorInput 207
FileInputStream 126
File menu 125
FILL_BOTH 61
FILL_HORIZONTAL 61
FILL_VERTICAL 61
fillPolygon() 39, 47
fillRectangle() 39, 41, 49
fillRoundRectangle() 39
fillText() 39
Filter menu 158–159
Filters 157
findConnection() 224–225
findFigureAt() 28, 95
firePropertyChange() 207, 227
fireSelectionChanged() 208
fish-eye effect 149
fisheyeNode() 148–149
Flow 3
FlowLayout 59–60, 62
FocusListener 25, 27
Font 28, 32
FrameBorder 43
FreeformFigure 98–100
FreeformFigures 179
FreeformGraphicalRootEditPart 179
FreeformLayer 99–100, 189
FreeformLayeredPane 100, 104
FreeformLayout 189
FreeformViewport 100, 107, 179
G
ac 39
GEF xxiii, 1–2, 7, 128, 183
Adding and Removing EditParts 220
Commands and Tools 219
Editor 201, 219
Examples 2
Flow 3
Logic 4
Shapes 2
Text 4
WindowBuilder 5
XMind 5
getFreeformExtent() 107
getGender() 119, 139, 158, 190
getGraphAdapter() 122
getGraphControl() 143, 151
getGraphicalViewer() 203, 253
getHeight() 124, 229, 231
getHighlightColor() 152, 154
getHost() 236
getHusband() 194
getImage() 119, 139
getInsertionReference() 235, 237–238
getInsets() 45
getLayoutEntity() 172
getLayoutManager() 18
getLeft() 14, 34
getLineStyle() 39
getLineWidth() 39
getLocation() 18, 72–75, 103, 108, 110–111, 238
getMarriage() 134, 136, 232
getMarriages() 133, 189
getMaximumSize() 56
getMinimumSize() 56
getModelChildren() 193–194, 198
getModelSourceConnections() 193–194, 198
getName() 190, 207
getNewObject() 234, 251
getNewObjectType() 234
getNodeFigure() 146, 155
getNodeHighlightColor() 148
getNotes() 148, 156, 189–190, 229–231
getNotesContainer() 191–192
getOffspring() 134, 197–198
getOwner() 72, 75, 103, 108, 110–111
getPaletteRoot() 203, 250
getPaletteViewer() 252
getParentsMarriage() 136, 198, 232
getPeople() 133, 136, 189
getPersonFigure() 122, 222
getPreferredSize() 11–13, 30–31, 51–52, 56, 144
getReconnectSourceCommand() 244–245
getReconnectTargetCommand() 245–246
getRelationships() 134
getResourceAsStream() 65, 118, 124, 130–131
getRight() 14, 34
getRoot() 207
getRootEditPart() 213
getSelected() 208
getSelection() 151, 213, 215
getSelectionSynchronizer() 217
getShell() 206–207
getSite() 206–207
getSize() 148
getSource() 136, 146, 155, 171
getSourceAnchor() 84
getSourceConnectionAnchor() 198–199, 242, 246–247
getStart() 21
getStartCommand() 241–242
getTarget() 196
getTargetAnchor() 84
getTargetConnectionAnchor() 196, 198–199, 242–243, 247
gexText() 138, 156
gexToolTip() 28
gexTooltrip() 148–149, 154
gexTop() 14, 34
gexTopright() 72
getTotalNumberOfLayoutSteps() 169
getUpdateManager() 18
gexViewer() 220, 225–226
gexWidth() 124, 229, 231
gexWidthinLayout() 172
gexWife() 194
getWorkspace() 207
gexX() 229, 231
gexY() 229, 231
gexYearMarried() 138, 144
Global Edit Menu Actions 248
Google xxiii, xxy–xxvi, 6
Google Plug-in for Eclipse, see GPE
Google Web Toolkit, see GWT 5
GPE xxiii
gradient 41
Graph 143, 150–151
GraphConnection 146, 155
GRAPHICAL_NODE_ROLE 240
Graphical Editing Framework GEF SDK 7
Graphical Editing Framework, see GEF
Graphical Editing Framework Zest Visualization
Toolkit SDK feature 129
GraphicalEditor 202, 217
GraphicalEditorWithFlyoutPalette 202–203, 250
GraphicalEditorWithPalette 202, 250, 252
GraphicalEditPart 190, 211
GraphicalNodeEditPolicy 225, 244, 245, 246, 247
GraphicalViewer 203, 207, 213–214
GraphicalViewerImpl 179
GraphicalViewerKeyHandler 217
Index

Graphics
  Drawing 39
  Property access 39
  Saving state 39
Graphics 38-39, 41, 44-46, 49, 53
GraphLabel 150
GraphNode 152, 172
GraphViewer 131-132, 137-138, 143, 151, 157-158
GridLayout 10, 60-61, 105, 131
GridLayoutAlgorithm 162
GroupBoxBorder 43-44
GroupRequest 240, 248
GWT xxvi, 5
GWT Designer xxiii

Handle 211, 236
handleException() 206
hasChildren() 136, 138
hasFocus() 28
hashCode() 193
heavyweight 22
Highlight 151-152
highlight() 152
Hit Testing 95
HorizontalLayoutAlgorithm 164
HorizontalShift 160-163, 167, 169-170, 172
HorizontalTreeLayoutAlgorithm 164
husbandChanged() 121, 224

IColorProvider 137, 141, 143, 146, 148
IConnectionStyleProvider 137, 153-154
IEditorInput 204
IEditorPart 180, 227
IEntityConnectionStyleProvider 137, 153-154
IEntityStyleProvider 137, 148-149
IFigureProvider 137, 144
IFile 204, 206-207
IFileEditorInput 204, 206-207
IGraphContentProvider 132, 135
IGraphEntityContentProvider 132-134
IGraphEntityRelationshipContentProvider 132, 134
IGraphLabel 137
Image 32, 65, 119, 121, 139
ImageFigure 29, 32, 66
ImageUtilities 29, 32
indexOf() 229-231, 237
Indigo 7
INestedContentProvider 132, 136, 156
initializeGraphicalViewer() 203, 252
initializePaletteViewer() 252
inputChanged() 133-135
InputStream 131
Insets 45-46
installEditPolicy() 210, 234-240, 248
Install New Software menu 185
Instantiations xxii-xxiii, xxvi
InternalNode 167-168, 171-172
InternalRelationship 167-168, 171-172
InvalidLayoutConfiguration 168
INVERTED_TRIANGLE_TIP 77
IPath 207
IProgressMonitor 206
isAncestor() 216
isCoordinateSystem() 28
isDisposed() 10
ISelection 214
ISelectionChangedListener 213
IStyle 146
IStyleProvider 137, 146
isOffspringConnection() 197-198, 243, 245
isOpaque() 28
isSaveAsAllowed() 206
IStructuredContentProvider 132
IStructuredSelection 213-215
isValidConfiguration() 168
isValidSource() 228, 245-246
isValidTarget() 228, 242, 246-247
isVisible() 28
IToolBarManager 248

Jaime Wren xxvi
JFace 131-132, 157, 179, 182

Keyboard 217
KeyListener 25, 27

Label 10, 12, 22, 29, 32-33, 36, 43-45, 47-48, 50, 56-57, 61, 64, 71, 87-88, 96, 121, 135
LabelAnchor 71
LabelProvider 138
LabelRetargetAction 248
Layer 29, 92-93, 95, 99
Layered 29
LayeredPane 29, 93, 98, 100, 104
Layers 91-95
LAYOUT_ROLE 233-239
LayoutAlgorithm 163, 167, 169, 172
Index

Layout Algorithms 160
  Composite 161
  Custom 167
  Directed Graph 162
  Graph 162
  Horizontal 164
  Horizontal Shift 163
  Horizontal Tree 164
  Radial 164
  Spring 165
  Tree 166
  Vertical 166
LayoutEntity 172
LayoutManager 18, 55
Layout Managers 10, 55
  Common 57
  Constraints 55
  Using 63
LayoutStyles 140, 161–165, 169, 172
lightweight 22
Lightweight Drawing System 22
LightweightSystem 10, 22–24
LINE_DASH 44
LINE_DOT 47
LineBorder 32, 44–45, 47, 50, 66, 208
Listener 24
Listeners 115
Listening for Model Changes 219
locationChanged() 116, 222–223
Locator 58
Logic 4

M
  main() 9, 11, 13–14, 17, 131, 137, 187
  ManhattanConnectionRouter 85
  MANIFEST.MF 185–186
  MarginBorder 43–44, 47, 50, 66, 189, 208
  markSaveLocation() 206
  MarqueeToolEntry 249, 251
  Marriage 114–116, 119–120, 133–134, 138, 144,
  172, 188, 193–194, 198, 220–221, 224–225, 227–
  228, 232–234, 240–241, 244, 249, 252
  MarriageAdapter 119–121, 124
  marriageAdded() 220
  marriageAnchor 74–75, 79, 103, 107–109, 111–112,
  146, 155, 195–196, 199
  marriageChanged() 116, 120, 223–224
  MarriageEditPart 188, 190–191, 194, 196, 198,
  MarriageFigure 33–35, 63–64, 69, 74–76, 79, 91,
  95–96, 102–104, 107, 109–110, 112, 123, 144,
  146, 155, 195, 209, 211
  MarriageFigures 121
  MarriageDirectionalNodeEditPolicy 241, 243–245
  MarriageLayoutAlgorithm 170, 172
  MarriageListener 119, 121, 221–222, 224
  marriageRemoved() 220
  Math 75, 107, 110
  Menu 105–106, 125, 127, 158
  MenuItem 105–106, 125, 127, 158
  MessageDialog 128
  MidpointLocator 86
Models
  Displaying 203
  GEF 113
  Hooking Diagram to Model 124
  Hooking model to a diagram 118
  Listeners and Adapters 119
  POJO 113
  Populating the Diagram 116
  Reading 116
  Reading from a File 125
  Saving 205
  Serializing model information 126
  State Changes 177
  Storing the Diagram 126
  Types 176
  Writing to a File 127
Model-View-Controller, see MVC
ModifiedSelectionManager 214
mouse button 18
mouseDragged() 18
MouseListener 17, 25, 27
mouse listener 25
MouseMotionListener 17, 25, 27
mousePressed() 17–18
mouseReleased() 19
Move and Resize Command 228
MoveAndResizeGenealogyElementCommand 223, 228
MoveAndResizeGenealogyElementCommand() 235
moveHandle() 211
Moving and Resizing Components 235
MVC 113, 176
MyCreateCommand 232
MyDeleteCommand 232
MyOtherCommand 232

N
  nameChanged() 116, 120, 222
  Nested Content 156
  nested figures 22, 35
newConnection() 84, 86
newFigure() 84, 86
newFigureAndConnection() 77, 79, 81, 85, 88
### Index

<table>
<thead>
<tr>
<th>Page</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
<td>newSAXParser()</td>
</tr>
<tr>
<td>119–121</td>
<td>NoteAdapter</td>
</tr>
<tr>
<td>116</td>
<td>noteAdded()</td>
</tr>
<tr>
<td>116, 120, 220, 226</td>
<td>NoteBorder</td>
</tr>
<tr>
<td>114–115, 229–231, 238–239</td>
<td>NoteContainer</td>
</tr>
<tr>
<td>116</td>
<td>NoteContainerListener</td>
</tr>
<tr>
<td>188–191, 194, 198, 210, 221–226, 226, 235, 238, 240</td>
<td>NoteFigure</td>
</tr>
<tr>
<td>119, 221–222</td>
<td>NoteListener</td>
</tr>
<tr>
<td>116, 120, 220, 226</td>
<td>noteRemoved()</td>
</tr>
<tr>
<td>114</td>
<td>Notes</td>
</tr>
<tr>
<td>106, 125, 127</td>
<td>NULL</td>
</tr>
<tr>
<td>NULLConnectionRouter 85</td>
<td><strong>O</strong></td>
</tr>
<tr>
<td>ObjectShare xxvi</td>
<td>Object Technology International xxii</td>
</tr>
<tr>
<td>121</td>
<td>Palette</td>
</tr>
<tr>
<td>113, 176</td>
<td>Polygon</td>
</tr>
<tr>
<td>77–78, 198, 243</td>
<td>PolygonDecoration</td>
</tr>
<tr>
<td>13–14, 19, 34</td>
<td>Polyline</td>
</tr>
<tr>
<td>29, 31</td>
<td>PolylineConnection</td>
</tr>
<tr>
<td>13–14, 19, 34</td>
<td>PolylineDecoration</td>
</tr>
<tr>
<td>29</td>
<td>popState()</td>
</tr>
<tr>
<td>32, 87–88</td>
<td>PositionConstants</td>
</tr>
<tr>
<td>168</td>
<td>postLayoutAlgorithm()</td>
</tr>
<tr>
<td>109–111</td>
<td>PrecisionDimension</td>
</tr>
<tr>
<td>109–111</td>
<td>PrecisionPoint</td>
</tr>
<tr>
<td>167, 170</td>
<td>preLayoutAlgorithm()</td>
</tr>
<tr>
<td>115</td>
<td>Presentation information</td>
</tr>
<tr>
<td>182</td>
<td>PrintAction</td>
</tr>
</tbody>
</table>

**Note**: The index includes various terms and their page numbers related to various aspects of the document, such as object technology, person, and graphical design elements, among others. The terms are organized alphabetically, with page numbers indicating their respective locations within the document.
println() 127
PrintWriter 126–128, 206
PROP_DIRTY 206, 227
PROP_INPUT 207
pushState() 39

Q
QualityEclipse Book Samples view 20

R
RadialLayoutAlgorithm 164–165
RADIUS 108, 110–111
RCP 9, 17
RCP Developer xxv
readAndClose() 117–118, 124, 126, 130–131, 187, 203–204
readAndDispatch() 10
ReconnectRequest 245–247
recreateCommand() 245–247
RectangleFigure 12, 19, 22, 31, 33–34, 42
redo 248
RedoRetargetAction 248
refreshVisuals() 190
Relationships, see Connections
RelativeBendpoint 81–83
Relative coordinates 102
relocate() 58
remove() 120
removeChild() 220
removeNote() 231
removeNoteNotify() 229–231
removeNotify() 222–223
removePerson() 227, 232
removePersonListener() 120, 222
removeSourceConnection() 224
removeTargetConnection() 224
Reorder Command 229
reordered 236
Reordering Components 236
ReorderNoteCommand 229, 237
Reparent Command 230
reparented 230
Reparenting Components 238
ReparentNoteCommand 230, 238–239
Requests 180, 182
ResizeHandle 211
resolveRelationships() 117
ResourcesPlugin 207
restoreState() 39
RootComponentEditPolicy 248
RootEditPart 178–179, 203
root figure 10
RotatableDecoration 76–77
RoundedRectangle 31
run() 9–10, 105, 107, 117–118, 131, 187

S
Sample Code
Book 20
Borders 43
Clickables 32
Shapes 30
SAVE 128
SaveAsDialog 207
saveFile() 127–128, 187
Save menu 127
SAX Parser 116
SAXParserFactory 117
ScalableFigure 104
ScalableFreeformLayeredPane 104–105
ScalableFreeformRootEditPart 179, 186, 203
scaleToFit() 106–107
Scaling 104
Dimensions 107
Figures 104
Zoom menu 105
Scrolling 96
ScrollingGraphicalViewer 179, 186
ScrollPane 29
SELECT_ALL 248
SelectAllAction 182
SELECTED 208
SELECTED_NONE 208
SELECTED_PRIMARY 208
SelectEditPartTracker 211, 236
Selection 207
Accessibility 217
Change Listener 212
Edit Policy 209
Keyboard Actions 217
Making Visible 207
Manager 214
Multiple Editors 217
Synchronizing 217
Tools 250
SELECTION_FEEDBACK_ROLE 209–210, 235
selectionChanged() 213
SelectionChangedEvent 213
SelectionChangedListener 214
SelectionChangedListener 212
SelectionEvent 106, 125, 127, 151, 159
SelectionListener 106, 123, 127, 151, 159
SelectionManager 214
SelectionModificationChange 212–213
SelectionSynchronizer 217
SelectionTool 181
SelectionToolEntry 249
selfStyleConnection() 146, 154–155
selfStyleNode() 146
setAfterNote() 229–230, 237, 239
setBackgroundColor() 12, 14, 28, 30–31, 33–34, 42, 47–49, 78–79, 152, 195, 198, 243
setBorder() 28, 43–44, 48, 50, 66, 189, 208, 211
setContents() 10, 93, 97, 178, 180, 187–188, 203
setMessageSize() 12, 14, 31, 34
setNextRouter() 84
setOldContainer() 230, 238–239
setOpaque() 28
setOpaque() 48–49
setName() 120–121, 222
setPartName() 204, 207
setPreferredSize() 12, 14, 30, 33–34, 44, 50, 56, 66, 208
setRelativeDimensions() 81, 83
setRelativePosition() 87–88
setRootEditPart() 186, 203
setScale() 106–107
setSelected() 207–208
setSelection() 214
setSize() 10, 124, 131, 222, 227, 229, 231
setSource() 228, 245
setSourceAnchor() 15, 70–72, 75, 79, 146, 155
setSourceDecoration() 77
setSpacing() 50, 63, 66, 192, 208
setStart() 14, 34
setStartCommand() 241
setTarget() 228, 242, 245–246
setTargetAnchor() 15, 70–72, 75, 79
setTargetDecoration() 77–79, 195, 198, 243
setTemplate() 77–79, 195, 198, 243
setToolTip() 155
setUDistance() 88
setVDistance() 88
setVerticalSpacing() 57
setViewport() 100
setWidth() 208
Shapes 2, 29–30
Shell 10, 105, 125–128, 131, 158
ShiftDiagramLayoutAlgorithm 167, 169, 172
ShortestPathConnectionRouter 85–86, 91–93
showSourceConnectionFeedback() 247
showTargetConnectionFeedback() 243, 247
SimpleEtchedBorder 44
SimpleFactory 251
Index

Z
  Color 141
  Connection Highlight 153
  Content Provider 132
  Custom Figures 144
  Filters 157
  Installation 129
  Label Provider 138
  Layout Algorithms 160
  Model-View separation 113
  Nested Content 156
  Node Size 140
  Plug-in Dependencies 130
  Presentation 137
  Setup 129
  Styling 153
  Styling and Anchors 146
  Subinterfaces 132
  Tooltips 153
  ZestContentProvider3 135
  ZestStyles 154
  Zooming, see Scaling
  Zoom menu 105
  Z-Order 40