UNIVERSITY OF THE PHILIPPINES MANILA
COLLEGE OF ARTS AND SCIENCES
DEPARTMENT OF PHYSICAL SCIENCES AND MATHEMATICS

XML GUI Generator (XGG): an XML User Interface Language for Java Swing

A Special Problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Roy Gian S.P Balderrama
April 2008

ACCEPTANCE SHEET

The Special Problem entitled "XML GUI Generator (XGG): an XML User Interface Language for Java Swing" is prepared and submitted by Roy Gian S.P Balderrama in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

_____

Ma. Sheila A. Magboo, MS

Adviser

EXAMINERS	APPROVED	DISAPPROVED
1. Gregorio B. Baes, Ph.D (Candidate)	_____	_____
2. Avegail D. Carpio, MS (Candidate)	_____	_____
3. Richard Bryann L. Chua, MS	_____	_____
4. Aldrich Colin K. Co, MS (Candidate)	_____	_____
5. Vincent Peter C. Magboo, MD, MS	_____	_____
6. Geoffrey A. Solano, MS	_____	_____

_____

Date

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Geoffrey A. Solano, MS
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Alex C. Gonzaga, Ph.D.
Chair
Department of Physical
Sciences and Mathematics

Reynaldo H. Imperial, Ph.D.
Dean
College of Arts and Sciences

Abstract

XML UI Language (XUL) is a new way in creating Graphical User Interface Applications in the web or in native platforms via an HTML like syntax of an XML file. XUL is very expressive and makes creating GUI application easier. XUL in Java is mostly focused in web development and is implemented by dynamically reading XML files each time the application starts, therefore requiring that all items in the XML files matches other parts of the application. XGG, instead of dynamically processing the XML file, compiles the XML file and outputs it as a structured Java code, which when compiled produces the desired screen output. XGG also creates a file where all the methods and variables needed by the screen are placed, and updates it when needed. XGG also supports Style Sheets via another XML file, binding of a Java Class file or a database result, proper indention in generated code, overwriting the drawing functions of a Swing Component and automatically compiling and testing the output. XGG architecture is based strictly on the Model View Controller (MVC) paradigm, where a GUI application is split into 3 parts, effectively separating code and making integration and maintenance easier.

Keywords: XML, XUL, Java Swing, Compiler, Model View Controller paradigm

– TABLE OF CONTENTS –

I. INTRODUCTION

A. BACKGROUND OF THE STUDY

The Standard Generalized Markup Language (SGML) is a metalanguage used to define markup language for different uses. It is originally designed to provide a human and machine readable format to share in industries.

The syntax for SGML is very broad and complex. Attributes may or may not require a double quotation. You can have more than one way of expressing tags. SGML does not have a clear hierarchical organization. This complexity however, deemed it unusable in small scale general use [1].

There have been many markup languages that are derived from SGML. The most famous is the Hypertext Markup Language or HTML, which is used to display data on the web. Another markup language derived from SGML is the Extensible Markup Language or XML. It is a general purpose markup language created for the sharing of data within different systems, particularly via the internet.

Unlike HTML, XML is stricter than SGML by adding semantic constraints, and thus is easier to parse and is able to support more data structures. It also has support for extensions, thru the use of namespaces [2].

There are many technologies that use XML nowadays:

- Scalable Vector Graphics or SVG, a static or dynamic two-dimensional vector image [3]
- Simple Object Access Protocol or SOAP, a protocol for server-client communication mainly used for method invocation [4]
- CAMbase, an XML based database for Complementary and Alternative Medicine (CAM) [5]
- XForms, a replacement for HTML forms [6]
- Mozilla XUL or XML User Interface Language, a language developed by Mozilla to be used by it's cross-platform applications like Firefox [7]

As Mozilla XUL become more famous, there have been many versions of it that can be used in the web and other programming languages. XML UI languages nowadays are slowly becoming popular due to Adobe's Flex and Microsoft's release of WPF and are regarded by some to be the next step in UI Development programming [8].

It features a clear separation of the design of the UI and the application logic. It enables UI designers to work separately with the logic programmers and it also shortens coding. XML is also referred to as declarative typed language and is referred as more expressive than normal coding [7].

B. STATEMENT OF THE PROBLEM

Java's Swing UI toolkit's architecture is structured and hierarchical, and therefore it is easy to create XML UI languages on top if it. However, Java's strict object oriented structure makes it hard to implement an XML UI language that focuses on the ease for the logic side.

For example, other languages that have function pointers, like C++, can just simply register a method to the components event handlers. XUL created in such languages can just easily do the code for registering a function. While in Java, you need to extend a Listener Object, override its functions, instantiate the listener and attach it to the event handlers. There is no easy way to separate the function of the listener object from the whole class containing the components.

XUL languages in Java, instead on generating classes, create the UI from an inputted XML file on runtime, and expect that the logic code "matches" all that is needed from the design code. So what normally happens is that the user would have to be extra careful to match the identifiers from the XML code to the Java code. It would be very difficult for someone, especially a beginner, to debug the program.

Another thing that is not convenient to do in Java Swing is when trying to create custom components, either from scratch or by extending another component. To create a custom component, one would extend a component class, fill the paint method using graphical methods and use their shape class counterparts to describe when a mouse event is within that component. XUL can make the process easier by being able to provide a way to express how a custom component would look like.

XUL made in Java Swing separates the UI from the application code. This helps in clearing the code, but some would still be using just one class to put all other code. Such is not considered a good practice in object oriented programming. We can introduce the Model-View-Controller programming paradigm in a XUL, just like what other web-based XUL has done. MVC splits the programming design into 3 parts, the data model, its interface and the communication between the model and interface [9].

Most XML languages created needs the programmer to also deploy additional runtime libraries using the XML language. While some of the libraries are small, this is not appealing and not practical to those who are making very small GUI programs in Java.

C. OBJECTIVES

1. To be able to create an XML UI language that can be compiled to Java code containing the corresponding interface layout that supports the events described in the inputted XML file.

2. To be able to provide three interfaces; a command line interface, a graphical user interface front end and a simple plug-in for Eclipse IDE.

3. The user can do the following with a XUL file:

    - Define the layout of the Swing components
    - Specify the properties of a Swing component using attributes.
    - Define events generated by a Swing component; the events are to be filled .in a back-end file
    - Do data binding for Java Beans
    - Bind a database table in a XUL file
    - Change the graphical routines used to draw a Swing component
    - Use an XML Style Sheet to change group of attributes of a whole XUL file

4. Allows the user of the command line interface to:

    - Input an XUL file via the command line program and receive an output of a Java code of a class that contains the UI described in the input.
    - Choose if the output is an abstract class that provides functions for the events described in the input file or as a runnable class that only has the UI.
    - Choose if the XUL file is for the main class or not.
    - Choose to create UI used for applets.
    - Receive informative error messages when the inputted file does not comply with the XUL.
    - Choose to have the generated code be compiled and have the Java byte code as the output.
    - Choose to have the file that contains the events be generated.

- Supply the file that contains the events that are used by the UI file. That file will be updated to contain all the events that are described in the inputted XML file.
- Choose the bracing style and indentation of the generated Java code.

5. Allows the user of the front end to:

- Browse or drop in the window the XUL file to be used as the input.
- Change the compiler setting through radio buttons and checkboxes, and save the settings for next use.
- Use all the features of the command line program.

6. Allows the user of the plug-in to:

- See the result of the XUL file while editing it without manually recompiling the file.
- See a tree view of the XUL file to examine the structure of the XUL.
- Use all the features of the command line program except those that are already provided by the IDE.

D. SIGNIFICANCE OF THE STUDY

The creation of a simple yet powerful XML UI language in Java is a helpful tool for programmers who are familiar with Swing to conceptualize UI design. This can be used as a tool to assist in creating simple UIs to advanced UIs requiring custom components. It supports the usage of the Model View Controller Architecture and prevents the clutter of logic and GUI code so it helps the programmer in having a better approach when creating GUI applications. It can be used as a stepping stone in learning other advanced XML UI languages.

E. SCOPE AND LIMITATIONS

- Supports only the Java programming language.

- Supports only the Swing UI Toolkit.
- Supports only MySQL as database models.
- The system creates Java code to be compiled with the default Java compiler, and will not be adding new features to the default Java compiler.

F. ASSUMPTIONS

- The user has installed the Java Development Kit with the version number to be at least 1.5 / 5.0.
- The user manually creates a Java Bean class or a database to bind to the UI.
- The user has installed the driver for MySQL in Java.
- The user is familiar with Swing in the Java language, and has basic knowledge in XML or HTML.

II. REVIEW OF RELATED LITERATURE

Ever since the creation of SGML in the 1960's, many people already knew that it would be a very useful format in the computer science world. HTML is a markup language derived from SGML and has become one of the most useful languages in the web. However SGML has problems that caused it to be not used much, like the language's complexity that parsers made for it are slow, and the markup language lack versatility, as it cannot be expanded dynamically. A group of people tried to make an open standards version of it. In 1998, XML is created and became a W3C recommendation. XML has versatility to expand due to namespaces, power to easily express most common data, strict syntax so that parsers made for it are efficient and platform independent nature [1 - 2, 10].

XML's characteristics have made it into a very important format especially in the web. It caused many new technologies for the web to appear, and is viewed to be the next step from HTML. [10] In XHTML, an XML language designed to replace HTML, is now being slowly recognized and will one day replace HTML. Through the use of Document Type Definition (DTD), a format describing a markup language, one can create a unique document and let the Web browsers understand it. Data formats in XML or can be converted to XML will be human and machine readable, allowing for better communication [11 - 12].

XML is also used in databases available online [5]. The power of XML to adapt to changes in a specific document's specification means that the tools to manipulate data are very powerful because it is not fully dependent on the structure of the data. An example of such an application is a search engine that can find a specific data, for example "title of a book" in many XML databases, even though the database does not use the same tag or format for the title of a book. There are also many new approaches developed from how XML is used to store data that complex database queries can now be done within them [13 - 14].

One particular use for XML was to simplify coding for user interfaces, whether it is a rich browser application or a stand-alone application. Its syntax is almost like HTML, where the tags are the components of a UI (buttons, text field etc.). The components of a UI are created from top to bottom using the formatting defined in the markup file itself. This particular XML language is called XML UI Language, or XUL [9, 15 - 16].

In XUL (pronounced "zool"), a designer can create a UI easily by using a plain text editor and knowledge in programming is not needed. XUL hides the inner workings when creating the UI, leaving only the designing part. It is also possible for XUL to be language dependent, enabling for one particular XUL to work within Java and C++. It can also be widget independent, with one XUL code to be compatible for Windows, GTK, etc. Therefore it promotes code reusability when changing platforms or widgets. XUL may also clearly separate the design of the UI and the logic of the program [17 - 18]. XUL also has the potential to support the Model View Controller Architecture, a programming practice where design and logic is separated within 3 parts. XUL already separates the design, or the view. It is up to the implementation of the XUL or to the programmers if they will use the model [19 - 20].

One of the first successful XUL developed is Mozilla's XUL, an XUL developed by Mozilla to be used by its cross-platform applications like Firefox. Javascript is the scripting language used for the application logic code. Applications developed from this are either a Firefox extension, a Remote XUL application – an application that is ran in a Firefox browser, a XULRunner application, an application that can run if the client has XULRunner, Mozilla's platform is installed. Some of its features are portability, localization and support with other XML languages like SVG and MathML [21].

Another successful XUL developed is Luxor, which aims to be better featured than Mozilla's XUL. However, it is more conceived as a port of Mozilla XUL to Java. It supports multiple widget sets, has its own API for building UIs, web server and portal engine. It supports Java and Python as its scripting languages. It gives clear separation of content, appearance, behavior and locale [22].

Thinlet is an XUL developed by Robert Bajzat for applets and mobile devices. It uses Java as its core programming language. The aim of Thinlet is to create an XUL engine that does not require a very extensive and bulky runtime library because its primary target is for the mobile devices. It has its own GUI toolkit; it does not use any of Java's primary toolkits like Swing [23].

SwiXML is an XUL developed by Wolf Paulus that focuses on only generating Java Swing UI structures for java applications or applets. It only supports the basic Java Swing but has more advantages than many other XUL created for Java. Many programmers that are experienced with Swing need not learn much to grasp the language. It does not have its own layer or wrapper for the Swing toolkit meaning hat it is faster than other XUL. This also means that SwiXML has smaller runtime needed [24].

Microsoft has also been developing a XUL for their Windows Vista OS. It is called WPF, codenamed Avalon, part of the .NET Framework 3.0. Its markup language is Extensible Application Markup Language or XAML, a language derived from XML. WPF can be used to create standalone applications or sandboxed applications that can run in a browser, called XAML Browser Applications or XBAP. XBAP has a subset called Silverlight that can run in any platform, and is Microsoft's counterpart for Flash [25].

III. THEORETICAL FRAMEWORK

A. STANDARD GENERALIZED MARKUP LANGUAGE (SGML)

The Standard Generalized Markup Language or SGML is a markup language originally designed to enable the sharing of machine readable formats in large projects to different networks. It is primarily intended for text and database publishing, and one of the first applications of SGML was the second edition of Oxford English Dictionary.

The language is complex, which prevented it to be used in small-scale data. The complexity also causes SGML parsers to be inefficient. There are many ways to implement an opening and closing tag, and errors are often ignored [1].

B. HYPERTEXT MARKUP LANGUAGE (HTML)

The Hypertext Markup Language or HTML is the markup language used for displaying text in a web browser. It can display images and other objects. HTML supports scripting languages to make pages more interactive. It is the most widely used language in the internet [26].

C. EXTENSIBLE MARKUP LANGUAGE (XML)

The Extensible Markup Language or XML is a markup language specifically designed to describe data. It does have a predefined set of tags like HTML; instead users of XML define their own set of tags. On its own, XML can only describe the data. What makes XML useful in many ways is that there are efficient tools that can be easily designed to interact with any type of XML data to produce the desired result.

XML is designed to be widely used in the internet. Different systems employ different ways to read data and because of this, data interchange between different systems may be hard. XML is there to bridge the gap about incompatible data types. It can support a wide variety of uses. It has consistent syntax so that XML parsers can be efficient and easy to implement. XML is both human and machine readable. Most of all, XML documents is easy to design and create.

Another feature of XML is that it is very expressive and is a strong declarative language. There are many data structures that are easier to describe in XML than any other formats. It can describe data that can have many attributes, children elements, and have many forms easily. It is also human readable and can be made by hand, meaning using only a simple text editor. These characteristics make XML a good language to be used for describing user interfaces made up of layouts of components like buttons, text fields etc [27].

XML documents

A data object or file is considered as an XML document if the data object passes as a well-formed XML file. A well-formed XML file is a document that has passed the constraints of a XML file, and all files described in the document referred to as an XML file

also passes the constraints. There is also a valid XML document, which is a well-formed XML document that passes another set of constraints. The additional constraint is conforming to a schema defined by a Document Type Definition.

XML syntax

Each element in XML, just like any other markup language, is defined by a tag. There are 2 types of elements present in XML:

Non-empty Elements:

Non-empty elements have a content surrounded by a start and an end tag. Only the start tag can have attributes. The content tag can contain another element or data. This is similar to the body tag in HTML.

Example:

```
<tag attrib="value">Content</tag>
```

Empty Elements:

These are the elements that have no content. It only consists of the empty-element tag. The empty-element tag is like the start tag except that it end with a "/>" and not a ">". Alternatively, a start tag and an end tag without any content is also considered an empty element.

Example:

```
<emptytag attrib="value" />
```

As we can see, tag names and attribute names can have only 1 word. Also attribute values must be enclosed with double quotes. XML also have other constructs like comments and CDATA sections, used for escaping strings.

The top of every XML element is marked by a tag called the root; all other tags must be under the root. XML tags follow a strict hierarchical structure with the root at the top [28].

D. OBJECT ORIENTED PROGRAMMING

Object Oriented Programming is a programming language model that revolves around the usage of "objects". Rather than the normal view of a program as a group of functions that comprises the input-process-output cycle, OOP views the program as a group of entities called "objects". An object consists of its parts and characteristics, or the data; and its behavior, or the methods. Objects make programs more accurately conform to the way humans deal with real world. [29]

E. JAVA PROGRAMMING LANGUAGE

Java is a programming language developed by Sun Microsystems. According to Sun, "Java is a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language." It is one of the most popular languages today, with only the .NET languages as its only competition in the mainstream programming. [30]

F. MODEL-VIEW-CONTROLLER ARCHITECTURE

Model-View-Controller is a programming paradigm created for interactive applications, or GUI enabled applications. It basically splits the code into 3 parts; the model, the view and the controller.

Model – contains the data and methods that manipulate the data to be used in an application. This is also referred to as the "business logic" of an application. The model can exist by itself, but it has no way of communicating with the user. It knows that it will be used by a view to display its data, but knows nothing about its view's structure.

View – contains the representation of the data in the model. This is the UI part. The view is mainly concerned with how will it present the model and how can a user interact with the model presented. The view needs a model and knows that a controller will be filled in to take care of interactions of the user, but knows nothing of the controller's structure.

Controller – contains the methods that are used for the communication of the user and the application. This basically connects the model and view. These are normally the events generated when a user performs an action in the view, to manipulate the model.

As seen in Figure 1, the solid lines indicate direct association, meaning that a component can use every resource the other component have, while the dashed lines indicate indirect association, meaning that a component notifies the other component in case of changes. [31]

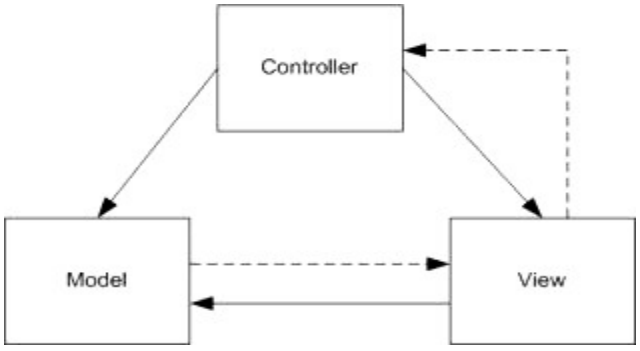

Figure 1: Model View Controller Diagram, XGG

The MVC design pattern has many advantages:

1. A Model can have multiple views, which means that different kinds of interfaces can be used to show a certain model, and the user can choose which of those interfaces he/she wants.
2. Since the 3 components are separated, each component can be developed almost independent of each other.
3. Changes in one component normally do not affect other components.
4. Bugs can be easily spotted since it can be isolated in any of the 3 components. This also greatly improves unit testing.

G. JAVA SWING

Swing is one of Java's UI toolkits. It is a set of GUI components with a pluggable look and feel. Swing components can have the look and feel of the Operating system it runs on such as Windows, Linux and Mac.

Swing is built on top of the Abstract Window Toolkit or AWT. Swing uses some components from AWT. Some layout managers and event listeners came directly from AWT. The difference between the two is that AWT uses the operating system native components such as buttons and checkboxes. This approach is usually not preferable because each operating system does not have the same implementation in their windowing system. Swing uses pure Java code when creating components so that they remain the same across different operating systems, with the exception of their look and feel. Swing components are referred as "lightweight" because they do not allocate operating system native windowing objects while AWT is referred as "heavyweight". [30]

H. JAVA BEANS

Java Beans is a component model made by Java to be a guide in writing reusable components. It used a set of rules, or design patterns like naming conventions, to make a class a bean. There are many tools and Java classes that are created that support beans. Many Java classes are beans, like all classes from Java Swing. [30]

The most basic characteristic of a Java Bean is the utilization of its properties. Properties are variables that are Variables are usually private / protected, and are accessed with methods starting with "set" and "get" followed by the variable name.

I. SIMPLE API FOR XML

The Simple API for XML (SAX) is a serial access parser API for XML. SAX provides a mechanism for reading data from an XML document. [32]

A parser which implements SAX functions as a sequential parser using event calls. The user defines a number of callback methods that will be called when events occur during parsing. The SAX events include:

* XML Text nodes
* XML Element nodes
* XML Processing Instructions
* XML Comments

Java has a built-in SAX Parser in their default API.

J. XML UI LANGUAGE (XUL)

XUL (pronounced as "zool") is an XML language that is used to describe the orientation and relationships of a certain user interface. It separates the code into two parts: the design and the logic. The UI design is placed in an XML file while the back-end code or the logic is in a separate programming language such as C# and Java or a scripting language like JavaScript. [8]

The XUL file looks almost like HTML. It can describe how a certain component would be displayed like its position, color, text etc. In a well-written XUL code, one can easily have an idea on how a UI looks.

An example of a simple XUL might look like this:

```
<window>
  <panel layout="vertical">
    <textfield id="togreet" />
    <button action="dogreet" color="red">
      Click to Greet!
    </button>
  </panel>
</window>
```

In the above text we can see that the UI is composed of a text field and a button inside a panel with a vertical layout. The button's color is red and has the text "Click to Greet!" We also know that a certain function named "dogreet" will be called when a user clicks the button.

K. DEFINITION OF TERMS

- Parser - is the process of analyzing a sequence of tokens to determine its grammatical structure with respect to a given formal grammar.
- Compiler – computer program that translates a computer language into another computer language.
- Plug in – is a computer program that interacts with a host application to provide a certain, usually very specific, function "on demand"
- Schema – a way to define the structure, content and, to some extent, the semantics of XML documents
- Document Type Definition – an XML schema language. It describes a set of rules that an XML file would have to follow
- Document Type Declaration – also called DOCTYPE. A tag that appears near the start of the XML document that binds an XML file to a Document Type Definition.
- XML Namespace – A unique vocabulary for tag names and attributes in an XML instance.
- Data Binding – It is the binding of two variables one from the back-end module and the other a property of a component, such that if one of them changes, the other variable is informed of the change.

IV. DESIGN AND IMPLEMENTATION

The first step is to define the XUL grammar that will be used throughout the proposed system. The following are guidelines and rules used to create a XUL grammar for this system:

1. The following components in the Swing UI toolkit (as of version 1.5) should be expressed in the XUL.

Containers:

Box	JApplet
JDesktopPane	JFrame
JInternalFrame	JPanel
JScrollPane	JSplitPane
JTabbedPane	JToolBar

Layout:

BorderLayout	BoxLayout
CardLayout	FlowLayout
GridBagLayout	GridLayout
OverlayLayout	

GUI components:

JButton	JCheckBox
JComboBox	JEditorPane
JFormattedTextField	JLabel
JList	JMenuBar
JMenu	JMenuItem
JCheckBoxMenuItem	JRadioButtonMenuItem
JSeparator	JPasswordField
JPopupMenu	JProgressBar
JRadioButton	JSlider
JSpinner	JTable
JTextArea	JTextField
JTextPane	JToggleButton
JTree	

2. The XUL should be as simple as possible, and would provide no features that can promote logic code inside the XUL, such as scripting inside the XUL code.

3. The XUL should support the Model-View-Controller. The Model is a class (a bean class is preferable for binding) or a database result that is referenced I the XUL file. The View is the XUL file's generated code and the Controller is the extended class with the abstract event methods filled.

4. The XUL should be able to express most of the graphical routines of Java's Graphics2D.

   All Shape classes that can be used in drawing components would be mapped and can be expressed in XUL.

Arc	CubicCurve
Ellipse	GeneralPath
Line	QuadCurve
Rectangle	RectangularShape
RoundRectangle	

5. The XUL file should be able to support writing Frames, Applets, Style sheets, and extending other Swing components.

6. The compiler will output 2 files. The first file is the "GUI" file, where all the code for outputting the interface is located. The other class is the "Event" file, where all the needed methods and variable declarations are located. This class extends from the GUI file. In Prototype mode, there is only 1 output file. The "GUI" file that also contains the main method, everything included in the "Event" file is ignored in this mode.

7. The GUI file contains two methods that are called in the constructor, "createComponents()" and "initialize()". The method "createComponents()" is where the code for creating all Swing components are located. The method "initialize()" is empty but is located before "createComponent()", it's purpose is to contain initialization code.

8. The compiler will only accept filenames ending with ".xml" or ".xgg". The XUL file need not be a valid XML document. The name of the class is the filename of the XUL file. The GUI file adds the suffix "GUI" to the file and the Event file uses the same name as the XUL file. For example, if the file name is "HelloWorld.xml", the "GUI" file name is "HelloWorldGUI.java" and the Event" file name is "HelloWorld.java".

9. Tag names and attribute names are case sensitive to provide a consistent look. Tag names start with uppercase letters. Attribute names will follow Java's convention of

starting with lowercase letters then making the next words start with uppercase letters (“treeNode”, “frontColor” etc.).

10. Swing components that represent a visible object (like buttons) are mapped by using their class names as names for tags and bean properties as its attributes. The attribute “id” refers to that object’s variable name. A component’s content is its text or label property, so <JButton text=”text” /> is equal to <JButton>text</JButton>. If the component does not have the text property, then that component cannot have content.

```
<JButton id=”btn” background=”red”>
  Hello
</JButton>
```

There are also special attributes in other Swing Components. The full list is included in the Appendix.

11. A tag inside a tag indicates that it is added in the container of the parent tag. For example, to make a JButton be contained inside a JPanel, you should put the <JButton> tag in between <JPanel> and </JPanel>.

Other components have special children tags. JComboBox and JList have the Item tag which corresponds to their children. The “value” attribute or the content is the value of the children. JTable have Column and Row tags. The column corresponds to the column in a table and the row is a row of data in the table. JTree have the Node tags. It is like the <Item> tag except it can have Nodes as children.

12. The root XML object can contain Models. Each XUL file has only one Model tag. The Model is used to point data bindings and variables to the XUL file. The Model tag can be a Java Class or a MySQL database query result. Special attribute values are needed to access the values of the model.

```
<JFrame>
  <Model path=”modelclass” />
</JFrame>
```

If the model is of type list or database, then you can only access the model's data under components whose parent have the attribute "model.loop" set to "true". There can be only one "model.loop" attribute in the whole XUL file. The children of the component with the "model.loop" attribute would be looped per item in the model.

13. Each attribute type in each element has a corresponding Java Type. Strings and ints are the most common. Primitive types such as int are declared as they are declared in Java files, like "12", or "5.0". Strings are declared normally, like "text". You can use constants as the values of the attributes, like "Color.red" and "KeyEvent.VK_A". Some Java types like Dimension and LayoutManagers, have special ways for expressing their values. The full lists of the special ways in expressing such values are included in the Appendix. Not all Java Type can be expressed directly, like ActionMap, or a custom type. These types relies on the special attribute types, like {var} and {late}.

Attribute contents have special parameters:

a) {var varname} – The attribute uses the value of the given variable. "varname" can only point to variables created using the 'id' attribute.

b) {late varname} - creates a variable named "varname" that is originally set to a default value (usually null) that must be filled out in the initialize() method

c) {bind bindtype varname} – The attribute binds with the given variable; the given variable must be a bound property (has a setter method with a ChangeListener) of a bean class given in the Model.

The bindtype can be "to", "from" or "both".

- to - when the model property changes, the element property will follow.
- from - when the element property changes, the model property will follow.
- both - when any one property changes, the other property will follow.

d) {modelvar varname} - points to a variable in the model with type class or list.

e) {modelmethod methodname} - points to a method in the model with type class or list. Note that it only accepts methods that return the required value and with no parameters.

f) {data columnname} – retrieves the data for the “columnname” in the current row for the model with type database.

14. Swing Layout managers are important to describe the orientation of the components. They are normally associated with containers like JPanel. All Swing Layout managers can be initialized as an attribute in a panel.

15. The Swing components can also have attributes that correspond to a method name of an event. An example of this is the “actionPerformed” method in the ActionListener class. The value of the attribute will be the name of the method in the event file. The method will be invoked in the same way as the “actionPerformed” method inside an ActionListener will be invoked by a swing component.

```
<JButton text=”text” actionPerformed=”doClick” />
Creates the method:
   public abstract void doClick(ActionEvent e);
```

16. To change the graphical routines of a component, a UI tag is inserted inside the component’s content. A UI has 2 child tag types. Component is the graphical routines for drawing a button. The Border is the default routine for drawing a border. They can have a “condition” attribute, if it is true then the routine is called. The attribute “contains” denotes that Swing considers a mouse on top of a component when it is over the painted area.

```
<JButton text=” Hello!”>
  <UI>
    <Component>
       <Paint type="fill"
         value="GradientPaint(0,0,white,getWidth(),
         getHeight(),black,true)"/>
       <Ellipse do=”fill" x="0" y="0" width="getWidth()"
```

```
            length="getLength()" contains="true" />
          <Text horizontal="center" vertical="center"
              value="{inner text}"/>
        </Component>
        <Border>
          <Eclispe do="draw" x="0" y="0" width=" getWidth()"
             length="getHeight()" />
        </Border>
      </UI>
    </JButton>
```

17. Styles are separate files that are used to define the properties of the components and are used like a CSS file. To use a Style, reference their url in the top level container.

```
 Filename: styles.xml
<Styles>
  <JButton background=”red”/>
</Styles>

<JFrame>
   <Style path=”styles.xml”/>
   <!--code -->
</JFrame>
```

Once the grammar for XGG has been established, the next step is to create the compiler of the XUL that has been created. We will first transform the XUL code into an internal format and then convert it to Java code.

We will use the Simple API for XML or SAX to parse the file to convert it to an internal format, which will handle the structure of a Java file like imports, methods, declarations etc. The internal format consists of a main class that contains the Model, Styles and the list of element tags. It also holds the compiler options, which would be used in outputting the code.

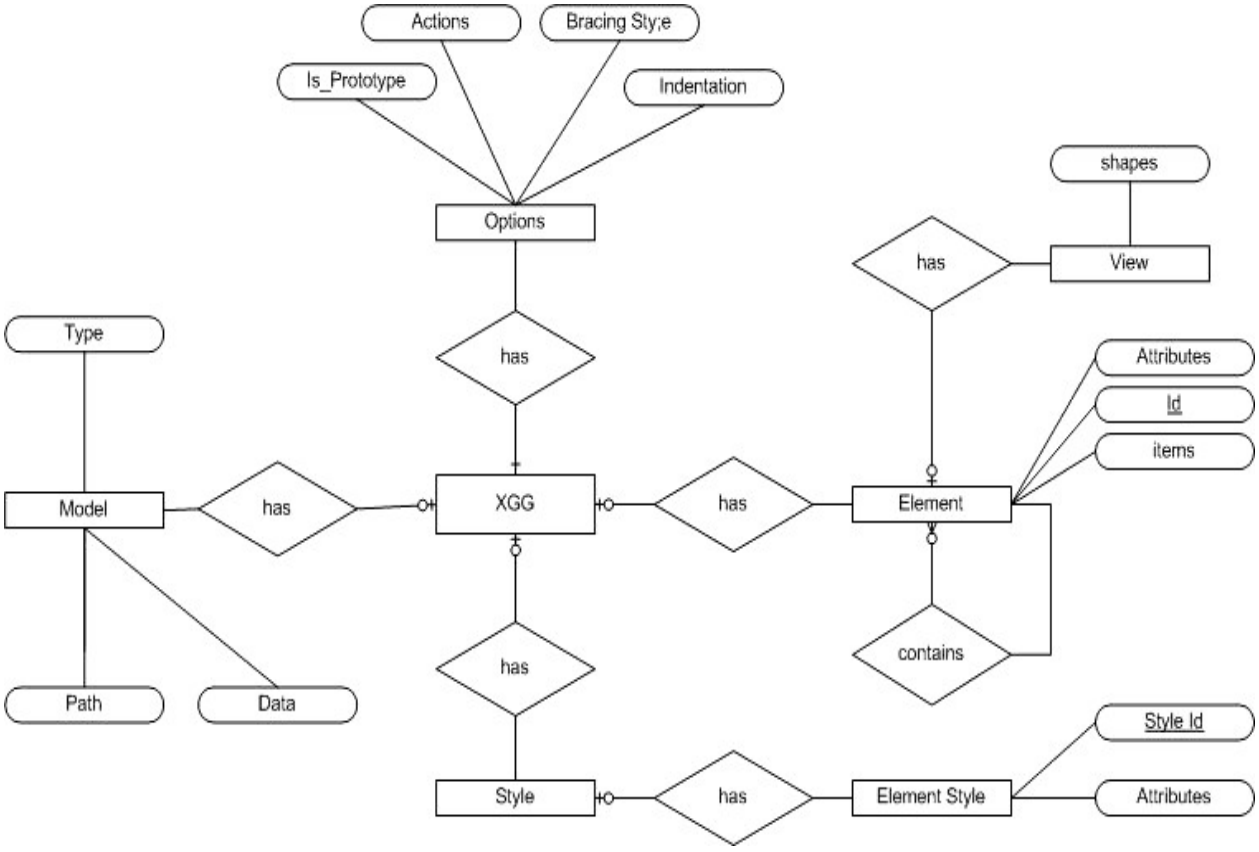

Figure 2: ERD of the Internal Format, XGG

The creation of the internal format has two main phases. The first phase is SAX parsing. The type of every tag (Element, Model, Styles) is determined and added to the internal format accordingly. Needed imports, variables and events are determined in this phase, syntax errors are also detected in this phase. The result of this phase is a raw internal format (see Figure 3).

The raw internal format is then processed to ensure that each data can create a valid Java code. Optimizations and Formatting in code and are handled in this phase. Error in structures like adding two children in a JScrollPane and attribute parsing errors like in LayoutManagers are detected in this phase.

After completing the internal format, it's method to output Java code is called and is stored in the output file. Event files are also checked if it needs to be updated.

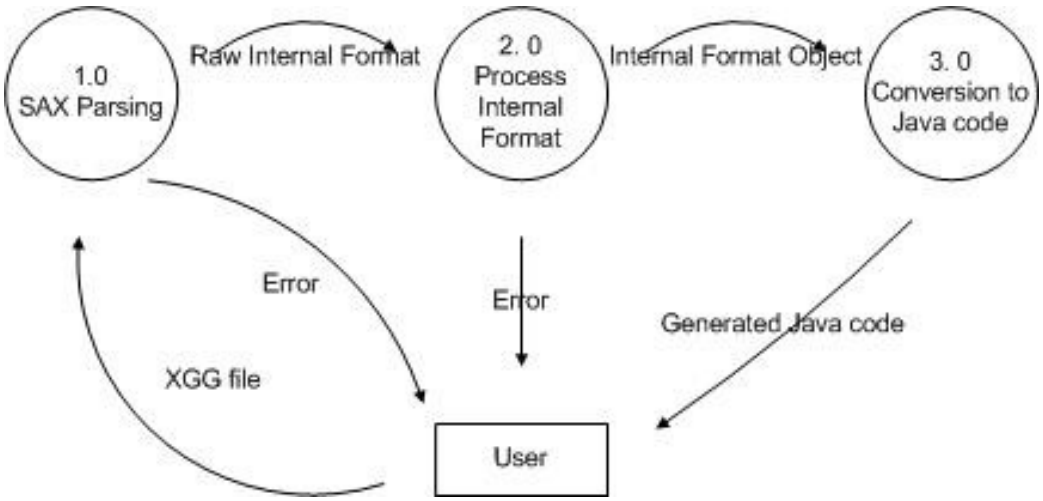

Figure 3: DFD of the Compiler, XGG

V. RESULTS

When you run the command line compiler without inputting an argument, it will display its default help screen. It shows some basic information, instructions on how to use it and the different command line options available.

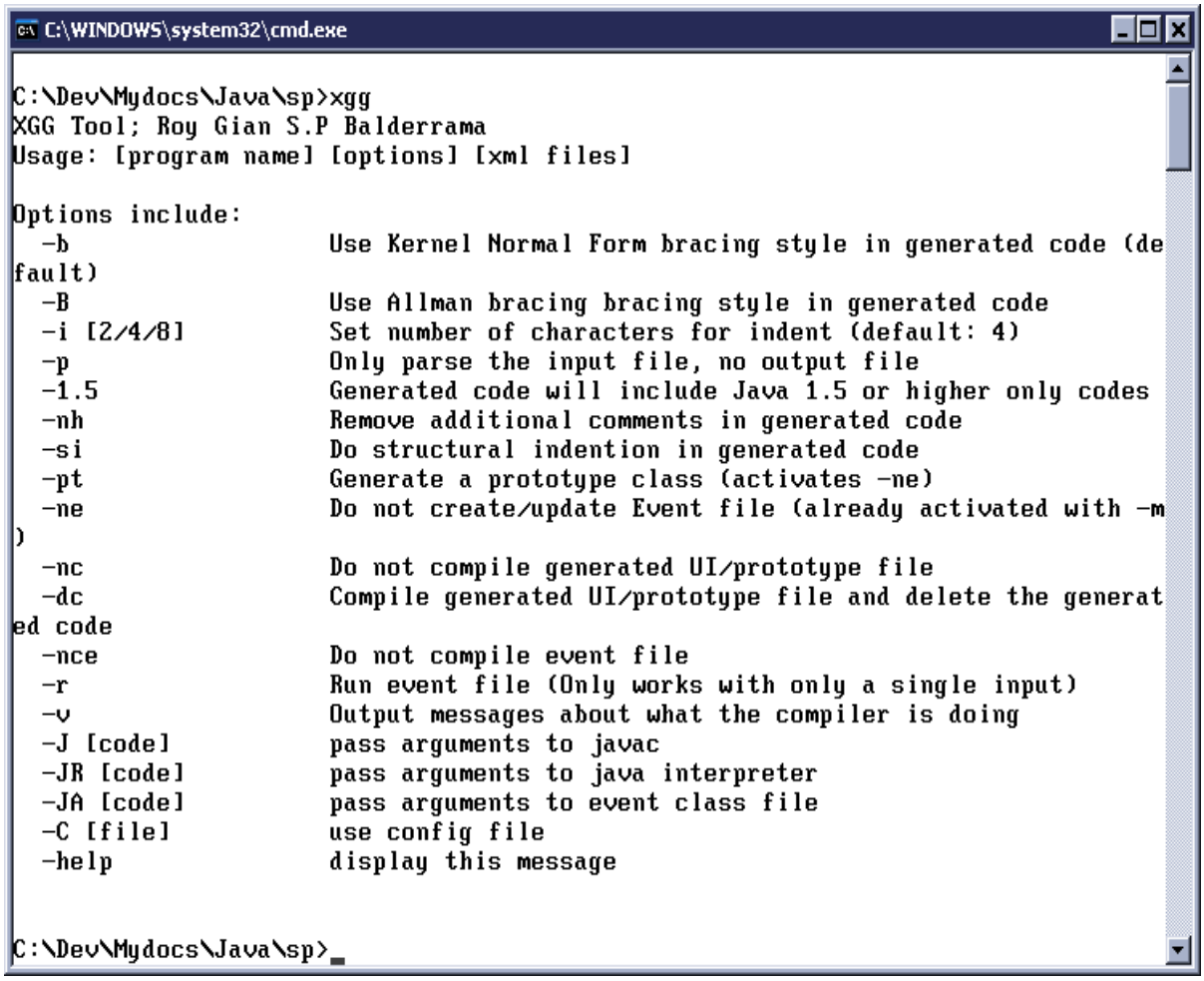

Figure 4: Command Line Compiler Default Message, XGG

When you ran the command line compiler with a file as an argument, the compilation process will start. Errors, warnings and notifications will be outputted. Errors are in the format of “Error: file: line number, column number: error message”. Shown on Figure 5 is a sample output of an error. The error is from a tag where the attribute “layout” is misspelled as “layou t”. XML attribute name does not have attribute names, and spaces after an attribute name is usually followed by the ‘=’ character.

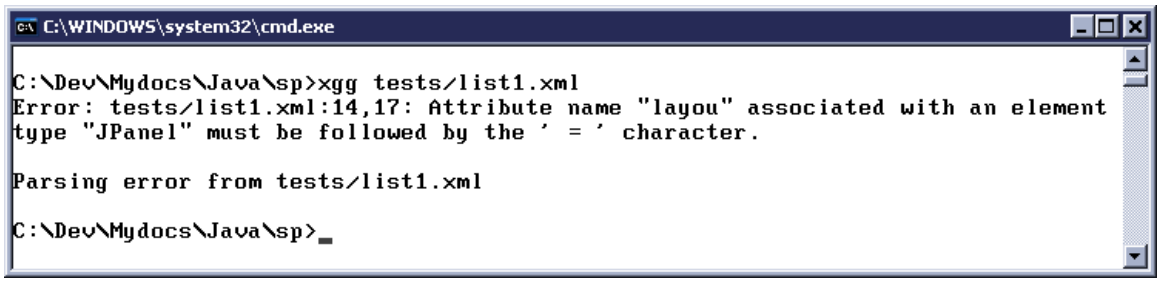

Figure 5: Error Encountered in Command Line Compiler, XGG

In Figure 6, the error has been fixed and the compilation is finished. The default options states that the generated GUI and Event file is automatically compiled. The XGG compiler alerts the user that it is running the Java compiler "javac" and also outputs its messages. "Operation complete" will be displayed once the compiler is successful in processing the file.

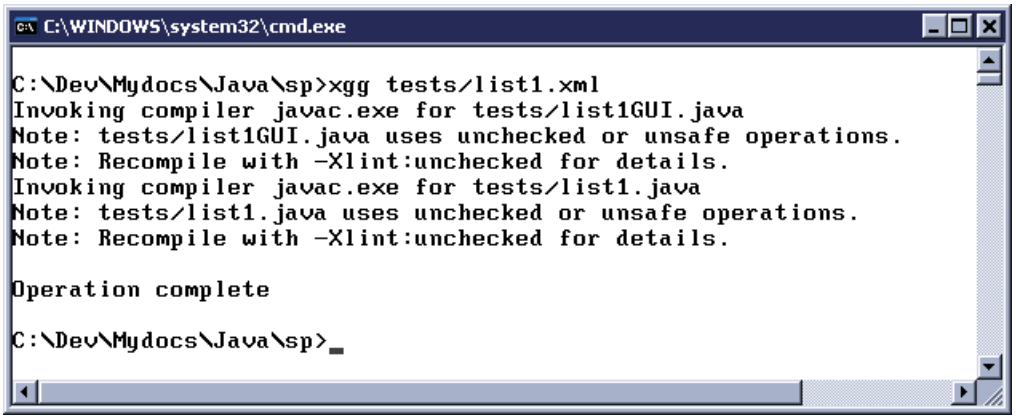

Figure 6: Successful Operation of the Command Line Compiler, XGG

The GUI front end is shown in Figure 7. The user can browse or drag a file to the screen, select the compiler options, then start the compilation. The front end can do everything that the command line compiler can do. The front end captures all outputs of the compiler and the compiler program if the run option was chosen to a text area. The compiler options are saved when the user exits the front end and is loaded when the front end is started again.

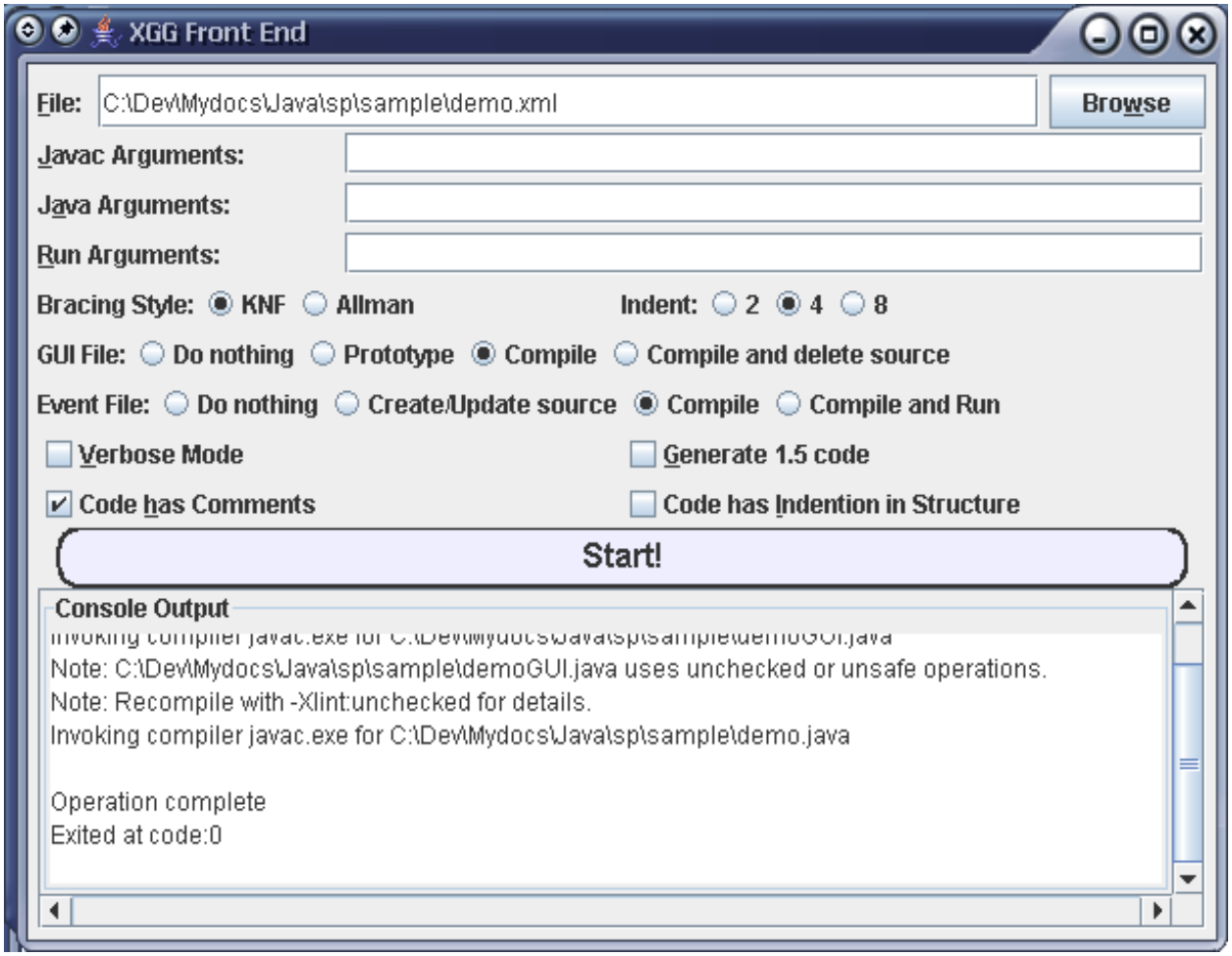

Figure 7: GUI Front End, XGG

The simple plug-in for Eclipse is shown in Figure 8. It provides syntax highlighting for XML, extra tab to switch between the XGG file and the GUI file, tree view of the XGG file and automation of the compilation, errors are also shown in the problems view of Eclipse.

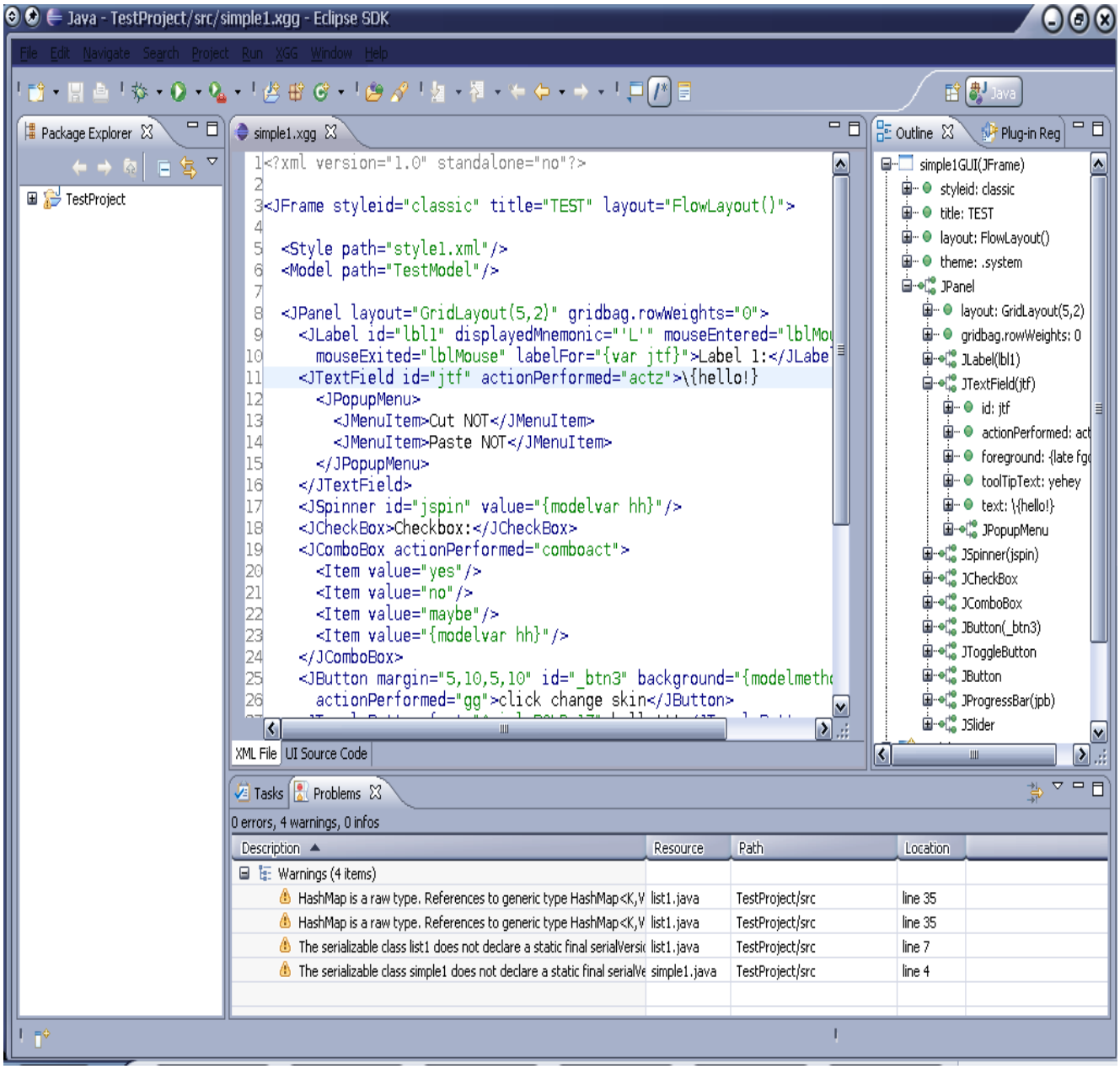

Figure 8: Simple Plug-in for Eclipse, XGG

A simple XGG file is displayed on Figure 9. The file is named “BasicTest.xml”. As you can see, through the expressiveness of xml, the frame consists of a label, textfield, then a button. The button has an event called “doClick”. The file is compiled and the output GUI file is illustrated on Figure 10. The Event file, with only the doClick method modified, is shown on Figure 11. Finally we have the program result, on Figure 12. This simple example shows how much effort is reduced through the use of the XGG compiler.

```
<JFrame title="Sample" layout="FlowLayout()">
    <JLabel>Label:</JLabel>
     <JTextField id="tfield" columns="10"/>
    <JButton actionPerformed="doClick">Click</JButton>
</JFrame>
```

Figure 9: Sample Input “BasicTest.xml”, XGG

BasicTestGUI.java

```
34          tfield = new JTextField();
35          tfield.setColumns(10);
36          add(tfield);
37
38          //JButton '__JButton1' with start tag ending at (4, 38)
39          JButton __JButton1 = new JButton();
40          __JButton1.setText("Click");
41          //__JButton1.actionPerformed = doClick;
42          __JButton1.addActionListener(
43              new ActionListener() {
44
45                  public void actionPerformed(ActionEvent e) {
46                      doClick(e);
47                  }
48              }
49          );
50          add(__JButton1);
51
52          pack();
53          setVisible(true);
54          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
55      }
56
57      public abstract void doClick(ActionEvent e);
58  }
```

Figure 10: GUI File “BasicTestGUI.java” in a Text Editor, XGG

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class BasicTest extends BasicTestGUI {

    public BasicTest() {
        super();
```

```
    }

    public static void main(String[] args) {
        //MAIN PROGRAM CODE HERE
        new BasicTest();
    }

    public void initialize() {
        //PUT INITIALIZATION HERE
    }

    public void doClick(ActionEvent e) {
     JOptionPane.showMessageDialog(null,"text in Textfield
is"+tfield.getText());
    }
}
```

Figure 11: Source Code of "BasicTest.java", XGG

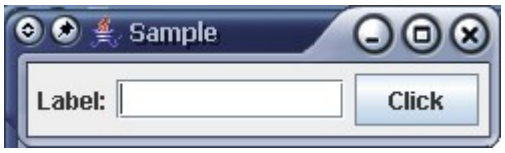

Figure 12: Output Program of "BasicTest.java", XGG

VI. DISCUSSION

The XGG compiler is a simple yet powerful XUL in Java. It takes an XML file as an input and outputs properly indented Java code that creates the UI layout specified in the XML file, along with another file containing code that is needed to add events for the UI.

The compiler supports features commonly found in XUL like style sheets, binding of classes, and some unique features like database support and customization of drawing functions of Swing components.

The XGG compiler has many advantages over the common XUL implementations in Java. Unlike in the run time XUL implementations, you do not need to include the XML file in the application. It also runs faster since UI screens made with XGG does not spend time to parse and use slow Java routines (reflection) each time it starts. XGG does not require applications compiled by it to include a runtime package, unlike other XULs. This makes it ideal to be used on small-scale applications. Finally, in XGG you can use extended Swing components while other XULs cannot.

XGG does not have a schema or DTD, unlike normal XML based languages. The reason for this is because XGG accepts any supported Swing Components and all classes derived from these Components. Such custom components can also have their own attributes set through the use of Bean properties. Such extensibility cannot be expressed in a Schema or a DTD.

A disadvantage that XGG had against other XUL is that it is severely limited by the Swing API. Other XUL extends Swing and add custom layout managers and new and original components, while in XGG you are stuck with the basic components and layout managers.

Against Visual GUI builders, one advantage of the XGG is that the separation of logic and design code is enforced. If you are not careful with builders, it is easy for your code to be cluttered. XGG also does not hard code the coordinates of the components and is normally better when creating windows that is frequently resized. Unlike in builders, you can easily create a dynamic layouting of components in XGG using list and database models.

A disadvantage of XGG when compared to builders is that less knowledge is needed to work with builders than with XGG. Visualization is also easier since builders are WYSIWYG in their approach in creating the UI.

The grammar design of XGG is fairly simple, with familiar Java Swing classes and properties as the valid tag and attributes in an XGG file. It does not offer advanced XML features unlike most XULs, like namespaces and CDATA scripting. In fact XGG files feels like HTML files without the javascript. This simplicity makes it a good choice if the user wants to start learning the basics and to feel XUL since it is now being used in some major technologies like Adobe Flex and Microsoft's WPF.

As a learning tool, XGG can be quite effective, although it is not recommended for users with no knowledge of Swing especially with Java. Users with basic knowledge with Swing can check the sample XGG files and check the commented GUI file to learn more. Users, over time, can easily visualize the layouts that would be used in every UI, meaning that they could be very well efficient with hand coding as well. They may also develop a habit to separate logic code and design code, one of the main features of Model View Controller Paradigm, not only in Java but also in other developmental environments.

VII. CONCLUSION

The XML GUI Generator (XGG) is an effective approach in creating and/or learning how to create Swing components screens. The user, if familiar to HTML, can easily get used to in creating Swing Screens. It also alleviates problems in code due to the merging of code for creating the GUI and the logic code. The user may also learn on using the MVC architecture when the user uses the Model classes in creating programs.

VIII. RECOMMENDATIONS

- Re-implement the SAX engine to give a more accurate location of errors in the XUL file.
- Add more supported Swing Components like Timer, JDialog, JWindow etc.
- Add the following features in the plug-in:
    - Provide templates for different customized Components

- Allow the user to select a portion or edit the XUL file using it's tree view.
- Implement a visual editor

IX. BIBLIOGRAPHY

[1] Standard Generalized Markup Language, [http://en.wikipedia.org/wiki/Sgml]

[2] XML, [http://en.wikipedia.org/wiki/Xml]

[3] Scalable Vector Graphics (SVG), [http://www.w3.org/Graphics/SVG/]

[4] Cover Pages: Simple Object Access Protocol (SOAP), [http://xml.coverpages.org/soap.html]

[5] T. Ostermann, H. Zillman, C. Raak et al. “CAMbase – A XML-based bibliographical database on Complementary and Alternative Medicine (CAM)”, Biomedical Digital Libraries 2007.

[6] The Forms Working Group, [http://www.w3.org/MarkUp/Forms/]

[7] XML User Interface Language (XUL) Project, [http://www.mozilla.org/projects/xul/]

[8] Open XUL Alliance - Creating A Rich Internet For Everyone, [http://xul.sourceforge.net/]

[9] A Swing Architecture Overview, [http://java.sun.com/products/jfc/tsc/articles/architecture/]

[10] G. Wilkes IV. “XML and the New Design Regime: Disputes Between Designers, Application Developers, Authors, and Readers in Changing Technological Conditions and Perceptions of Social and Professional Need”, ACM Journal of Computer Documentation May 2002/Vol. 26, No. 2.

[11] R. Buragapu. “Emerging Web Technologies - XML Based Rich Clients.”

[12] M. Rees. “Evolving the Browser Towards a Standard User Interface Architecture”, Australian Computer Society Inc. 2001.

[13] Smith A, Mahoney A., Rydberg-Cox JA. “Management of XML Documents in an Integrated Digital Library”.

[14] H. Jiang,H. Lu et al. “Path materialization revisited: an efficient storage model for XML data”, ACM International Conference Proceeding Series; Vol. 18.

[15] J. White, B. Kolpackov et al. “Reducing Application Code Complexity With Vocabulary-Specific XML Language Bindings”, 43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA.

[16] J. Wusteman. “About XML: from Ghostbusters to libraries: the power of XUL”, Library Hi Tech, Volume 23, Number 1, 2005, pp. 118-129(12).

[17] M. Abrams, C. Phanouriou. “UIML: an appliance-independent XML user interface language”, May 2000.

[18] S. Ahmed, G. Ashrafa, “Model-based user interface engineering with design patterns”, December 2006.

[19] S. Morse, C. Anderson. “Introducing Application Design And Software Engineering Principles In Introductory CS Courses: Model-View-Controller Java Application Framework”, Consortium for Computing Sciences in Colleges, 2004.

[20] S. Hansen, T. Fossum. “Refactoring Model-View-Controller”, Consortium for Computing Sciences in Colleges, 2005.

[21] XUL Tutorial – MDC, [http://developer.mozilla.org/en/docs/XUL_Tutorial]

[22] Luxor - XML UI Language (XUL) Toolkit, [http://luxor-xul.sourceforge.net/]

[23] Thinlet, [http://www.thinlet.com/index.html]

[24] SwiXML, Generate javax.swing at runtime based on XML, [http://www.swixml.org/]

[25] Windows Presentation Foundation, [http://msdn2.microsoft.com/en-us/netframework/aa663326.aspx]

[26] HTML, [http://en.wikipedia.org/wiki/Html]

[27] E. Harold. XML Bible, 2nd ed. Wiley Publishing, 2001.

[28] W3C Recommendation. Extensible Markup Language (XML) 1.0 (Fourth Edition), 2006.

[29] S. O’Brien. Turbo Pascal 6: The Complete Reference. McGraw-Hill, 1991.

[30] Java Reference Documentation, [http://java.sun.com/reference/docs/index.html]

[31] S. Burbeck, Ph.D: “Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)”, 1992.

[32] Simple API for XML, [http://en.wikipedia.org/wiki/Simple_API_for_XML]

X. APPENDIX

Appendix A

List of Supported Component Tags and Attributes

All Components:

Attribute	Java Type	Comments
background	java.awt.Color	
cursor	java.awt.Cursor	
enabled	boolean	
focusCycleRoot	boolean	
focusTraversalPolicy	java.awt.FocusTraversalPolicy	
focusTraversalPolicyProvider	boolean	
focusable	boolean	
font	java.awt.Font	
foreground	java.awt.Color	
hasmain	java.awt.boolean	Use only in root tags. If true then the event file has a main method. Initially true for JFrame and JApplet.
id	java.lang.String	The variable name of the component.
layout	java.awt.LayoutManager	Pointless to use in Non-container components like JButton and JTextField
model.data	(varies)	If the component is created by XGG and has a model, then this attribute lets you pass a model object to the component.
model.loop	boolean	Always "true".
name	java.lang.String	This does not refer to the variable name of the compnent. Components use this property as a sort of tag/extra property. "id" refers to the name of the variable.
visible	boolean	

Attribute – Event	Listener
ancestorMoved	HierarchyBoundsListener
ancestorResized	HierarchyBoundsListener
caretPositionChanged	InputMethodListener
componentAdded	ContainerListener
componentHidden	ComponentListener
componentMoved	ComponentListener
componentRemoved	ContainerListener
componentResized	ComponentListener
componentShown	ComponentListener
focusGained	FocusListener
focusLost	FocusListener
hierarchyChanged	HierarchyListener
inputMethodTextChanged	InputMethodListener
keyPressed	KeyListener
keyReleased	KeyListener
keyTyped	KeyListener
mouseClicked	MouseListener
mouseDragged	MouseMotionListener
mouseEntered	MouseListener
mouseExited	MouseListener
mouseMoved	MouseMotionListe

	ner
mousePressed	MouseListener
mouseReleased	MouseListener
mouseWheelMove d	MouseWheelListe ner
propertyChange	PropertyChangeLi stener

All Components except Top Level Containers (JFrame and JApplet):

Attribute	**Java Type**	**Comments**
actionMap	javax.swing.ActionM ap	
alignmentX	float	
alignmentY	float	
autoscrolls	boolean	
border	javax.swing.border. Border	
componentPopupMe nu	javax.swing.JPopup Menu	Only available starting at Java 1.5
constraint	int	Used by the parent Container when adding this Component
constraint.anchor	int	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.fill	int	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.gridheight	int	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.gridwidth	int	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.gridx	int	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.gridy	int	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.insets	java.awt.Insets	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.ipadx	int	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.ipady	int	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.weightx	double	Used by GridBagConstraints if the parent's layout is GridBagLayout
constraint.weighty	double	Used by GridBagConstraints if the parent's layout is GridBagLayout
debugGraphicsOptio ns	int	
doubleBuffered	boolean	
inheritsPopupMenu	boolean	Only available starting at Java 1.5
inputVerifier	javax.swing.InputVe rifier	
maximumSize	java.awt.Dimension	
minimumSize	java.awt.Dimension	
nextFocusableComp onent	java.awt.Component	
opaque	boolean	
preferredSize	java.awt.Dimension	
requestFocusEnable d	boolean	
tab.title	java.lang.String	The title of this Component if under a JTabbedPane
tab.tip	java.lang.String	The tool tip of this Component if under a

		JTabbedPane
tab.icon	javax.swing.Icon	The Icon of this Component if under a JTabbedPane
toolTipText	java.lang.String	
transferHandler	javax.swing.TransferHandler	
verifyInputWhenFocusTarget	boolean	

Attribute – Event	**Listener**
ancestorAdded	AncestorListener
ancestorMoved	AncestorListener
ancestorRemoved	AncestorListener
vetoableChange	VetoableChangeListener

All Containers:

Attribute	**Java Type**	**Comments**
gridbag.columnWeights	double[]	If the layout is a GridBagLayout, use this to set the columnWeights of the layout
gridbag.columnWidths	double[]	If the layout is a GridBagLayout, use this to set the columnWidths of the layout
gridbag.rowHeights	int[]	If the layout is a GridBagLayout, use this to set the rowHeights of the layout
gridbag.rowWeights	int[]	If the layout is a GridBagLayout, use this to set the rowWeights of the layout

AbstractButton (JButton, JCheckBox, JMenu, JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem, JRadioButton and JToggleButton's parent class):

Attribute	**Java Type**	**Comments**
UI	javax.swing.plaf.ButtonUI	
action	javax.swing.Action	
actionCommand	java.lang.String	
borderPainted	boolean	
contentAreaFilled	boolean	
disabledIcon	javax.swing.Icon	
disabledSelectedIcon	javax.swing.Icon	
displayedMnemonicIndex	int	
focusPainted	boolean	
horizontalAlignment	int	SwingConstants.RIGHT, SwingConstants.LEFT , SwingConstants.CENTER , SwingConstants.LEADING , SwingConstants.TRAILING
horizontalTextPosition	int	SwingConstants.RIGHT, SwingConstants.LEFT , SwingConstants.CENTER , SwingConstants.LEADING , SwingConstants.TRAILING
icon	javax.swing.Icon	
iconTextGap	int	
label	java.lang.String	
margin	java.awt.Insets	
mnemonic	int	

model	javax.swing.ButtonModel	
multiClickThreshhold	long	
pressedIcon	javax.swing.Icon	
rolloverEnabled	boolean	
rolloverIcon	javax.swing.Icon	
rolloverSelectedIcon	javax.swing.Icon	
selected	boolean	
selectedIcon	javax.swing.Icon	
text	java.lang.String	
verticalAlignment	int	SwingConstants.CENTER, SwingConstants.TOP SwingConstants.BOTTOM
verticalTextPosition	int	SwingConstants.CENTER, SwingConstants.TOP SwingConstants.BOTTOM

Attribute – Event	**Listener**
actionPerformed	ActionListener
itemStateChanged	ItemListener
stateChanged	ChangeListener

JTextComponent (JEditorPane, JFormattedTextField, JPasswordField, JTextArea, JTextField and JTextPane's parent class):

Attribute	**Java Type**	**Comments**
UI	javax.swing.plaf.TextUI	
caret	javax.swing.text.Caret	
caretColor	java.awt.Color	
caretPosition	int	
componentOrientation	java.awt.ComponentOrientation	ComponentOrientation.LEFT_TO_RIGHT, ComponentOrientation.RIGHT_TO_LEFT, ComponentOrientation.UNKNOWN
disabledTextColor	java.awt.Color	
document	javax.swing.text.Document	
dragEnabled	boolean	
editable	boolean	
focusAccelerator	char	
highlighter	javax.swing.text.Highlighter	
keymap	javax.swing.text.Keymap	
margin	java.awt.Insets	
navigationFilter	javax.swing.text.NavigationFilter	
selectedTextColor	java.awt.Color	
selectionColor	java.awt.Color	
selectionEnd	int	
selectionStart	int	
text	java.lang.String	

Attribute – Event	**Listener**

caretUpdate	CaretListener

Box:

Attribute	Java Type	Comments
axis	int	BoxLayout.X_AXIS, BoxLayout.Y_AXIS, BoxLayout.LINE_AXIS or BoxLayout.PAGE_AXIS.

JApplet:

Attribute	Java Type	Comments
JMenuBar	javax.swing.JMenuBar	
contentPane	java.awt.Container	
glassPane	java.awt.Component	
layeredPane	javax.swing.JLayeredPane	
stub	java.applet.AppletStub	
theme	java.lang.String	The Fully Qualified Class Name of the Swing Plaf Theme. Special Values: .system – The native system Look .cross – Java's cross platform look

JButton (child of AbstractButton):

Attribute	Java Type	Comments
defaultCapable	boolean	

JCheckBox (child of AbstractButton):

Attribute	Java Type	Comments
borderPainteDFlat	boolean	

JCheckBoxMenuItem (child of AbstractButton):

Attribute	Java Type	Comments
accelerator	javax.swing.KeyStroke	
armed	boolean	
state	boolean	

Attribute – Event	Listener
menuDragMouseDragged	MenuDragMouseListener
menuDragMouseEntered	MenuDragMouseListener
menuDragMouseExited	MenuDragMouseListener

menuDragMouseReleased	MenuDragMouseListener
menuKeyPressed	MenuKeyListener
menuKeyReleased	MenuKeyListener
menuKeyTyped	MenuKeyListener

JComboBox:

Attribute	Java Type	Comments
UI	javax.swing.plaf.ComboBoxUI	
action	javax.swing.Action	
actionCommand	java.lang.String	
editable	boolean	
editor	javax.swing.ComboBoxEditor	
keySelectionManager	javax.swing.JComboBox.KeySelectionManager	
lightWeightPopupEnabled	boolean	
maximumRowCount	int	
model	javax.swing.ComboBoxModel	
popupVisible	boolean	
prototypeDisplayValue	java.lang.Object	
renderer	javax.swing.ListCellRenderer	
selectedIndex	int	
selectedItem	java.lang.Object	

Attribute – Event	Listener
popupMenuCanceled	PopupMenuListener
popupMenuWillBecomeInvisible	PopupMenuListener
popupMenuWillBecomeVisible	PopupMenuListener

JDesktopPane:

Attribute	Java Type	Comments
UI	javax.swing.plaf.DesktopPaneUI	
desktopManager	javax.swing.DesktopManager	
dragMode	int	LIVE_DRAG_MODE, OUTLINE_DRAG_MODE
selectedFrame	javax.swing.JInternalFrame	

JEditorPane (child of JTextComponent):

Attribute	Java Type	Comments
contentType	java.lang.String	
editorKit	javax.swing.text.EditorKit	
page	java.net.URL	

Attribute – Event	Listener
hyperlinkUpdate	HyperlinkListener

JFormattedTextField (child of JTextComponent):

Attribute	Java Type	Comments
action	javax.swing.Action	
actionCommand	java.lang.String	
columns	int	
focusLostBehavior	int	
formatterFactory	javax.swing.JFormattedTextField.AbstractFormatterFactory	
horizontalAlignment	int	LEFT , CENTER, RIGHT, LEADING, TRAILING
scrollOffset	int	
value	java.lang.Object	

JFrame:

Attribute	Java Type	Comments
JMenuBar	javax.swing.JMenuBar	
alwaysOnTop	boolean	
contentPane	java.awt.Container	
cursor	int	
defaultCloseOperation	int	WindowConstants.DO_NOTHING_ON_CLOSE WindowConstants.HIDE_ON_CLOSE WindowConstants.DISPOSE_ON_CLOSE EXIT_ON_CLOSE
extendedState	int	
focusableWindowState	boolean	
glassPane	java.awt.Component	
iconImage	java.awt.Image	
layeredPane	javax.swing.JLayeredPane	
locationByPlatform	boolean	
locationRelativeTo	java.awt.Component	
maximizedBounds	java.awt.Rectangle	
menuBar	java.awt.MenuBar	
resizable	boolean	
size	java.awt.Dimension	
State	int	
theme	java.lang.String	The Fully Qualified Class Name of the Swing Plaf Theme. Special Values: .system – The native system Look .cross – Java's cross platform look
title	java.lang.String	
undecorated	boolean	

Attribute – Event	Listener
windowActivated	WindowListener
windowClosed	WindowListener
windowClosing	WindowListener
windowDeactivated	WindowListener
windowDeiconified	WindowListener
windowGainedFocus	WindowFocusListener
windowIconified	WindowListener
windowLostFocus	WindowFocusListener
windowOpened	WindowListener
windowStateChanged	WindowStateListener

JInternalFrame:

Attribute	Java Type	Comments
JMenuBar	javax.swing.JMenuBar	
UI	javax.swing.plaf.InternalFrameUI	
closable	boolean	
closed	boolean	
contentPane	java.awt.Container	
defaultCloseOperation	int	WindowConstants.DO_NOTHING_ON_CLOSE WindowConstants.HIDE_ON_CLOSE WindowConstants.DISPOSE_ON_CLOSE
desktopIcon	javax.swing.JInternalFrame.JDesktopIcon	
frameIcon	javax.swing.Icon	
glassPane	java.awt.Component	
icon	boolean	
iconifiable	boolean	
layer	int	
layeredPane	javax.swing.JLayeredPane	
maximizable	boolean	
maximum	boolean	
menuBar	javax.swing.JMenuBar	
normalBounds	java.awt.Rectangle	
resizable	boolean	
size	java.awt.Dimension	
title	java.lang.String	

Attribute – Event	Listener
internalFrameActivated	InternalFrameListener
internalFrameClosed	InternalFrameListener
internalFrameClosing	InternalFrameListener
internalFrameDeactivated	InternalFrameListener
internalFrameDeiconified	InternalFrameListener
internalFrameIcon	InternalFrameLi

ified	stener
internalFrameOpe	InternalFrameLi
ned	stener

JLabel:

Attribute	Java Type	Comments
UI	javax.swing.plaf. LabelUI	
disabledIcon	javax.swing.Icon	
displayedMnemon ic	int	The int value of a character, use '(character)' (w/ the quotes) as input
displayedMnemon icIndex	int	
horizontalAlignme nt	int	SwingConstants.RIGHT, SwingConstants.LEFT , SwingConstants.CENTER , SwingConstants.LEADING , SwingConstants.TRAILING
horizontalTextPosi tion	int	SwingConstants.RIGHT, SwingConstants.LEFT , SwingConstants.CENTER , SwingConstants.LEADING , SwingConstants.TRAILING
icon	javax.swing.Icon	
iconTextGap	int	
labelFor	java.awt.Compon ent	
text	java.lang.String	
verticalAlignment	int	SwingConstants.CENTER, SwingConstants.TOP SwingConstants.BOTTOM
verticalTextPositio n	int	SwingConstants.CENTER, SwingConstants.TOP SwingConstants.BOTTOM

JList:

Attribute	Java Type	Comments
UI	javax.swing.plaf.ListUI	
cellRenderer	javax.swing.ListCellRe nderer	
dragEnabled	boolean	
fixedCellHeigh t	int	
fixedCellWidth	int	
layoutOrientati on	int	VERTICAL, HORIZONTAL_WRAP, VERTICAL_WRAP
listData	[Ljava.lang.Object;	
model	javax.swing.ListModel	
prototypeCellV alue	java.lang.Object	
selectedIndex	int	
selectedIndice s	Int[]	
selectionBackg round	java.awt.Color	
selectionForeg round	java.awt.Color	
selectionMode	int	ListSelectionModel.SINGLE_SELECTION ListSelectionModel.SINGLE_INTERVAL_SELECTION ListSelectionModel.MULTIPLE_INTERVAL_SELECTION
selectionModel	javax.swing.ListSelecti onModel	
valueIsAdjustin g	boolean	
visibleRowCou	int	

nt

Attribute – Event	Listener
valueChanged	ListSelectionListener

JMenuBar:

Attribute	Java Type	Comments
UI	javax.swing.plaf.MenuBarUI	
borderPainted	boolean	
helpMenu	javax.swing.JMenu	
margin	java.awt.Insets	
selectionModel	javax.swing.SingleSelectionModel	

JMenu (child of AbstractButton):

Attribute	Java Type	Comments
accelerator	javax.swing.KeyStroke	
armed	boolean	
componentOrientation	java.awt.ComponentOrientation	ComponentOrientation.LEFT_TO_RIGHT, ComponentOrientation.RIGHT_TO_LEFT, ComponentOrientation.UNKNOWN
delay	int	
popupMenuVisible	boolean	

Attribute – Event	Listener
menuCanceled	MenuListener
menuDeselected	MenuListener
menuDragMouseDragged	MenuDragMouseListener
menuDragMouseEntered	MenuDragMouseListener
menuDragMouseExited	MenuDragMouseListener
menuDragMouseReleased	MenuDragMouseListener
menuKeyPressed	MenuKeyListener
menuKeyReleased	MenuKeyListener
menuKeyTyped	MenuKeyListener
menuSelected	MenuListener

JMenuItem (child of AbstractButton):

Attribute	Java Type	Comments
accelerator	javax.swing.KeyStroke	
armed	boolean	

Attribute – Event	Listener
menuDragMouseDragged	MenuDragMouseListener

menuDragMouseEntered	MenuDragMouseListener
menuDragMouseExited	MenuDragMouseListener
menuDragMouseReleased	MenuDragMouseListener
menuKeyPressed	MenuKeyListener
menuKeyReleased	MenuKeyListener
menuKeyTyped	MenuKeyListener

JPasswordField (child of JTextComponent):

Attribute	Java Type	Comments
action	javax.swing.Action	
actionCommand	java.lang.String	
columns	int	
echoChar	char	
horizontalAlignment	int	LEFT , CENTER, RIGHT, LEADING, TRAILING
scrollOffset	int	

JPopupMenu:

Attribute	Java Type	Comments
UI	javax.swing.plaf.PopupMenuUI	
borderPainted	boolean	
invoker	java.awt.Component	
label	java.lang.String	
lightWeightPopupEnabled	boolean	
selected	java.awt.Component	
selectionModel	javax.swing.SingleSelectionModel	

Attribute – Event	Listener
menuKeyPressed	MenuKeyListener
menuKeyReleased	MenuKeyListener
menuKeyTyped	MenuKeyListener
popupMenuCanceled	PopupMenuListener
popupMenuWillBecomeInvisible	PopupMenuListener
popupMenuWillBecomeVisible	PopupMenuListener

JProgressBar:

Attribute	Java Type	Comments
UI	javax.swing.plaf.ProgressBarUI	
borderPainted	boolean	
indeterminate	boolean	

maximum	int	
minimum	int	
model	javax.swing.BoundedRangeModel	
orientation	int	HORIZONTAL, VERTICAL
string	java.lang.String	
stringPainted	boolean	
value	int	

JRadioButton (child of AbstractButton):

Attribute	Java Type	Comments
group	javax.swing.ButtonGroup	The ButtonGroup that the JRadioButton registers to.

JRadioButtonMenuItem (child of AbstractButton):

Attribute	Java Type	Comments
accelerator	javax.swing.KeyStroke	
armed	boolean	
group	javax.swing.ButtonGroup	The ButtonGroup that the JRadioButton registers to.

Attribute – Event	Listener
menuDragMouseDragged	MenuDragMouseListener
menuDragMouseEntered	MenuDragMouseListener
menuDragMouseExited	MenuDragMouseListener
menuDragMouseReleased	MenuDragMouseListener
menuKeyPressed	MenuKeyListener
menuKeyReleased	MenuKeyListener
menuKeyTyped	MenuKeyListener

JScrollPane:

Attribute	Java Type	Comments
UI	javax.swing.plaf.ScrollPaneUI	
columnHeader	javax.swing.JViewport	
columnHeaderView	java.awt.Component	
componentOrientation	java.awt.ComponentOrientation	ComponentOrientation.LEFT_TO_RIGHT ComponentOrientation.RIGHT_TO_LEFT ComponentOrientation.UNKNOWN
horizontalScrollBar	javax.swing.JScrollBar	
horizontalScrollBarPolicy	int	ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS

rowHeader	javax.swing.JViewport	
rowHeaderView	java.awt.Component	
verticalScrollBar	javax.swing.JScrollBar	
verticalScrollBarPolicy	int	ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS
viewport	javax.swing.JViewport	
viewportBorder	javax.swing.border.Border	
viewportView	java.awt.Component	
wheelScrollingEnabled	boolean	

JSeparator:

Attribute	**Java Type**	**Comments**
UI	javax.swing.plaf.SeparatorUI	
orientation	int	

JSlider:

Attribute	**Java Type**	**Comments**
UI	javax.swing.plaf.SliderUI	
extent	int	
inverted	boolean	
labelTable	java.util.Dictionary	
majorTickSpacing	int	
maximum	int	
minimum	int	
minorTickSpacing	int	
model	javax.swing.BoundedRangeModel	
orientation	int	
paintLabels	boolean	
paintTicks	boolean	
paintTrack	boolean	
snapToTicks	boolean	
value	int	
valueIsAdjusting	boolean	

JSpinner:

Attribute	**Java Type**	**Comments**
UI	javax.swing.plaf.SpinnerUI	
editor	javax.swing.JComponent	
model	javax.swing.SpinnerModel	
value	java.lang.Object	

JSplitPane:

Attribute	Java Type	Comments
UI	javax.swing.plaf.SplitPaneUI	
continuousLayout	boolean	
dividerLocation	int	
dividerSize	int	
lastDividerLocation	int	
oneTouchExpandable	boolean	
orientation	int	JSplitPane.VERTICAL_SPLIT (above/below orientation of components) JSplitPane.HORIZONTAL_SPLIT (left/right orientation of components)
resizeWeight	double	

JTabbedPane:

Attribute	Java Type	Comments
UI	javax.swing.plaf.TabbedPaneUI	
model	javax.swing.SingleSelectionModel	
selectedComponent	java.awt.Component	
selectedIndex	int	
tabLayoutPolicy	int	WRAP_TAB_LAYOUT , SCROLL_TAB_LAYOUT
tabPlacement	int	TOP , BOTTOM , LEFT , RIGHT

JTable:

Attribute	Java Type	Comments
UI	javax.swing.plaf.TableUI	
autoCreateColumnsFromModel	boolean	
autoResizeMode	int	AUTO_RESIZE_OFF, AUTO_RESIZE_NEXT_COLUMN, AUTO_RESIZE_SUBSEQUENT_COLUMNS, AUTO_RESIZE_LAST_COLUMN, AUTO_RESIZE_ALL_COLUMNS
cellEditor	javax.swing.table.TableCellEditor	
cellSelectionEnabled	boolean	
columnModel	javax.swing.table.TableColumnModel	
columnSelectionAllowed	boolean	
dragEnabled	boolean	
editingColumn	int	
editingRow	int	
gridColor	java.awt.Color	
intercellSpacing	java.awt.Dimension	
model	javax.swing.table.TableModel	
preferredScrollableViewportSize	java.awt.Dimension	

rowMargin	int	
rowSelectionAllowed	boolean	
selectionBackground	java.awt.Color	
selectionForeground	java.awt.Color	
selectionMode	int	ListSelectionModel.SINGLE_SELECTION ListSelectionModel.SINGLE_INTERVAL_SELECTION ListSelectionModel.MULTIPLE_INTERVAL_SELECTION
selectionModel	javax.swing.ListSelectionModel	
showGrid	boolean	
showHorizontalLines	boolean	
showVerticalLines	boolean	
surrendersFocusOnKeystroke	boolean	
tableHeader	javax.swing.table.JTableHeader	

JTextArea (child of JTextComponent):

Attribute	Java Type	Comments
columns	int	
lineWrap	boolean	
rows	int	
tabSize	int	
wrapStyleWord	boolean	

JTextField (child of JTextComponent):

Attribute	Java Type	Comments
action	javax.swing.Action	
actionCommand	java.lang.String	
columns	int	
horizontalAlignment	int	LEFT , CENTER, RIGHT, LEADING, TRAILING
scrollOffset	int	

JTextPane (child of JTextComponent):

Attribute	Java Type	Comments
contentType	java.lang.String	
editorKit	javax.swing.text.EditorKit	
logicalStyle	javax.swing.text.Style	
page	java.net.URL	
styledDocument	javax.swing.text.StyledDocument	

Attribute – Event	Listener
hyperlinkUpdate	HyperlinkListener

JToggleButton (child of AbstractButton):
None

JToolBar:

Attribute	Java Type	Comments
UI	javax.swing.plaf.ToolBarUI	
floatable	boolean	
orientation	int	HORIZONTAL, VERTICAL
rollover	boolean	

JTree:

Attribute	Java Type	Comments
UI	javax.swing.plaf.TreeUI	
anchorSelectionPath	javax.swing.tree.TreePath	
cellEditor	javax.swing.tree.TreeCellEditor	
cellRenderer	javax.swing.tree.TreeCellRenderer	
dragEnabled	boolean	
editable	boolean	
expandsSelectedPaths	boolean	
invokesStopCellEditing	boolean	
largeModel	boolean	
leadSelectionPath	javax.swing.tree.TreePath	
model	javax.swing.tree.TreeModel	
rootVisible	boolean	
rowHeight	int	
scrollsOnExpand	boolean	
selectionModel	javax.swing.tree.TreeSelectionModel	
selectionPath	javax.swing.tree.TreePath	
selectionPaths	[Ljavax.swing.tree.TreePath;	
selectionRow	int	
selectionRows	[I	
showsRootHandles	boolean	
toggleClickCount	int	
visibleRowCount	int	

Attribute – Event	Listener
treeCollapsed	TreeExpansionListener
treeExpanded	TreeExpansionListener
treeWillColla	TreeWillExpandLi

pse	stener
treeWillExpa	TreeWillExpandLi
nd	stener
valueChang	TreeSelectionList
ed	ener

Appendix B

Attribute Value Syntax

1. **Primitive types** (int, double, short, char) – Express them as you declare them in Java. You can also declare ints using characters by adding ’’ between the character (ie. ‘F’ is equivalent to the integer value of character F).

2. **Wrapper classes for the primitive types** (Integer, Character, Double etc.) – Treat them as you would treat the primitive types. XGG automatically handles the conversion.

3. **String** (java.lang.String) – Express them as you declare them in Java.

4. **Color** (java.awt.Color) – Express them using the RRGGBB notation (ie. “#FF0000” for red). Alternatively, you can use the constants declared in the Color class (ie. “red”, “black”). You can also suffix “.brighter” and “.darker” on the constants to achieve more varieties (ie. “red.brighter”, “blue.darker”).

5. **LayoutManager** (java.awt.LayoutManager) – Express them by inputting the constructor of any LayoutManager object without the new keyword, only those present in Java 1.5 are valid. “FlowLayout()” and “FlowLayout(2)” are both valid.

6. **Border** (java.swing.border.Border) – Express them by inputting the constructor of any Border object without the new keyword, only those present in Java 1.5 are valid. “EmptyBorder(2,2,2,2)” and “TitledBorder(Title)” are both valid.

7. **Font** (java.awt.Font) – Express them by using the syntax defined for the decode(String) function of Font (See the Java Documentation for more details). “Arial BOLD 15” and “Arial” are both valid.

8. **Icon** (javax.swing.Icon) – Express them by inputting the url or the path of the image file.

9. **Image** (java.awt.Image) – Treat them like the Icon type.

10. **Insets** (java.awt.Insets) – Express them by inputting four comma-separated integers (ie. “5, 5, 5, 5”).

11. **Dimension** (java.awt.Dimension) – Express them by inputting two comma-separated integers (ie. “5, 5”).

12. **KeyStroke** (javax.swing.KeyStroke) – Express them by using the syntax defined for the getKeyStroke(String) function of KeyStroke (See the Java Documentation for more details). “alt shift X” and “ctrl F4” are both valid.

13. **Paint** (java.awt.Paint) – Either treat it as Color or express them by using GradientPaint constructors without the new keyword. “#00FF00” and “GradientPaint(0, 0, white, 100, 100, black, true)” are both valid.

14. **Stroke** (java.awt.Stroke) – Express them by inputting the constructor of BasicStroke object without the new keyword, only those present in Java 1.5 are valid. “BasicStroke()” and “BasicStroke(5.0)” are both valid.

15. **int[]** and **double[]** – Express them by inputting any number of comma-separated values. (ie. “52.0, 20.6, 45.3”).

```
**********
FILE: sp/SP.java

package sp;

import sp.compiler.*;

/**
 * The command line main class.
 */
public class SP {
public static void main(String args[]) {
  try {
    new CompilerArgs(args).start();
  } catch (ExitException e) {
    System.exit(e.getExitID());
  }
}
}

**********
FILE: sp/SPUtil.java

package sp;

import java.util.*;
import java.lang.reflect.*;
import java.beans.*;

/**
 * Contains many utility methods used throughout
the XGG compiler.
 */
public final class SPUtil {
public static final String STYLE_DEFAULT =
":default";

private static final String reserved[] = new String[]
{ "abstract", "boolean",
    "break", "byte", "case", "catch", "char", "class",
"continue",
    "createComponents", "curmodel", "default", "do",
"double", "else",
```

```
    "extends", "false", "final", "finally", "float", "for",
"if", "implements",
    "import", "initialize", "instanceof", "int",
"interface", "long", "main",
    "model", "modelGroup", "native", "new", "null",
"package", "private",
    "protected", "public", "return", "short", "sqlerror",
"static", "strictfp",
    "super", "switch", "synchronized", "this", "throw",
"throws", "transient",
    "true", "try", "void", "volatile", "while" };

public static boolean CONSOLE = true;
private static String lbrk = "\n", allman = "\n{\n",
knf = " {\n";

static {
  lbrk = System.getProperty("line.separator", "\n");
  allman = lbrk + '{' + lbrk;
  knf = " {" + lbrk;
}

public static String openBrace(boolean aType) {
  //Allman - true, Kernel Normal Form - false
  return aType ? allman : knf;
}

public static String lineBreak() {
  return lbrk;
}

public static void errPrint(String file, int line, int
column, String str) {
  if (CONSOLE)
    System.err.println(str);
}

public static void errPrint(String str) {
  errPrint("", 0, 0, str);
}

public static void outPrint(String file, int line, int
column, String str) {
  if (CONSOLE)
    System.out.println(str);
}

public static void outPrint(String str) {
  outPrint("", 0, 0, str);
}

public static boolean isLegalVar(String str) {
  str = str.toLowerCase();
  if (str.length() == 0)
    return false;
  for (int i = 0; i < str.length(); i++) {
    char c = str.charAt(i);
    if (!(((c >= 'a') && (c <= 'z')) || (c == '_') || (c ==
'$') || ((i > 0)
      && (c >= '0') && (c <= '9'))))
      return false;
  }

  return true;
}

public static boolean isUnreservedVar(String str) {
  if (str.startsWith("__"))
    return false;

  if (binarySearchString(str, reserved, 0,
reserved.length - 1) != null)
    return false;

  return true;
}

public static String getModifiersStr(int aMod) {
  if (aMod == 0)
    return "";
  else
    return Modifier.toString(aMod) + ' ';
}

public static String getShortName(String str) {
  return str.substring(str.lastIndexOf('.') + 1);
}

public static String getSubClassName(Class<?> cls)
{
  return getSubClassName(cls, ".");
}

public static String getSubClassName(Class<?> cls,
String repl) {
  String str = cls.getName();
  str = str.substring(str.lastIndexOf('.') + 1);

  return str.replaceFirst("\\$", repl);
}

public static String indent(String aSource, int
anIndent) {
  char[] spaces = new char[anIndent];
  Arrays.fill(spaces, ' ');

  StringBuffer result = new StringBuffer();

  String[] lines = aSource.split(lbrk);
  for (int i = 0; i < lines.length; i++) {
    if (i > 0)
      result.append(lineBreak());
    if (lines[i].length() > 0)
      result.append(spaces);
    result.append(lines[i]);
  }

  return result.toString();
}

public static BeanInfo getBeanInfo(Class<?>
cName) {
  BeanInfo biInfo = null;
  try {
    biInfo = Introspector.getBeanInfo(cName,
Introspector.USE_ALL_BEANINFO);
  } catch (IntrospectionException e) {
    System.out.println("Warning: Introspection error
with " + cName
      + " has occured!");
  }
  return biInfo;
}

public static PropertyDescriptor
binarySearchAtts(String str,
  PropertyDescriptor pd[], int start, int fin) {
  if (start >= fin) {
    if ((start == fin) &&
(str.compareTo(pd[start].getName()) == 0))
      return pd[start];
    return null;
  }

  int mid = (fin + start) / 2;
  int cmp = str.compareTo(pd[mid].getName());

  if (cmp > 0)
```

```
    return binarySearchAtts(str, pd, mid + 1, fin);
  else if (cmp < 0)
    return binarySearchAtts(str, pd, start, mid - 1);
  else
    return pd[mid];
}

public static String binarySearchString(String str,
String arr[], int start,
  int fin) {
  if (start >= fin) {
    if ((start == fin) && (str.compareTo(arr[start])
== 0))
      return arr[start];
    return null;
  }

  int mid = (fin + start) / 2;
  int cmp = str.compareTo(arr[mid]);

  if (cmp > 0)
    return binarySearchString(str, arr, mid + 1, fin);
  else if (cmp < 0)
    return binarySearchString(str, arr, start, mid - 1);
  else
    return arr[mid];
}

public static String escapeStr(String str) {
  StringBuffer result = new StringBuffer();

  for (int i = 0; i < str.length(); i++) {
    char c = str.charAt(i);
    if (c == '\\' || c == '"') {
      result.append('\\');
    } else if (c == '\n') {
      result.append("\\n");
      continue;
    } else if (c == '\t') {
      result.append("\\t");
      continue;
    } else if (c == '\r') {
      result.append("\\r");
      continue;
    } else if (c < 32 || c > 127) {
      String valstr = Integer.toString((int) c, 16);
      int vallen = valstr.length();

      result.append("\\u");
      while (vallen < 4) {
        result.append('0');
        vallen++;
      }
      result.append(valstr);
      continue;
    }
    result.append(c);
  }

  return result.toString();
}

public static String escapeChar(char c) {
  if (c == '\'')
    return "\'";
  else if (c == '\\')
    return "\\\\";
  else if (c == '\n')
    return "\n";
  else if (c == '\r')
    return "\r";
  else if (c == '\t')
    return "\t";
  else
    return c + "";
}

public static boolean isClassAssignable(Class<?>
parentcls, Class<?> cls) {
  if (parentcls.getClassLoader() ==
cls.getClassLoader())
    return parentcls.isAssignableFrom(cls);
  else {
    if (parentcls.isAssignableFrom(cls))
      return true;

    Class<?> scls = cls;
    Class<?> spcls = parentcls;
    while ((scls != null) && (spcls != null)) {
      if (((spcls == null)) || (!
spcls.getName().equals(scls.getName())))
        return false;

      scls = scls.getSuperclass();
      spcls = spcls.getSuperclass();
    }
    return true;
  }
}
}

**********
FILE: sp/XGGException.java

package sp;

/**
 * The base exception class used throughout the
XGG Compiler.
 */
public abstract class XGGException extends
Exception {
public static boolean DODEBUG = false;

public static final void debugStr(String str) {
  if (SPUtil.CONSOLE)
    System.out.println('[' + str + ']');
}

public abstract int getLineNumber();

public abstract int getColumnNumber();

public abstract String getError();
}

**********
FILE: sp/ExitException.java

package sp;

/**
 * Thrown to indicate that exit is needed.
 */
public class ExitException extends Exception {
static final long serialVersionUID = 'e' * 'x' * 'i' * 't';
private int exitID = 0;

public ExitException(int anExitID) {
  super("Error id: " + anExitID);
  exitID = anExitID;
}

public int getExitID() {
  return exitID;
}
}
```

```
**********
FILE: sp/code/JBracedCode.java

package sp.code;

/**
 * The java code that has indentation and bracing.
This class supports passing
 * of indentation and bracing style to ther code.
 */
public abstract class JBracedCode {
protected boolean bBrace = false;
protected int indents = 4;

public final String openBrace() {
  return sp.SPUtil.openBrace(bBrace);
}

public final String indent(String aSource) {
  if (indents == 0)
    return aSource;
  return sp.SPUtil.indent(aSource, indents);
}

public final boolean getBraceType() {
  return bBrace;
}

public final void setBraceType(boolean aType) {
  bBrace = aType;
}

public final int getIndents() {
  return indents;
}

public final void setIndents(int i) {
  indents = i;
}

public final void copyCodeParams(JBracedCode
aCode) {
  bBrace = aCode.bBrace;
  indents = aCode.indents;
}
}

**********
FILE: sp/code/JClassType.java

package sp.code;

/**
 * Interface representing a code that contains
variables and methods - a class
*/
public interface JClassType {
  public void addVar(JVar aVar);

  public void addMethod(JMethod aMethod);
}

**********
FILE: sp/code/JArg.java

package sp.code;

/**
 * The code object for a method argument.
 */
public class JArg extends JBracedCode {
protected String sName, sType;

public JArg(String aName, String aType) {
  sName = aName;
  sType = aType;
}

public String getName() {
  return sName;
}

public String getType() {
  return sType;
}

public String toString() {
  return sType + ' ' + sName;
}

public static String toString(JArg[] anArgs) {
  if (anArgs == null)
    return "";
  StringBuffer result = new StringBuffer();
  for (int i = 0; i < anArgs.length; i++) {
    result.append(anArgs[i]);
    if ((i + 1) < anArgs.length)
      result.append(", ");
  }
  return result.toString();
}
}

**********
FILE: sp/code/JVar.java

package sp.code;

import java.util.*;
import java.lang.reflect.*;

/**
 * The code object for a variable/field declaration.
 */
public class JVar extends JArg {
private int iMod;
private String sInitval;
private JQuickClass qInitVal = null;
private boolean isCreated = false;

public JVar(String aName, String aType) {
  this(Modifier.PRIVATE, aName, aType, null);
}

public JVar(String aName, String aType, String
anInitval) {
  this(Modifier.PRIVATE, aName, aType, anInitval);
}

public JVar(int amod, String aName, String aType,
String anInitval) {
  super(aName, aType);
  sInitval = anInitval;
  iMod = amod;
}

public void setModifier(int aMod) {
  iMod = aMod;
}

public void setCreated(boolean b) {
  isCreated = b;
}

public boolean isCreated() {
  return isCreated;
}
```

```
public void setInitVal(String str) {
  sInitval = str;
}

public String getInitVal() {
  return sInitval;
}

public void setQInitVal(JQuickClass jqc) {
  qInitVal = jqc;
}

public JQuickClass getQInitVal() {
  return qInitVal;
}

public String toString() {
  StringBuffer result = new StringBuffer();

  if (!isCreated) {
    result.append(sp.SPUtil.getModifiersStr(iMod));
    result.append(sType);
    result.append(' ');
  }
  result.append(sName);

  if (sInitval != null) {
    result.append(" = ");
    result.append(sInitval);
  } else if (qInitVal != null) {
    result.append(" =");
    if (qInitVal.isHasBody())
      result.append(sp.SPUtil.lineBreak());
    else
      result.append(' ');
    qInitVal.copyCodeParams(this);
    result.append(qInitVal.toString());
  }
  result.append(';');

  return result.toString();
}

public static String toString(LinkedList<JVar> aVars,
JBracedCode jbc) {
  if (aVars == null)
    return "";

  StringBuffer result = new StringBuffer();
  Iterator<?> i = aVars.iterator();
  while (i.hasNext()) {
    JVar var = (JVar) i.next();
    if (jbc != null)
      var.copyCodeParams(jbc);

    result.append(var);
    result.append(sp.SPUtil.lineBreak());
  }
  return result.toString();
}
}

**********
FILE: sp/code/JMethod.java

package sp.code;

import java.util.*;
import java.lang.reflect.*;

/**
 * The code object for a method declaration.
 */
public class JMethod extends JBracedCode {
private int iMod;
private String sReturns;
private String sName;
private JArg[] jaArgs;
private StringBuffer sbBody;
private boolean isConstructor = false;
private LinkedList<JVar> vars = new
LinkedList<JVar>();
private LinkedList<String> tothrows = new
LinkedList<String>();

public JMethod(int aMod, String aReturns, String
aName, JArg[] anArgs) {
  iMod = aMod;
  sReturns = aReturns;
  sName = aName;
  jaArgs = anArgs;
}

public JMethod(int aMod, String aReturns, String
aName) {
  this(aMod, aReturns, aName, null);
}

public JMethod(String aReturns, String aName) {
  this(Modifier.PUBLIC, aReturns, aName, null);
}

public JMethod(Method meth) {
  this(meth.getModifiers(), null, meth.getName(),
null);
  Class<?> cls = meth.getReturnType();
  if (cls != null)
    sReturns =
sp.SPUtil.getShortName(cls.getName());

  Class<?> methargs[] =
meth.getParameterTypes();
  if (methargs.length > 0) {
    JArg args[] = new JArg[methargs.length];
    for (int i = 0; i < args.length; i++)
      args[i] = new JArg("e" + (i > 0 ?
Integer.toString(i) : ""), sp.SPUtil
        .getShortName(methargs[i].getName()));
    jaArgs = args;
  }
}

public void setName(String aName) {
  sName = aName;
}

public String getName() {
  return sName;
}

public StringBuffer getBody() {
  return sbBody;
}

public void setArgs(JArg[] anArgs) {
  jaArgs = anArgs;
}

public JArg[] getArgs() {
  return jaArgs;
}

public void addVar(JVar aVar) {
  vars.add(aVar);
}

public void addThrows(String anException) {
  tothrows.add(anException);
```

```
}

public void setModifier(int aMod) {
  iMod = aMod;
}

public void setBody(StringBuffer aBody) {
  sbBody = aBody;
}

public void setConstructor(boolean aCons) {
  isConstructor = aCons;
}

public boolean getConstructor() {
  return isConstructor;
}

public static JMethod makeMainMethod() {
  JArg mainargs[] = new JArg[1];
  mainargs[0] = new JArg("args", "String[]");

  return new JMethod(Modifier.PUBLIC |
Modifier.STATIC, null, "main", mainargs);
}

public String getClassBody() {
  StringBuffer result = new StringBuffer();

  result.append(JVar.toString(vars, this));

  if (sbBody != null)
    result.append(sbBody);

  return result.toString();
}

public String toString() {
  StringBuffer result = new StringBuffer();

  result.append(sp.SPUtil.getModifiersStr(iMod));

  if (!isConstructor) {
    if (sReturns != null)
      result.append(sReturns);
    else
      result.append("void");
    result.append(' ');
  }

  result.append(sName);

  result.append('(');
  if (jaArgs != null)
    result.append(JArg.toString(jaArgs));
  result.append(')');

  Iterator<?> i = tothrows.iterator();
  if (i.hasNext()) {
    result.append(" throws ");
    while (i.hasNext()) {
      result.append(i.next());
      if (i.hasNext())
        result.append(", ");
    }
  }

  if (!Modifier.isAbstract(iMod)) {
    result.append(openBrace());

    if (sbBody != null)
      result.append(indent(getClassBody()));

    result.append(sp.SPUtil.lineBreak());
    result.append('}');
  } else
    result.append(';');

  return result.toString();
}
}

**********
FILE: sp/code/JQuickClass.java

package sp.code;

import java.util.*;

/**
 * The code object of an anonymous class.
 */
public class JQuickClass extends JBracedCode
implements JClassType {
protected String name = null;
protected String passvals = null;
protected LinkedList<JVar> vars = new
LinkedList<JVar>();
protected LinkedList<JMethod> methods = new
LinkedList<JMethod>();
protected boolean hasBody = true;

protected JQuickClass() {}

public JQuickClass(String aName) {
  name = aName;
}

public String getClassBody() {
  StringBuffer result = new StringBuffer();
  boolean dospace = false;

  if (vars.size() > 0) {
    result.append(JVar.toString(vars, this));
    dospace = true;
  }

  Iterator<?> i = methods.iterator();
  while (i.hasNext()) {
    JMethod meth = (JMethod) i.next();
    meth.copyCodeParams(this);
    if (dospace)
      result.append(sp.SPUtil.lineBreak());
    else
      dospace = true;
    result.append(meth);
    result.append(sp.SPUtil.lineBreak());
  }

  return result.toString();
}

public void addMethod(JMethod aMethod) {
  methods.add(aMethod);
}

public void addVar(JVar aVar) {
  vars.add(aVar);
}

public void setPassValues(String s) {
  passvals = s;
}

public void setHasBody(boolean b) {
  hasBody = b;
}
```

```
public boolean isHasBody() {
  return hasBody;
}

public String toString() {
  return toStringBuffer().toString();
}

public StringBuffer toStringBuffer() {
  StringBuffer result = new StringBuffer();

  result.append("new ");
  result.append(name);
  result.append('(');
  if (passvals != null)
    result.append(passvals);
  result.append(')');

  if (hasBody) {
    result.append(openBrace());

    result.append(indent(getClassBody()));

    result.append(sp.SPUtil.lineBreak());
    result.append('}');
  }

  return result;
}
}

**********
FILE: sp/code/JEvent.java

package sp.code;

import java.util.*;
import java.beans.*;
import java.lang.reflect.*;

/**
 * The code object for a subclassed eventlistener.
 */
public class JEvent extends JQuickClass {
private EventSetDescriptor esd = null;
private HashMap<String, String> methodstable =
new HashMap<String, String>();
private boolean temp = true;
private String spName = null;

public JEvent(EventSetDescriptor anEsd) {
  super(sp.SPUtil.getShortName(anEsd.getListenerT
ype().getName()));
  esd = anEsd;

  Method meths[] = esd.getListenerMethods();
  for (int i = 0; i < meths.length; i++) {
    JMethod jm = new JMethod(meths[i]);
    jm.setModifier(Modifier.PUBLIC);
    addMethod(jm);
  }
}

public Class<?> getTargetClass() {
  return esd.getListenerType();
}

public boolean isTemp() {
  return temp;
}

public void setTemp(boolean aTemp) {
  temp = aTemp;
}

public String getSpecialName() {
  return spName;
}

public void setSpecialName(String str) {
  spName = str;
}

public boolean checkEqual(JEvent anEvent) {
  if (!
esd.getName().equals(anEvent.esd.getName()))
    return false;

  HashMap<String, String> compared =
anEvent.methodstable;

  Iterator<?> keys =
methodstable.keySet().iterator();

  while (keys.hasNext()) {
    String strkey = (String) keys.next();
    String str1 = (String) methodstable.get(strkey);
    String str2 = (String) compared.get(strkey);

    if ((str1 == null) || (!str1.equals(str2)))
      return false;
  }

  return true;
}

public void addListenerMethod(String aName, String
aMethod) {
  methodstable.remove(aName);
  methodstable.put(aName, aMethod);

  Iterator<?> i = methods.iterator();

  while (i.hasNext()) {
    JMethod meth = (JMethod) i.next();

    if (aMethod.equals(meth.getName())) {
      StringBuffer sb = new StringBuffer();
      sb.append(aName);
      sb.append('(');

      JArg args[] = meth.getArgs();

      if (args != null)
        for (int j = 0; j < args.length; j++) {
          sb.append(args[j].getName());
          if ((j + 1) < args.length)
            sb.append(", ");
        }

      sb.append(");");
      meth.setBody(sb);
    }
  }
}

public String toStringAttr(JBracedCode jbc) {
  StringBuffer result = new StringBuffer();

  if (jbc != null)
    copyCodeParams(jbc);

  result.append("add");
  result.append(sp.SPUtil.getShortName(esd.getList
enerType().getName()));

  if (temp) {
    result.append('(');
```

```
    result.append(sp.SPUtil.lineBreak());

    result.append(indent(toString()));

    result.append(sp.SPUtil.lineBreak());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
   } else {
    result.append('(');
    result.append(spName);
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
   }

   return result.toString();
}
}

**********
FILE: sp/code/JClass.java

package sp.code;

import java.util.*;
import java.lang.reflect.*;

/**
 * The code object of a class declaration.
 */
public class JClass extends JBracedCode implements
JClassType {
private int iMod;
private String sName;
private String sParentclass;
private String sPackagename = null;
private LinkedList<String> imports = new
LinkedList<String>();
private StringBuffer interfaces = new StringBuffer();
private LinkedList<JVar> vars = new
LinkedList<JVar>();
private LinkedList<JMethod> methods = new
LinkedList<JMethod>();
private LinkedList<JClass> innerclasses = new
LinkedList<JClass>();

private StringBuffer sbBody = new StringBuffer();

public JClass(int aMod, String aName, String
aParentclass) {
  iMod = aMod;
  sName = aName;
  sParentclass = aParentclass;
}

public JClass(String aName, String aParentclass) {
  this(Modifier.PUBLIC, aName, aParentclass);
}

public JClass(String aName) {
  this(Modifier.PUBLIC, aName, null);
}

public void addInterface(String str) {
  if (interfaces.length() == 0)
    interfaces.append(str);
  else {
    interfaces.append(", ");
    interfaces.append(str);
  }
}

public void addImport(String str) {
  imports.add(str);
}

public void addVar(JVar aVar) {
  vars.add(aVar);
}

public void addMethod(JMethod aMethod) {
  methods.add(aMethod);
}

public void addInnerClass(JClass aClass) {
  innerclasses.add(aClass);
}

public void setPackage(String aPackage) {
  sPackagename = aPackage;
}

public String getPackage() {
  return sPackagename;
}

public String getName() {
  return sName;
}

public void setModifier(int aMod) {
  iMod = aMod;
}

public void setBody(StringBuffer aBody) {
  sbBody = aBody;
}

public String getClassBody() {
  StringBuffer result = new StringBuffer();
  boolean dospace = false;

  if (vars.size() > 0) {
    result.append(JVar.toString(vars, this));
    dospace = true;
  }

  Iterator<?> i = innerclasses.iterator();
  while (i.hasNext()) {
    JClass jcls = (JClass) i.next();
    jcls.copyCodeParams(this);

    if (dospace)
      result.append(sp.SPUtil.lineBreak());
    else
      dospace = true;
    result.append(jcls);
    result.append(sp.SPUtil.lineBreak());
  }

  i = methods.iterator();
  while (i.hasNext()) {
    JMethod meth = (JMethod) i.next();
    meth.copyCodeParams(this);

    if (dospace)
      result.append(sp.SPUtil.lineBreak());
    else
      dospace = true;
    result.append(meth);
    result.append(sp.SPUtil.lineBreak());
  }

  result.append(sbBody);

  return result.toString();
}

public String toString() {
```

```
  StringBuffer result = new StringBuffer();

  if (sPackagename != null) {
    result.append("package ");
    result.append(sPackagename);
    result.append(';');
    result.append(sp.SPUtil.lineBreak());
    result.append(sp.SPUtil.lineBreak());
  }

  if (imports.size() > 0) {
    Iterator<?> i = imports.iterator();
    while (i.hasNext()) {
      result.append("import ");
      result.append(i.next());
      result.append(';');
      result.append(sp.SPUtil.lineBreak());
    }
    result.append(sp.SPUtil.lineBreak());
  }

  result.append(sp.SPUtil.getModifiersStr(iMod));
  result.append("class ");
  result.append(sName);
  if (sParentclass != null) {
    result.append(" extends ");
    result.append(sParentclass);
  }

  if (interfaces.length() > 0) {
    result.append(" implements ");
    result.append(interfaces);
  }

  result.append(sp.SPUtil.openBrace(bBrace));
  result.append(indent(getClassBody()));
  result.append(sp.SPUtil.lineBreak());
  result.append('}');

  return result.toString();
}
}

**********
FILE: sp/code/JShape.java

package sp.code;

import sp.format.*;
import sp.format.code.*;
import org.xml.sax.helpers.*;
import java.util.*;

/**
 * The code object for a group of or a single
statement of method/s from
 * <code>java.awt.geom</code> and
<code>java.awt.Graphics2D</code>.
 */
public class JShape extends XGGTag implements
ViewType, TreeType {
private static final int SHAPE_2D = 1, SHAPE_COLOR
= 2, SHAPE_STROKE = 3,
  SHAPE_TEXT = 4, SHAPE_GPATH = 5, SHAPE_PATH
= 6, SHAPE_IMAGE = 7,
  SHAPE_BLOCK = 11;
private int mode = 0;
private XGGElement elem;
private XGGView view;
private String type = null, varname = null;
private StringBuffer cache = null;

private LinkedList<JShape> children = null;
private JShape parent = null;

public JShape(String aType, XGGElement anElem,
AttributesImpl anAtts) {
  super(anAtts);
  type = aType;
  elem = anElem;
  if (elem != null) {
    xggMain = elem.getMain();
    view = elem.getView();
  }
}

public String getViewType() {
  return type;
}

public String getTagType() {
  return type;
}

public String getErrorHeader() {
  return "Error in UI Tag";
}

public String validate() {
  String errstr = obtainMode();
  if (errstr != null)
    return errstr;

  for (int i = 0; i < atts.getLength(); i++) {
    String qName = atts.getQName(i);
    if (mode == SHAPE_2D) {
      if ((!qName.equals("do")) && (!
qName.equals("contains"))) {
        try {
          targetClass.getField(qName);
        } catch (Exception e) {
          return "Invalid attribute '" + qName + "'.";
        }
      } else if (qName.equals("contains")) {
        String qVal = atts.getValue(i);
        if (!qVal.equals("true") && !
qVal.equals("false"))
          return printContains(qName, "true and
false");
      } else {
        String qVal = atts.getValue(i);
        if (!qVal.equals("draw") && !qVal.equals("fill")
          && !qVal.equals("both"))
          return printContains(qName, "draw, fill and
both");
      }
    } else if (mode == SHAPE_COLOR) {
      if (qName.equals("type")) {
        String qVal = atts.getValue(i);
        if (!qVal.equals("draw") && !qVal.equals("fill"))
          return printContains(qName, "draw and fill");
      } else if ((!qName.equals("value")) && (!
qName.equals("valueexp")))
        return "Invalid attribute '" + qName + "'.";
    } else if (mode == SHAPE_STROKE) {
      if (!qName.equals("value"))
        return "Invalid attribute '" + qName + "'.";
    } else if (mode == SHAPE_TEXT) {
      if (qName.equals("orientation")) {
        String qVal = atts.getValue(i);
        if (!qVal.equals("horizontal") && !
qVal.equals("vertical"))
          return printContains(qName, "horizontal and
vertical");
      } else if ((!qName.equals("value")) && (!
qName.equals("x"))
        && (!qName.equals("y")) && (!
qName.equals("valueexp")))
```

```
        return "Invalid attribute '" + qName + "'.";
    } else if (mode == SHAPE_IMAGE) {

      if ((!qName.equals("x")) && (!
qName.equals("y"))
        && (!qName.equals("url")) && (!
qName.equals("width"))
        && (!qName.equals("height")))
        return "Invalid attribute '" + qName + "'.";
    } else if (mode == SHAPE_GPATH) {
      if (qName.equals("contains")) {
        String qVal = atts.getValue(i);
        if (!qVal.equals("true") && !
qVal.equals("false"))
          return printContains(qName, "true and
false");
      } else if (qName.equals("do")) {
        String qVal = atts.getValue(i);
        if (!qVal.equals("draw") && !qVal.equals("fill")
          && !qVal.equals("both"))
          return printContains(qName, "draw, fill and
both");
      } else
        return "Invalid attribute '" + qName + "'.";
    } else if (mode == SHAPE_PATH) {
      if ((!qName.equals("x")) && (!
qName.equals("y")))
        return "Invalid attribute '" + qName + "'.";
    } else if (mode == SHAPE_BLOCK) {
      if (!qName.equals("condition"))
        return "Invalid attribute '" + qName + "'.";
    } else
      return "The tag '" + type + "' is unsupported for
now.";
  }

  if (mode == SHAPE_IMAGE) {
    if (!isAttributeExist("url"))
      return "Attribute 'url' is required!";
  }

  return null;
}

private String printContains(String qName, String
vals) {
  return "The '" + qName + "' attribute can only
have " + vals
    + " as possible values.";
}

private String obtainMode() {
  mode = 0;
  try {
    targetClass = Class.forName("java.awt.geom." +
type + "2D$Double");
    mode = SHAPE_2D;
    return null;
  } catch (ClassNotFoundException e) {}

  if (type.equals("Paint"))
    mode = SHAPE_COLOR;
  else if (type.equals("Stroke"))
    mode = SHAPE_STROKE;
  else if (type.equals("Text"))
    mode = SHAPE_TEXT;
  else if (type.equals("GeneralPath"))
    mode = SHAPE_GPATH;
  else if (type.equals("Path"))
    mode = SHAPE_PATH;
  else if (type.equals("Image"))
    mode = SHAPE_IMAGE;
  else if ((type.equals("Component")) ||
(type.equals("Border")))
    mode = SHAPE_BLOCK;

  if (mode == 0)
    return "Unknown view tag type";
  return null;
}

public void addChild(JShape js) {
  if (children == null)
    children = new LinkedList<JShape>();
  children.add(js);
  js.parent = this;
}

public StringBuffer getPaintCode() throws
XGGTagException {
  StringBuffer result = new StringBuffer();

  if (elem.getMain().getOptions().isComments()) {
    result.append("// UI tag '");
    result.append(type);
    result.append("' with start tag ending at (");
    result.append(getLineNumber());
    result.append(", ");
    result.append(getColumnNumber());
    result.append(')');
    result.append(sp.SPUtil.lineBreak());
  }

  try {
    StringBuffer temp = getCacheCode();
    if (temp != null)
      result.append(temp);
  } catch (XGGTagException e) {
    throw e;
  }

  if ((mode == SHAPE_2D) || (mode ==
SHAPE_GPATH)) {
    String dostr = getAttributeValue("do");
    if ((dostr != null) && (!dostr.equals("draw"))) {
      result.append("__g2d.setPaint(__fillpnt);");
      result.append(sp.SPUtil.lineBreak());
      result.append("__g2d.fill(");
      result.append(varname);
      result.append(");");
      result.append(sp.SPUtil.lineBreak());
    }
    if ((dostr == null) || (!dostr.equals("fill"))) {
      result.append("__g2d.setPaint(__drawpnt);");
      result.append(sp.SPUtil.lineBreak());
      result.append("__g2d.draw(");
      result.append(varname);
      result.append(");");
      result.append(sp.SPUtil.lineBreak());
    }
  } else if (mode == SHAPE_COLOR) {
    String typestr = getAttributeValue("type");
    if ((typestr == null) || (typestr.equals("draw")))
      result.append("__drawpnt = ");
    else
      result.append("__fillpnt = ");

    boolean toext = true;
    String valuestr = getAttributeValue("valueexp");
    if (valuestr == null) {
      valuestr = getAttributeValue("value");
      toext = false;
    } else if (isAttributeExist("value")) {
      xggMain.giveWarning("Attribute 'valuestr'
overrides 'value'. Attibute"
        + " 'value' will be ignored", elem, this);
    }
    if (valuestr == null) {
```

```
      valuestr = "";
      toext = false;
    }

    try {
      if (!toext)
        result.append(ValueConverter.convertValue(va
luestr,
          java.awt.Paint.class, java.awt.Color.class,
elem));
      else
        result.append(valuestr);

      result.append(';');
      result.append(sp.SPUtil.lineBreak());
    } catch (sp.XGGException e) {
      throw new XGGTagException(this,
XGGConversionException.getNeededError(
        "value", valuestr,
java.awt.Paint.class.getName(), e));
    }
  } else if (mode == SHAPE_STROKE) {
    String valuestr = getAttributeValue("value");
    if (valuestr == null)
      valuestr = "";
    result.append("__g2d.setStroke(");
    try {
      String paintstr =
ValueConverter.convertValue(valuestr,
        java.awt.Stroke.class,
java.awt.BasicStroke.class, elem);
      result.append(paintstr);
      result.append(");");
      result.append(sp.SPUtil.lineBreak());
    } catch (sp.XGGException e) {
      throw new XGGTagException(this,
XGGConversionException.getNeededError(
        "value", valuestr,
java.awt.Stroke.class.getName(), e));
    }
  } else if (mode == SHAPE_TEXT) {
    boolean toext = true;
    String valuestr = getAttributeValue("valueexp");
    if (valuestr == null) {
      valuestr = getAttributeValue("value");
      toext = false;
    } else if (isAttributeExist("value")) {
      xggMain.giveWarning("Attribute 'valuestr'
overrides 'value'. Attibute"
        + " 'value' will be ignored", elem, this);
    }
    if (valuestr == null) {
      valuestr = "";
      toext = false;
    }

    result.append("__g2d.setPaint(__drawpnt);");
    result.append(sp.SPUtil.lineBreak());

    String orientstr =
getAttributeValue("orientation");
    if ((orientstr == null) ||
(orientstr.equals("horizontal"))) {
      result.append("__g2d.drawString(");

      try {
        if (!toext)
          result.append(ValueConverter.convertValue(v
aluestr, String.class,
            java.awt.Graphics.class, elem));
        else
          result.append(valuestr);
        result.append(", ");

        valuestr = getAttributeValue("x");
        if (valuestr == null)
          valuestr = "0";
        result.append(ValueConverter.checkExpression
(valuestr, "x"));

        result.append(", ");
        valuestr = getAttributeValue("y");
        if (valuestr == null)
          valuestr = "fontmetrics.getHeight()";
        result.append(ValueConverter.checkExpression
(valuestr, "y"));

        result.append(");");
        result.append(sp.SPUtil.lineBreak());
      } catch (sp.XGGException e) {
        throw new XGGTagException(this,
XGGConversionException.getNeededError(
          "value", valuestr, String.class.getName(), e));
      }
    } else {
      view.initStrVert();

      result.append("drawStringVertical(g,");

      try {
        if (!toext)
          result.append(ValueConverter.convertValue(v
aluestr, String.class,
            java.awt.Graphics.class, elem));
        else
          result.append(valuestr);

        result.append(", ");

        valuestr = getAttributeValue("x");
        if (valuestr == null)
          valuestr = "0";
        result.append(ValueConverter.checkExpression
(valuestr, "x"));

        result.append(", ");
        valuestr = getAttributeValue("y");
        if (valuestr == null)
          valuestr = "fontmetrics.getHeight()";
        result.append(ValueConverter.checkExpression
(valuestr, "y"));

        result.append(");");
        result.append(sp.SPUtil.lineBreak());
      } catch (sp.XGGException e) {
        throw new XGGTagException(this,
XGGConversionException.getNeededError(
          "value", valuestr, String.class.getName(), e));
      }
    }
  } else if (mode == SHAPE_IMAGE) {
    result.append("g.drawImage(");

    String valuestr = getAttributeValue("url");
    try {
      result.append(ValueConverter.convertValue(val
uestr, java.awt.Image.class,
        java.awt.Graphics.class, elem));
      result.append(", ");

      valuestr = getAttributeValue("x");
      if (valuestr == null)
        valuestr = "0";
      result.append(ValueConverter.checkExpression(
valuestr, "x"));

      result.append(", ");
      valuestr = getAttributeValue("y");
```

```
      if (valuestr == null)
        valuestr = "0";
      result.append(ValueConverter.checkExpression(
valuestr, "y"));

      String attrstr = "width";
      valuestr = getAttributeValue("width");
      if (valuestr == null) {
        valuestr = getAttributeValue("height");
        attrstr = "height";
      }

      if (valuestr != null) {
        result.append(", ");
        result.append(ValueConverter.checkExpression
(valuestr, attrstr));

        result.append(", ");
        attrstr = "height";
        String valuestr2 = getAttributeValue("height");
        if (valuestr2 == null) {
          valuestr2 = valuestr;
          attrstr = "width";
        }

        result.append(ValueConverter.checkExpression
(valuestr2, attrstr));
      }

      result.append(", this);");
      result.append(sp.SPUtil.lineBreak());
    } catch (sp.XGGException e) {
      throw new XGGTagException(this,
XGGConversionException.getNeededError(
        "url", valuestr,
java.awt.Image.class.getName(), e));
    }
  } else if (mode == SHAPE_PATH) {
    result.append(varname);

    try {
      String valuestr = getAttributeValue("x");
      if (valuestr == null)
        valuestr = "0";
      result.append(ValueConverter.checkExpression(
valuestr, "x"));

      result.append(", ");
      valuestr = getAttributeValue("y");
      if (valuestr == null)
        valuestr = "0";
      result.append(ValueConverter.checkExpression(
valuestr, "y"));

      result.append(");");
      result.append(sp.SPUtil.lineBreak());
    } catch (sp.XGGException e) {
      throw new XGGTagException(this, e);
    }

  } else if (mode == SHAPE_BLOCK) {
    if (children != null) {
      String condstr = getAttributeValue("condition");
      int indents = 0;

      if (condstr != null) {
        result.append("if ( ");
        try {
          result.append(ValueConverter.checkExpressio
n(condstr, "condition"));
        } catch (sp.XGGException e) {
          throw new XGGTagException(this, e);
        }
        result.append(" )");
        result.append(elem.getMain().getOptions().ope
nBrace());
        indents =
elem.getMain().getOptions().getIndents();
      }

      Iterator<?> i = children.iterator();
      while (i.hasNext()) {

        JShape child = (JShape) i.next();

        StringBuffer childcode = child.getPaintCode();
        if (indents == 0)
          result.append(childcode);
        else
          result.append(sp.SPUtil.indent(childcode.toStr
ing(), indents));
      }

      if (condstr != null) {
        result.append(sp.SPUtil.lineBreak());
        result.append('}');
        result.append(sp.SPUtil.lineBreak());
      }
    }
  }

  return result;
}

public StringBuffer getContainsCode() throws
XGGTagException {
  StringBuffer result = new StringBuffer();

  try {
    StringBuffer temp = getCacheCode();
    if (temp != null)
      result.append(temp);
  } catch (XGGTagException e) {
    throw e;
  }

  if ((mode == SHAPE_2D) || (mode ==
SHAPE_GPATH)) {
    result.append("if (");
    result.append(varname);
    result.append(".contains(x,y)) return true;");
    result.append(sp.SPUtil.lineBreak());
  } else
    return null;

  return result;
}

private StringBuffer getCacheCode() throws
XGGTagException {
  StringBuffer result = new StringBuffer();

  if (cache != null)
    return cache;
  else {
    if (mode == SHAPE_2D) {
      if (varname == null)
        varname = view.getNewName(targetClass);

      String classname =
sp.SPUtil.getSubClassName(targetClass);
      JVar jv = new JVar(0, varname, classname, "new
" + classname + "()");

      result.append(jv.toString());
      result.append(sp.SPUtil.lineBreak());

      for (int i = 0; i < atts.getLength(); i++) {
```

```
        String qName = atts.getQName(i);
        if ((!qName.startsWith("_")) && (!
qName.equals("do"))
          && (!qName.equals("contains"))) {
          result.append(varname);
          result.append('.');
          result.append(qName);
          result.append(" = ");
          try {
            result.append(ValueConverter.checkExpressi
on(atts.getValue(i),
              qName));
          } catch (sp.XGGException e) {
            throw new XGGTagException(this, e);
          }
          result.append(';');
          result.append(sp.SPUtil.lineBreak());
        }
      }
      cache = new StringBuffer(result);
    } else if (mode == SHAPE_GPATH) {
      if (varname == null)
        varname =
view.getNewName(java.awt.geom.GeneralPath.clas
s);

      JVar jv = new JVar(0, varname, "GeneralPath",
"new GeneralPath()");

      result.append(jv.toString());
      result.append(sp.SPUtil.lineBreak());

      if (children != null) {
        boolean haschild = false;
        Iterator<?> i = children.iterator();
        while (i.hasNext()) {

          JShape child = (JShape) i.next();

          if (!haschild)
            child.varname = varname + ".moveTo(";
          else
            child.varname = varname + ".lineTo(";
          result.append(child.getPaintCode());
          haschild = true;
        }
        if (haschild) {
          result.append(varname);
          result.append(".closePath();");
          result.append(sp.SPUtil.lineBreak());
        }
      }

      cache = new StringBuffer(result);
    }
  }

  return result;
}

public String toString() {
  return type;
}

public Object[] getTreeChildren() {
  LinkedList<Object> ll = new
LinkedList<Object>();

  for (int i = 0; i < atts.getLength(); i++)
    if (atts.getQName(i).charAt(0) != '_')
      ll.add(atts.getQName(i) + ": " +
atts.getValue(i));

  if (children != null) {
    Iterator<JShape> iter = children.iterator();
    while (iter.hasNext())
      ll.add(iter.next());
  }

  return ll.toArray(new Object[] { ll.size() });
}

public Object getTreeParent() {
  if (parent != null)
    return parent;
  return view;
}

public String getIconName() {
  return "jtypeassist_co.gif";
}

class ElementAttribute implements TreeType {
public String name;
public String value;
public JShape shape;

public ElementAttribute(String aName, String
aValue, JShape aShape) {
  name = aName;
  value = aValue;
  shape = aShape;
}

public String toString() {
  return name + ": " + value;
}

public Object[] getTreeChildren() {
  return new Object[0];
}

public Object getTreeParent() {
  return shape;
}

public String getIconName() {
  return "public_co.gif";
}
}
}

**********
FILE: sp/compiler/CompilerArgs.java

package sp.compiler;

import sp.*;
import java.io.*;
import java.util.*;

/**
 * The command line argument interpreter.
 */
public class CompilerArgs {
LinkedList<String> args = new
LinkedList<String>();
LinkedList<String> runs = new
LinkedList<String>();
CompilerOpts cOpts = new CompilerOpts();

static final String optString[][] = {
    { "-b", "Use Kernel Normal Form bracing style in
generated code (default)" },
    { "-B", "Use Allman bracing bracing style in
generated code" },
    { "-i [2/4/8]", "Set number of characters for
indent (default: 4)" },
```

```
    { "-p", "Only parse the input file, no output file" },
    { "-1.5", "Generated code will include Java 1.5 or
higher only codes" },
    { "-nh", "Remove additional comments in
generated code" },
    { "-si", "Do structural indention in generated
code" },
    { "-pt", "Generate a prototype class (activates
-ne)" },
    { "-ne", "Do not create/update Event file (already
activated with -m)" },
    { "-nc", "Do not compile generated UI/prototype
file" },
    { "-dc",
        "Compile generated UI/prototype file and
delete the generated code" },
    { "-nce", "Do not compile event file" },
    { "-r", "Run event file (Only works with only a
single input)" },
    { "-v", "Output messages about what the
compiler is doing" },
    { "-J [code]", "pass arguments to javac" },
    { "-JR [code]", "pass arguments to java
interpreter" },
    { "-JA [code]", "pass arguments to event class
file" },
    { "-C [file]", "use config file" }, { "-help", "display
this message" } };

public CompilerArgs(String anArgs[]) {
  for (int i = 0; i < anArgs.length; i++)
    args.add(anArgs[i]);

  if ((args.size() == 0) &&
(sp.XGGException.DODEBUG)) {
    sp.XGGException.debugStr("added argument
simple1.xml");
    args.add("tests\\simple1.xml");
  }
}

public void start() throws ExitException {

  if (args.size() == 0)
    printHelp();
  else
    parse();

  if (runs.size() > 1)
    cOpts.setRunEvent(false);

  Iterator<?> i = runs.iterator();
  while (i.hasNext()) {
    new ConsoleCompiler((String) i.next(), cOpts);
  }
}

private void parse() throws ExitException {
  int i = 0;
  while (i < args.size()) {
    String curstr = (String) args.get(i);

    if (sp.XGGException.DODEBUG)
      sp.XGGException.debugStr("Argument read: " +
curstr);

    if (curstr.equals("-b"))
      cOpts.setBraceType(false);
    else if (curstr.equals("-B"))
      cOpts.setBraceType(true);
    else if (curstr.equals("-1.5"))
      cOpts.setMode1_5(true);
    else if (curstr.equals("-p"))
      cOpts.setCheckOnly(true);
    else if (curstr.equals("-nh"))
      cOpts.setComments(false);
    else if (curstr.equals("-si"))
      cOpts.setStructIndent(true);
    else if (curstr.equals("-v"))
      cOpts.setVerbose(true);
    else if (curstr.equals("-pt"))
      cOpts.setPrototype(true);
    else if (curstr.equals("-ne"))
      cOpts.setDoEvent(false);
    else if (curstr.equals("-nc")) {
      cOpts.setCompile(false);
      cOpts.setDeleteCode(false);
    } else if (curstr.equals("-dc")) {
      cOpts.setCompile(true);
      cOpts.setDeleteCode(true);
    } else if (curstr.equals("-nce"))
      cOpts.setCompileEvent(false);
    else if (curstr.equals("-r"))
      cOpts.setRunEvent(true);
    else if (curstr.equals("-i")) {
      try {
        i++;

        int indents = Integer.parseInt((String)
args.get(i));
        if ((indents != 2) && (indents != 4) && (indents
!= 8))
          throw new NumberFormatException();

        cOpts.setIndents(indents);
      } catch (IndexOutOfBoundsException e) {
        error("No argument after '" + curstr + "'. Type
'-help' as parameter "
          + "for details");
      } catch (NumberFormatException e) {
        error("Argument after '" + curstr + "' is not 2,
4 or 8.");
      }
    } else if ((curstr.equals("-J")) || (curstr.equals("-
JR"))
      || curstr.equals("-JA")) {
      try {
        i++;
        if (curstr.equals("-J"))
          cOpts.setCompilerArgs((String) args.get(i));
        else if (curstr.equals("-JA"))
          cOpts.setRunArgs((String) args.get(i));
        else
          cOpts.setJavaArgs((String) args.get(i));
      } catch (IndexOutOfBoundsException e) {
        error("No argument after '" + curstr + "'. Type
'-help' as parameter "
          + "for details");
      }
    } else if (curstr.equals("-C")) {
      try {
        i++;
        String argstr = (String) args.get(i);

        File fPath = new File(argstr);

        try {
          BufferedReader bufrd = new
BufferedReader(new FileReader(fPath));
          int tempi = i + 1;

          String curline = bufrd.readLine();
          while (curline != null) {
            if (curline.length() > 0)
              args.add(tempi++, curline);
            curline = bufrd.readLine();
          }
```

```
        bufrd.close();
       } catch (FileNotFoundException fe) {
        error("File '" + fPath.getName() + "' not
found!");
       } catch (IOException fe) {
        error("Error in reading '" + fPath.getName() +
"'");
       }
     } catch (IndexOutOfBoundsException e) {
      error("No argument after '" + curstr + "'. Type
'-help' as parameter "
        + "for details");
     }
   } else if (curstr.equals("-help"))
    printHelp();
   else if (curstr.equals("--debug"))
    sp.XGGException.DODEBUG = true;
   else {
    File fPath = new File(curstr);

    if ((!fPath.exists()) || (!fPath.isFile())) {
     String str = fPath.getName();

     if ((str.length() > 0) && (str.charAt(0) == '-'))
      error("File '" + curstr
        + "' not found. It may be an invalid
parameter."
        + " Type '-help' as parameter for details");

     error("File '" + curstr + "' not found.");
    }

    String fname = fPath.getName();

    int dotpos = fname.lastIndexOf('.');

    if (dotpos > 0) {
     String extension = fname.substring(dotpos,
fname.length());
     if ((!extension.equals(".xml")) && (!
extension.equals(".xgg")))
      error("File '" + curstr + "' is not an xgg file.");
    } else
     error("File '" + curstr + "' is not an xgg file.");

    runs.add(curstr);
   }
   i++;
  }
}

private static final char[] fillspaces(int i) {
  char[] carr = new char[20 - optString[i]
[0].length()];

  Arrays.fill(carr, ' ');

  return carr;
}

public static void printHelp() {
  System.out.println("XGG Tool; Roy Gian S.P
Balderrama");
  System.out.println("Usage: [program name]
[options] [xml files]\n");
  System.out.println("Options include:");

  StringBuffer sb = new StringBuffer();
  for (int i = 0; i < optString.length; i++) {
   sb.append("  ");
   sb.append(optString[i][0]);
   sb.append(fillspaces(i));
   sb.append(optString[i][1]);
   sb.append("\n");
  }
  System.out.println(sb);
}

private static void error(String str) throws
ExitException {
  System.err.println(str);
  throw new ExitException(1);
}
}
```

```
**********
FILE: sp/compiler/CompilerOpts.java

package sp.compiler;

/**
 * The class that contains the compiler options.
 */
public class CompilerOpts {
private boolean brace = false, verbose = false,
prototype = false,
  doEvent = true, compile = true, deleteCode =
false, comments = true,
  structIndent = false, checkOnly = false,
compileEvent = true,
  runEvent = false, mode1_5 = false;
private int indents = 4;
private String sCompArgs = "", sCompName =
"javac.exe", sRunArgs = "",
  sJavaArgs = "";

public void setBraceType(boolean b) {
  brace = b;
  debugStr("Brace mode set to " + (b ? "Allman" :
"Kernel Normal Form")
    + " bracing");
}

public boolean getBraceType() {
  return brace;
}

public void setVerbose(boolean b) {
  verbose = b;
  debugStr("Verbose mode set to " + b);
}

public boolean isVerbose() {
  return verbose;
}

public void setPrototype(boolean b) {
  prototype = b;
  debugStr("Protoype mode set to " + b);
}

public boolean isPrototype() {
  return prototype;
}

public void setDoEvent(boolean b) {
  doEvent = b;
  debugStr("Do Event mode set to " + b);
}

public boolean isDoEvent() {
  return doEvent;
}

public void setCompile(boolean b) {
  compile = b;
  debugStr("Compile mode set to " + b);
}
```

```
public boolean isCompile() {
  return compile;
}

public void setDeleteCode(boolean b) {
  deleteCode = b;
  debugStr("Delete Code mode set to " + b);
}

public boolean isDeleteCode() {
  return deleteCode;
}

public void setComments(boolean b) {
  comments = b;
  debugStr("Comments mode set to " + b);
}

public boolean isComments() {
  return comments;
}

public void setStructIndent(boolean b) {
  structIndent = b;
  debugStr("Struct Indent mode set to " + b);
}

public boolean isStructIndent() {
  return structIndent;
}

public void setCompileEvent(boolean b) {
  compileEvent = b;
  debugStr("Compile Event mode set to " + b);
}

public boolean isCompileEvent() {
  return compileEvent;
}

public void setRunEvent(boolean b) {
  runEvent = b;
  debugStr("Run Event mode set to " + b);
}

public boolean isRunEvent() {
  return runEvent;
}

public void setCheckOnly(boolean b) {
  checkOnly = b;
  debugStr("Parse Only mode set to " + b);
}

public boolean isCheckOnly() {
  return checkOnly;
}

public void setMode1_5(boolean b) {
  mode1_5 = b;
  debugStr("1.5 Source mode set to " + b);
}

public boolean isMode1_5() {
  return mode1_5;
}

public void setIndents(int i) {
  indents = i;
  debugStr("Indent number set to " + i);
}

public int getIndents() {
  return indents;
}

public void setCompilerArgs(String str) {
  sCompArgs = str;
  debugStr("Arguments for Compiler set to " + str);
}

public String getCompilerArgs() {
  return sCompArgs;
}

public void setRunArgs(String str) {
  sRunArgs = str;
  debugStr("Arguments for Event set to " + str);
}

public String getRunArgs() {
  return sRunArgs;
}

public void setCompilerName(String str) {
  sCompName = str;
  debugStr("Compiler set to " + str);
}

public String getCompilerName() {
  return sCompName;
}

public void setJavaArgs(String str) {
  sJavaArgs = str;
  debugStr("Arguments for Java set to " + str);
}

public String getJavaArgs() {
  return sJavaArgs;
}

public String getGeneric(String str, String gen) {
  if (!mode1_5)
    return str;
  else
    return str + '<' + gen + '>';
}

public String openBrace() {
  return sp.SPUtil.openBrace(brace);
}

public String indent(String aSource) {
  if (checkOnly)
    return aSource;
  return sp.SPUtil.indent(aSource, indents);
}

private void debugStr(String str) {
  if (sp.XGGException.DODEBUG)
    sp.XGGException.debugStr(str);
}
}

**********
FILE: sp/compiler/ConsoleCompiler.java

package sp.compiler;

import sp.*;
import sp.format.*;
import org.xml.sax.*;
import org.antlr.runtime.*;
import java.io.*;
import javax.xml.parsers.*;
```

```
/**
 * The command line compiler for a single file.<br>
 * It does the following:<br>
 * <ul>
 * <li>Parses the xml/xgg file
 * <li>Compiles the generated internal format from
parsing to .java code
 * <li>Compiles the .java code to bytecoe
 * <li>Updates the corresponding "event" file and
compiles it to .java code
 * <li>Runs the event file if possible
 * </ul>
 */
public class ConsoleCompiler {
private String sXMLFile = null, sOutFile = null,
sEvtFile = null;
private InputStream isXML = null;
private SAXParserFactory factorySAX = null;
private SAXParser sax = null;
private PrintWriter pwFile = null;

private CompilerOpts cOpts = null;
private XGGFormat xggMain = null;

public ConsoleCompiler(String fname, CompilerOpts
aCOpts) throws ExitException {
  sXMLFile = fname;
  cOpts = aCOpts;
  xggMain = new XGGFormat(aCOpts, new
File(fname));

  if (sp.XGGException.DODEBUG) {
    sp.XGGException.debugStr("Verbose mode auto
set to true");
    cOpts.setVerbose(true);
  }

  try {
    isXML = new FileInputStream(sXMLFile);
    factorySAX = SAXParserFactory.newInstance();
    sax = factorySAX.newSAXParser();

    long l = System.currentTimeMillis();

    XGGParser xparser = new XGGParser(xggMain,
sXMLFile, cOpts.isVerbose());
    sax.setProperty("http://xml.org/sax/properties/lex
ical-handler", xparser);
    sax.parse(isXML, xparser);

    l -= System.currentTimeMillis();
    if (cOpts.isVerbose())
      System.out.println("Time spent in parsing: " + (-
l) + "ms");

    isXML.close();
    isXML = null;
  } catch (FileNotFoundException e) {
    errPrint(e, "File " + sXMLFile + " not found!",
false, 1);
  } catch (FactoryConfigurationError e) {
    errPrint(e, "Error creating SAX parser factory:",
true, 2);
  } catch (ParserConfigurationException e) {
    errPrint(e, "Error creating SAX parser:", true, 2);
  } catch (IOException e) {
    errPrint(e, "I/O error while reading from file:",
false, 2);
  } catch (SAXException e) {
    errPrint(e, "Parsing error from " + sXMLFile, false,
1);
  } finally {
    if (isXML != null) {
      try {
        isXML.close();
      } catch (IOException e) {}
      isXML = null;
    }
  }

  sOutFile = sXMLFile.substring(0,
sXMLFile.indexOf('.')) + "GUI";

  try {
    xggMain.setName(new File(sOutFile).getName());
    xggMain.setRootName(sXMLFile);

    if (cOpts.isVerbose())
      System.out.println("Compiling " + sXMLFile);

    long l = System.currentTimeMillis();

    String filestr = xggMain.writeFile();

    l -= System.currentTimeMillis();
    if (cOpts.isVerbose()) {
      System.out.println("Time spent in compiling: " +
(-l) + "ms");
      System.out.println("Creating " + sOutFile +
".java");
    }

    if (cOpts.isCheckOnly()) {
      System.out.println("\nOperation complete");
      return;
    }

    File fOutFile = new File(sOutFile + ".java");

    pwFile = new PrintWriter(fOutFile);
    pwFile.println("//Created by XGG Tool");
    pwFile.println("//Roy Gian S.P Balderrama");
    pwFile.println();
    pwFile.print(filestr);
    pwFile.close();

    if (cOpts.isDeleteCode())
      fOutFile.deleteOnExit();

    if (cOpts.isVerbose())
      System.out.println("Finished creating " +
sOutFile + ".java");
  } catch (FileNotFoundException err) {
    errPrint(err, "Error in creating/accessing " +
sOutFile + ".java", false, 2);
  } catch (XGGCompileException err) {
    errPrint(err, "Error in compiling " + sOutFile +
".java", false, 1);
  } catch (Exception err) {
    errPrint(err, "Unknown error while compiling " +
sOutFile + ".java", true,
      2);
  }

  try {
    if (cOpts.isCompile()) {
      String compName = cOpts.getCompilerName();

      System.out.println("Invoking compiler " +
cOpts.getCompilerName()
        + " for " + sOutFile + ".java");

      Process proc = Runtime.getRuntime().exec(
        compName + " " + cOpts.getCompilerArgs() +
" " + sOutFile + ".java");

      StreamGobbler errGobbler = new
StreamGobbler(proc.getErrorStream());
```

```
      StreamGobbler outGobbler = new
StreamGobbler(proc.getInputStream());

      errGobbler.start();
      outGobbler.start();

      if (proc.waitFor() != 0)
        throw new XGGRunException("Exit value is
nonzero");
      errGobbler.validate();
      outGobbler.validate();

      if (cOpts.isVerbose())
        System.out.println("End of invoking compiler "
          + cOpts.getCompilerName());

    }
   } catch (XGGRunException err) {
    errPrint(err, "Error encountered while running
compiler for " + sOutFile
      + ".java", false, 2);
   } catch (Exception err) {
    errPrint(err, "Error encountered while running
compiler for " + sOutFile
      + ".java", true, 2);
   }

   sEvtFile = sXMLFile.substring(0,
sXMLFile.indexOf('.'));

   BufferedReader EvtRead = null;
   try {
    if ((cOpts.isDoEvent()) && (!cOpts.isPrototype()))
{

      if (cOpts.isVerbose())
        System.out.println("Generating Updated Event
file " + sEvtFile
          + ".java");

      File fEvtFile = new File(sEvtFile + ".java");

      try {
        EvtRead = new BufferedReader(new
FileReader(fEvtFile));
      } catch (FileNotFoundException e) {
        EvtRead = null;
      }

      long l = System.currentTimeMillis();
      EventFileMaker evm = new EventFileMaker(new
File(sEvtFile).getName(),
        sEvtFile, xggMain, cOpts, EvtRead);

      String filestr = evm.writeFile();
      if (EvtRead != null) {
        EvtRead.close();
        EvtRead = null;
      }

      l -= System.currentTimeMillis();
      if (cOpts.isVerbose()) {
        System.out.println("Time spent in generating
updated event file: "
          + (-l) + "ms");
        System.out.println("Writing " + sOutFile +
".java");
      }

      File fbakFile = new File(sEvtFile + ".java.bak");

      if (fEvtFile.exists()) {
        InputStream copyin = new
FileInputStream(fEvtFile);
        OutputStream copyout = new
FileOutputStream(fbakFile);

        byte[] buf = new byte[1024];
        int len;

        while ((len = copyin.read(buf)) > 0)
          copyout.write(buf, 0, len);
        copyin.close();
        copyout.close();
      }

      if (evm.isDirty()) {
        pwFile = new PrintWriter(fEvtFile);
        pwFile.print(filestr);
        pwFile.close();
      }

      if ((cOpts.isCompileEvent()) &&
(cOpts.isCompile())) {
        String compName =
cOpts.getCompilerName();

        System.out.println("Invoking compiler " +
cOpts.getCompilerName()
          + " for " + sEvtFile + ".java");

        Process proc = Runtime.getRuntime().exec(
          compName + " " + cOpts.getCompilerArgs()
+ " " + sEvtFile + ".java");

        StreamGobbler errGobbler = new
StreamGobbler(proc.getErrorStream());
        StreamGobbler outGobbler = new
StreamGobbler(proc.getInputStream());

        errGobbler.start();
        outGobbler.start();

        if (proc.waitFor() != 0)
          throw new XGGRunException("Exit value is
nonzero");
        errGobbler.validate();
        outGobbler.validate();

        if (cOpts.isVerbose())
          System.out.println("End of invoking compiler
"
            + cOpts.getCompilerName());
      }
    }

    System.out.println("\nOperation complete");

    if ((!cOpts.isRunEvent()) || (!xggMain.isDoMain()))
      ;
    else if (((cOpts.isDoEvent()) && (!
cOpts.isPrototype()) && (cOpts
      .isCompileEvent()))
      || (cOpts.isPrototype())) {

      String sRunClass = xggMain.getPackage();
      if (sRunClass == null)
        sRunClass = "";
      else
        sRunClass = sRunClass + ".";

      if (!cOpts.isPrototype())
        sRunClass = sRunClass + new
File(sEvtFile).getName();
      else
        sRunClass = sRunClass + new
File(sOutFile).getName();
```

```
      System.out.println("Running " + sRunClass);

      Process proc = Runtime.getRuntime().exec(
        "java " + cOpts.getJavaArgs() + " " +
sRunClass + " "
          + cOpts.getRunArgs());

      StreamGobbler errGobbler = new
StreamGobbler(proc.getErrorStream());
      StreamGobbler outGobbler = new
StreamGobbler(proc.getInputStream());

      errGobbler.start();
      outGobbler.start();

      proc.waitFor();

    }
   } catch (FileNotFoundException err) {
    System.err.println("Error in creating/accessing "
+ sEvtFile + ".java");
   } catch (RecognitionException err) {
    System.err.println("Error in parsing the existing "
+ sEvtFile + ".java");
   } catch (XGGRunException err) {
    errPrint(err, "Error encountered while running
compiler for " + sEvtFile
      + ".java", false, 2);
   } catch (Exception err) {
    System.err.println("Unknown error while creating
event file " + sEvtFile
      + ".java");
    err.printStackTrace();
   } finally {
    try {
     if (EvtRead != null)
      EvtRead.close();
    } catch (IOException e) {
     System.out.println("\nUnknown error while
closing " + sEvtFile + ".java "
       + "file holders");
     e.printStackTrace();
    }
   }
}

public static void errPrint(Throwable e, String str,
boolean istrace,
  int anexitcode) throws ExitException {
  System.err.println('\n' + str);
  if (istrace)
    e.printStackTrace();
  throw new ExitException(anexitcode);
}
}

**********
FILE: sp/compiler/BaseParser.java

package sp.compiler;

import sp.format.*;
import java.util.*;
import java.beans.*;
import java.lang.reflect.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.xml.sax.ext.*;

/**
 * The base XML parser class, provides error and
warning handling methods.
 */

public class BaseParser extends DefaultHandler
implements LexicalHandler {
protected static final int MODE_ELEMENT = 0,
MODE_SPECIAL = 1, MODE_ITEM = 2,
  MODE_UI = 3;

protected boolean readingCDATA = false;
protected boolean firstelem = true;
protected boolean verbose = false;
protected boolean nonspec = true;
protected boolean canloop = false;
protected boolean islooped = false;
protected String sfname = null;
protected int parseMode = MODE_ELEMENT;

protected Locator locErr = null;
protected XGGFormat xggMain;

protected Stack<Object> stackElem = new
Stack<Object>();

public BaseParser(XGGFormat anXGG, String str,
boolean isVerbose) {
  xggMain = anXGG;
  sfname = str;
  verbose = isVerbose;
}

public void setDocumentLocator(Locator locator) {
  locErr = locator;
}

private void printError(String str,
SAXParseException e) {
  String errStr = sfname + ":" + e.getLineNumber()
+ "," + e.getColumnNumber()
    + ": " + e.getMessage();

  sp.SPUtil.errPrint(sfname, e.getLineNumber(),
e.getColumnNumber(), str + ": "
    + errStr);

  LinkedList<CompilerMessage> ll =
xggMain.getMessages();
  ll.add(new CompilerMessage(sfname,
e.getLineNumber(), e.getColumnNumber(),
    errStr));
}

public void error(SAXParseException e) throws
SAXException {
  warning(e);
}

public void warning(SAXParseException e) throws
SAXException {
  printError("Warning", e);
}

public void fatalError(SAXParseException e) throws
SAXException {
  printError("Error", e);
  throw new SAXException(e);
}

protected void passError(String str) throws
SAXException {
  fatalError(new SAXParseException(str, locErr));
}

protected void passWarning(String str) throws
SAXException {
  warning(new SAXParseException(str, locErr));
}
```

```
public void startDocument() {
  if (verbose)
    sp.SPUtil.outPrint("Parsing " + sfname);
}

public void endDocument() {
  if (verbose)
    sp.SPUtil.outPrint("Completed parsing for " +
sfname);
}

protected Object peekStack(Stack<?> stk) {
  try {
    return stk.peek();
  } catch (EmptyStackException e) {
    return null;
  }
}

protected void checkName(String elemId, String
idname) throws SAXException {
  if (elemId != null) {
    if (!sp.SPUtil.isLegalVar(elemId))
      passError("Invalid " + idname + " '" + elemId +
"'.");
    if (!sp.SPUtil.isUnreservedVar(elemId))
      passError("Reserved " + idname + " '" + elemId
+ "'. Use another one.");
  } else
    passError("Attribute '" + idname + "' have no
value.");
}

protected void checkNameAttr(String elemId, String
qName) throws SAXException {
  checkName(elemId, "id");
  if (!xggMain.addIdNames(elemId, qName)) {
    String usedstr = xggMain.checkIdName(elemId);
    passError("Duplicate id '" + elemId + "'."
      + ((usedstr != null) ? ("Already used in a/an " +
usedstr + ".") : ""));
  }
}

protected void checkNameEvent(String elemId,
String qAttr, Method aMethod)
  throws SAXException {
  checkName(elemId, "event name(" + qAttr + ")");
  if (!xggMain.addIdNamesEvent(elemId, aMethod))
{
    String usedstr = xggMain.checkIdName(elemId);
    if ((usedstr == null) || (usedstr.equals("event")))
      passError("event '" + elemId + "' used on 2 or "
        + "more different types of events.");
    else
      passError("event name'" + elemId + "' already
used "
        + "on a variable of type " + usedstr + ".");
  }
}

protected void attrChecks(AttributesImpl attsi,
Class<?> cls, String qName,
  boolean isStyle) throws SAXException {

  for (int i = 0; i < attsi.getLength(); i++) {
    if ((isStyle) && (attsi.getValue(i).startsWith("{"))
      && (!attsi.getValue(i).startsWith("{late")))
      passError("Special attributes (starting with '{'
other than late "
        + "binding) is not allowed in a style tag.");
    Class<?> spcls =
XGGSupported.isSpecialAttr(attsi.getQName(i), cls);
    if (attsi.getQName(i).equals("model.loop")) {
      if (isStyle)
        passError("Attribute '" + attsi.getQName(i)
          + "' is not allowed in styles file.");
      if (!canloop)
        passError("Attribute '" + attsi.getQName(i) + "'
is only allowed in "
          + "children tags of XGG files with list and
database models.");
      if (islooped)
        passError("Attribute '" + attsi.getQName(i)
          + "' can only be defined once.");
      if (!attsi.getValue(i).equals("true"))
        passError("Attribute '" + attsi.getQName(i)
          + "' is only allowed to have true as it's
value.");
      islooped = true;
      attsi.setType(i, spcls.getName());
    } else if (spcls != null)
      attsi.setType(i, spcls.getName());
    else if ((!attsi.getQName(i).equals("id"))
      && (!attsi.getQName(i).equals("styleid"))) {
      PropertyDescriptor attrpd =
xggMain.checkAttr(cls, attsi.getQName(i));
      if (attrpd != null) {
        if (attrpd.getWriteMethod() != null) {
          attsi.setType(i,
attrpd.getPropertyType().getName());
          continue;
        }
        passError("Attribute '" + attsi.getQName(i)
          + "' is a protected property of element '" +
qName + "'.");
      }

      Method methType =
xggMain.checkEventAttr(cls, attsi.getQName(i));
      if (methType == null)
        passError("Attribute '" + attsi.getQName(i)
          + "' is not a valid XGG Attribute for element '"
+ qName + "'.");
      else {
        checkNameEvent(attsi.getValue(i),
attsi.getQName(i), methType);
        attsi.setType(i, methType.getName());
      }
    }
  }
}

public void endCDATA() throws SAXException {
  readingCDATA = false;
}

public void startCDATA() throws SAXException {
  readingCDATA = true;
}

public void comment(char[] ch, int start, int length)
throws SAXException {}

public void endEntity(String name) throws
SAXException {}

public void startEntity(String name) throws
SAXException {}

public void endDTD() throws SAXException {}

public void startDTD(String name, String publicId,
String systemId)
  throws SAXException {}
```

```
}

**********
FILE: sp/compiler/XGGParser.java

package sp.compiler;

import sp.code.*;
import sp.format.*;
import java.io.*;
import java.util.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;

/**
 * The parser of the XGG/XML files. Converts them
to internal format.
 */
public class XGGParser extends BaseParser {
private Stack<XGGItem> stackItem = new
Stack<XGGItem>();
private Stack<ViewType> stackView = new
Stack<ViewType>();

public XGGParser(XGGFormat anXGG, String str,
boolean isVerbose) {
  super(anXGG, str, isVerbose);
}

private void isSpecialTagValid(String qName)
throws SAXException {
  if (peekTopElem() != xggMain.getHeadElement())
    passError(qName + " must be a child of the head
element!");

  if (!nonspec)
    passError("Special Tag '" + qName + "' cannot be
used after you have "
      + "used an XGG Element tag (JButton etc..). ");
}

private String checkPathTags(String qName,
Attributes attsi, int warnlen)
  throws SAXException {
  String pathstr = attsi.getValue("path");
  if (pathstr == null)
    passError(qName + " must contain the 'path'
attribute.");

  if (attsi.getLength() > warnlen)
    passWarning(qName + " have only 1 attribute:
'path'."
      + " Other attributes are ignored.");

  return pathstr;
}

public void startElement(String namespaceURI,
String localName, String qName,
  Attributes atts) throws SAXException {
  Class<?> cls = null;
  String elemId = null;
  boolean wasloop = islooped;
  AttributesImpl attsi = new AttributesImpl(atts);

  if (firstelem) {
    if (qName.equals("JFrame"))
      xggMain.setType(XGGFormat.MTYPE_JFRAME);
    else if (qName.equals("JApplet"))
      xggMain.setType(XGGFormat.MTYPE_JAPPLET);
    else if (xggMain.checkElement(qName) != null)
      xggMain.setType(XGGFormat.MTYPE_JOBJECT);
    else if (qName.equals("Styles"))
      passError("Styles file is not directly compiled!");
    else
      passError("Head element '" + qName
        + "' is invalid. Use only JFrame, JApplet, JObject
or Styles.");
  }

  if (qName.equals("Import")) {
    parseMode = MODE_SPECIAL;
    isSpecialTagValid(qName);

    String pathstr = checkPathTags(qName, attsi, 1);
    if (!xggMain.addImport(pathstr))
      passError("Package '" + pathstr + "' not
found!");
  } else if (qName.equals("Package")) {
    parseMode = MODE_SPECIAL;
    isSpecialTagValid(qName);

    String pathstr = xggMain.getPackage();
    if (pathstr != null)
      passError("Package name is already defined as'"
+ pathstr + "'. ");

    xggMain.setPackage(checkPathTags(qName,
attsi, 1));
  } else if (qName.equals("Model")) {
    parseMode = MODE_SPECIAL;
    isSpecialTagValid(qName);

    if (xggMain.getOptions().isPrototype())
      passWarning("Prototype mode ignores
Models.");

    if (xggMain.getModel() != null)
      passError("Only one model is allowed.");

    String modeltype = attsi.getValue("type");
    if ((modeltype == null) ||
(modeltype.equals("class"))
      || (modeltype.equals("list"))) {
      String pathstr = checkPathTags(qName, attsi,
(modeltype == null) ? 1 : 2);

      Class<?> clsmdl =
xggMain.checkElement(pathstr);

      if (clsmdl == null)
        passError("Class '" + pathstr + "' not found!
Try to add necessary "
          + "import tags if needed.");

      canloop = !((modeltype == null) ||
(modeltype.equals("class")));
      XGGClassModel xggcmdl = new
XGGClassModel(xggMain, clsmdl, canloop);

      xggMain.setModel(xggcmdl);
    } else if (modeltype.equals("database")) {
      canloop = true;

      if (attsi.getLength() > 1)
        passWarning(qName + " does not need any
attribute."
          + " Other attributes are ignored.");

      XGGDataModel xggdmdl = new
XGGDataModel(xggMain);
      xggMain.setModel(xggdmdl);
    } else
      passError("Model has wrong data type (only
class,list and "
        + "database allowed).");
  } else if (qName.equals("Action")) {
```

```
    parseMode = MODE_SPECIAL;
    isSpecialTagValid(qName);

    String errchk = XGGAction.validate(attsi);
    if (errchk != null)
      passError("In 'Action' Tag: " + errchk);

    XGGAction action = new XGGAction(xggMain,
attsi);
    checkNameAttr(action.getAttributeValue("id"),
"AbstractAction");

    checkNameEvent(action.getAttributeValue("actio
nPerformed"),
      "actionPerformed",
xggMain.checkEventAttr(javax.swing.JButton.class,
        "actionPerformed"));

    action.installPosition(locErr);
    xggMain.addAction(action);
  } else if (qName.equals("Style")) {
    parseMode = MODE_SPECIAL;
    isSpecialTagValid(qName);

    String pathstr = checkPathTags(qName, attsi, 1);
    parseStyle(pathstr);
    if (xggMain.getHeadElement() != null)
      xggMain.getApplyStyle(xggMain.getHeadEleme
nt());
  } else if (XGGSupported.isItemType(qName)) {
    if ((parseMode != MODE_ITEM) && (parseMode !
= MODE_ELEMENT))
      passError("Tag '" + qName + "' is an invalid
child element.");

    parseMode = MODE_ITEM;

    XGGElement pelem = peekTopElem();
    XGGItem parentitem = (XGGItem)
peekStack(stackItem);

    if ((parentitem == null) && (!
XGGSupported.legalItem(qName, pelem)))
      passError("Tag '"
        + qName
        + "' is an invalid child element"
        + ((pelem != null) ? " of element '" +
pelem.getClassName() + "'."
          : "."));

    if (!XGGSupported.legalChildItem(qName,
parentitem))
      passError("Tag '"
        + qName
        + "' is an invalid child element"
        + ((parentitem != null) ? " of item '" +
parentitem.getTagType() + "'."
          : "."));

    XGGItem curitem = new XGGItem(qName, pelem,
attsi);

    String errAttr = curitem.checkAttributes();
    if (errAttr != null)
      passError("Attribute '" + errAttr + "' is not
allowed in Tag '" + qName
        + "'.");

    curitem.installPosition(locErr);

    stackItem.push(curitem);
    if (parentitem != null)
      parentitem.addItem(curitem);
    else
      pelem.addItem(curitem);
  } else if (qName.equals("UI")) {
    if (parseMode != MODE_ELEMENT)
      passError("Tag 'UI' is an invalid child element.");
    parseMode = MODE_UI;

    XGGElement pelem = peekTopElem();

    if (!XGGSupported.legalElemView(pelem))
      passError("Tag 'UI' is not supported by this
element");

    if (attsi.getLength() > 0)
      passWarning(qName + " does not need any
attribute."
        + " Other attributes are ignored.");

    XGGView view = new XGGView(peekTopElem());

    stackView.push(view);
  } else if (parseMode == MODE_UI) {
    if (!XGGSupported.isViewType(qName))
      passError("Tag '" + qName + "' is not a UI
element.");

    ViewType parentview = (ViewType)
peekStack(stackView);
    if (!XGGSupported.legalViewChild(qName,
parentview))
      passError("Tag '"
        + qName
        + "' is an invalid child element"
        + ((parentview != null) ? " of tag '" +
parentview.getViewType() + "'."
          : "."));

    XGGElement pelem = peekTopElem();
    JShape js = new JShape(qName, pelem, attsi);

    JShape pjs = null;
    if (parentview instanceof XGGView)
      pjs = null;
    else if (!(parentview instanceof JShape))
      passError("Tag '" + qName + "' is an invalid UI
child element.");
    else
      pjs = (JShape) parentview;

    String errchk = pelem.getView().addViewElem(js,
pjs);
    if (errchk != null)
      passError(errchk);
    js.installPosition(locErr);
    stackView.push(js);
  } else {
    if (!firstelem)
      nonspec = false;
    parseMode = MODE_ELEMENT;

    cls = xggMain.checkElement(qName);
    if (cls == null)
      passError("Element tag '" + qName + "' is not a
valid Java Object.");

    if (!XGGSupported.isElementSupported(cls))
      passError("Element tag '" + qName + "' is an
unsupported XGG Element.");

    if (!XGGSupported.legalChildren(cls,
peekTopElem())) {
      XGGElement pelem = peekTopElem();
      passError("Element tag '"
        + qName
        + "' is an invalid child element"
```

```
       + ((pelem != null) ? " of element '" +
pelem.getClassName() + "'."
        : "."));
    }

    int idindex = atts.getIndex("id");
    if (idindex >= 0) {
      if (firstelem)
        passError("Head element cannot have an id.
Use 'this' instead.");
      elemId = attsi.getValue(idindex);
      checkNameAttr(elemId, qName);
      attsi.setType(idindex, String.class.getName());
    }
    idindex = atts.getIndex("styleid");
    if (idindex >= 0) {
      String styleid = attsi.getValue(idindex);
      checkName(styleid, "styleid");
      attsi.setType(idindex, String.class.getName());
    }

    attrChecks(attsi, cls, qName, false);

    XGGElement elem = new XGGElement(xggMain,
elemId, cls, attsi, sp.SPUtil
      .getBeanInfo(cls));

    elem.installPosition(locErr);

    xggMain.addElement(elem, peekTopElem());
    stackElem.push(elem);

    if ((wasloop != islooped) && (islooped))
      elem.setInLoop(true);

    firstelem = false;
  }
}

public void endElement(String namespaceURI,
String localName, String qName)
  throws SAXException {
  if (parseMode == MODE_ELEMENT) {
    if ((!endElemParse(qName)) &&
(sp.XGGException.DODEBUG))
      passWarning("[D] Wrong end parse.");
  } else if (parseMode == MODE_ITEM) {
    XGGItem enditem = (XGGItem)
peekStack(stackItem);
    if ((enditem == null) || (!
qName.equals(enditem.getTagType())))
      passError("Error in tag '" + qName + "' No
matching end tag.");

    stackItem.pop();
    if (peekStack(stackItem) == null)
      parseMode = MODE_ELEMENT;
  } else if (parseMode == MODE_UI) {
    ViewType endview = (ViewType)
peekStack(stackView);
    if (!qName.equals(endview.getViewType()))
      passError("Error in tag '" + qName + "' No
matching end tag.");

    stackView.pop();
    if (peekStack(stackView) == null)
      parseMode = MODE_ELEMENT;
  }
}

public void characters(char[] ch, int start, int length)
throws SAXException {
  String str;
  if (!readingCDATA)
    str = new String(ch, start, length).trim();
  else
    str = new String(ch, start, length);

  if (str.length() == 0)
    return;

  if (parseMode == MODE_ELEMENT) {
    XGGElement elem = peekTopElem();

    if (xggMain.checkAttr(elem.getTargetClass(),
"text") != null) {
      String extrastr = "\n";
      if ((readingCDATA) || (start <= 0)
        || ((ch[start - 1] != 13) && (ch[start - 1] !=
10)))
        extrastr = "";

      String warnstr =
elem.updateContentAttribute("text", str, extrastr);

      if (warnstr != null)
        passWarning(warnstr);
    } else
      passError("Element tag '" +
elem.getClassName() + "' cannot have text.");
  } else if (parseMode == MODE_ITEM) {
    XGGItem item = (XGGItem)
peekStack(stackItem);

    if (item == null)
      passWarning("Error in parsing.");
    else {
      String extrastr = "\n";
      if ((readingCDATA) || (start <= 0)
        || ((ch[start - 1] != 13) && (ch[start - 1] !=
10)))
        extrastr = "";

      String warnstr =
item.updateContentAttribute("value", str, extrastr);

      if (warnstr != null)
        passWarning(warnstr);
    }
  } else if (parseMode == MODE_UI) {
    ViewType view = (ViewType)
peekStack(stackView);

    if (view == null)
      passWarning("Error in parsing.");
    else {
      if (!view.getViewType().equals("Text"))
        passError("This tag cannot have text.");

      JShape js = (JShape) view;

      String extrastr = "\n";
      if ((readingCDATA) || (start <= 0)
        || ((ch[start - 1] != 13) && (ch[start - 1] !=
10)))
        extrastr = "";

      String warnstr =
js.updateContentAttribute("value", str, extrastr);

      if (warnstr != null)
        passWarning(warnstr);
    }
  } else
    passError("This tag cannot have text.");
}

protected XGGElement peekTopElem() {
```

```
  return (XGGElement) peekStack(stackElem);
}

private boolean endElemParse(String str) {
  XGGElement elem = peekTopElem();
  if ((elem != null)
    && ((str.equals(elem.getClassName())) ||
(xggMain.checkElement(str) == elem
      .getTargetClass()))) {
    stackElem.pop();
    return true;
  }
  return false;
}

protected InputStream getStyleFile(String fname)
throws FileNotFoundException,
  IOException {
  File stylefile = new File(fname);
  if ((!stylefile.isAbsolute()) && (fname.charAt(0) !=
File.separatorChar))
    fname =
xggMain.getFile().getAbsoluteFile().getParent() +
File.separator
      + fname;

  return new FileInputStream(fname);
}

private void parseStyle(String fname) throws
SAXException {
  InputStream isXML = null;

  try {
    isXML = getStyleFile(fname);

    SAXParserFactory factorySAX =
SAXParserFactory.newInstance();

    SAXParser sax = factorySAX.newSAXParser();

    sax.parse(isXML, new StyleParser(xggMain,
fname, verbose));

    isXML.close();
    isXML = null;

  } catch (FileNotFoundException e) {
    passError("File " + fname + " not found!");
  } catch (FactoryConfigurationError e) {
    passError("Error creating SAX parser factory for
Style file!");
  } catch (ParserConfigurationException e) {
    passError("Error creating SAX parser for Style
file!");
  } catch (IOException e) {
    passError("I/O error while reading from Style
file!");
  } finally {
    if (isXML != null) {
      try {
        isXML.close();
      } catch (IOException e) {}
      isXML = null;
    }
  }
}
}

**********
FILE: sp/compiler/StyleParser.java

package sp.compiler;

import sp.format.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

/**
 * The parser of the Styles files of the XGG/XML files.
Adds them to the
 * internal format.
 */
public class StyleParser extends BaseParser {
public StyleParser(XGGFormat anXGG, String str,
boolean isVerbose) {
  super(anXGG, str, isVerbose);
}

public void startElement(String namespaceURI,
String localName, String qName,
  Attributes atts) throws SAXException {
  AttributesImpl attsi = new AttributesImpl(atts);
  Class<?> cls = null;

  if (firstelem)
    if (!qName.equals("Styles"))
      passError("Styles file must have 'Styles' as the
head tag");

  if (stackElem.size() > 1) {
    XGGStyle pstyle = (XGGStyle)
peekStack(stackElem);
    passError("Styles element tags cannot have
children! Parent of this "
      + "tag is " + pstyle.getClassName() + "(line " +
pstyle.getLineNumber()
      + ')');
  }

  if (!firstelem) {
    cls = xggMain.checkElement(qName);
    if (cls == null)
      passError("Style tag target '" + qName + "' is
not a valid Java Object.");

    if (!XGGSupported.isElementSupported(cls))
      passError("Style tag target '" + qName
        + "' is an unsupported XGG Element.");
  } else
    cls = XGGStyle.class;

  XGGStyle xstyle = new XGGStyle(cls, xggMain,
attsi);

  if (!firstelem) {
    String stylevar = xstyle.getStyleId();

    if (!stylevar.equals(sp.SPUtil.STYLE_DEFAULT)) {
      if (!sp.SPUtil.isLegalVar(stylevar))
        passError("Invalid Style name '" + stylevar +
"'.");
      if (!sp.SPUtil.isUnreservedVar(stylevar))
        passError("Reserved Style name '" + stylevar
+ "'. Use another one.");
    }

    if (!xggMain.addStyle(xstyle)) {
      if (stylevar.equals(sp.SPUtil.STYLE_DEFAULT))
        passError("Style tag target '" + qName + "'
with default style id "
          + "has a duplicate.");
      else
        passError("Style tag target '" + qName + "'
with style id '" + stylevar
          + "' has a duplicate.");
    }
```

```
    attrChecks(attsi, cls, qName, true);
  } else {
    if (xstyle.countAttributes() > 0)
      passWarning("'Styles' tag target have no
attributes. Ignored.");
  }

  xstyle.installPosition(locErr);

  stackElem.push(xstyle);

  firstelem = false;
}

public void characters(char[] ch, int start, int length)
throws SAXException {
  String str = new String(ch, start, length).trim();
  if (str.length() == 0)
    return;

  if (stackElem.size() == 1)
    passError("'Styles' tag cannot have text");

  XGGStyle xstyle = (XGGStyle)
peekStack(stackElem);

  if (xggMain.checkAttr(xstyle.getTargetClass(),
"text") != null) {
    if (!xstyle.isAttributeExist("text"))
      xstyle.addAttribute("text", str,
String.class.getName());
    else
      passWarning("Attribute 'text' is used twice."
        + " Only the first one is used.");
  } else
    passError("Style tag target '" +
xstyle.getClassName()
      + "' cannot have text.");
}

public void endElement(String namespaceURI,
String localName, String qName)
  throws SAXException {
  if ((!endElemParse(qName)) &&
(sp.XGGException.DODEBUG))
    passWarning("[D] Wrong end parse on style
file.");
}

private boolean endElemParse(String str) {
  if (str.equals("Styles"))
    str = "XGGStyle";
  XGGStyle xstyle = (XGGStyle)
peekStack(stackElem);
  if ((xstyle != null) &&
(str.equals(xstyle.getClassName()))) {
    stackElem.pop();
    return true;
  }
  return false;
}
}

*********************************
FILE: sp/compiler/EventFileMaker.java

package sp.compiler;

import org.antlr.runtime.*;
import sp.format.*;
import sp.code.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;

/**
 * Creates/Updates the event file of a corresponding
XML/XGG file. The event
 * file normally contains the "main" method and the
methods that is called
 * by the events.
 */
public class EventFileMaker {
private XGGFormat xggMain = null;
private CompilerOpts cOpts = null;
private BufferedReader srcFile = null;
private String sName = null;
private String sPathName = null;
private boolean bDirty = false;
private String errMsg = null;

public EventFileMaker(String aName, String
aPathName, XGGFormat aMain,
  CompilerOpts aCOpts, BufferedReader aFile) {
  sName = aName;
  sPathName = aPathName;
  xggMain = aMain;
  cOpts = aCOpts;
  srcFile = aFile;
}

public boolean isDirty() {
  return bDirty;
}

public String getErrorMessage() {
  return errMsg;
}

public String writeFile() throws IOException,
RecognitionException {
  if (srcFile != null)
    return updateOld();
  else
    return makeNew();
}

private String makeNew() {
  JClass jc = new JClass(sName,
xggMain.getName());

  jc.setBraceType(cOpts.getBraceType());
  jc.setIndents(cOpts.getIndents());

  String sPackage = xggMain.getPackage();
  if (sPackage != null)
    jc.setPackage(sPackage);

  xggMain.writeImports(jc);

  XGGModel model = xggMain.getModel();
  if (model != null) {
    JMethod jcons = new JMethod(null, sName);
    jcons.setConstructor(true);
    jcons.setArgs(new JArg[] { new JArg("model",
model.getVarType(cOpts
      .isMode1_5())) });
    jcons.setBody(new
StringBuffer("super(model);"));
    jc.addMethod(jcons);
  } else {
    JMethod jcons = new JMethod(null, sName);
    jcons.setConstructor(true);
    jcons.setBody(new StringBuffer("super();"));
    jc.addMethod(jcons);
  }

  if (xggMain.isDoMain()) {
```

```
    JMethod jmain = JMethod.makeMainMethod();
    jc.addMethod(jmain);

    StringBuffer sbCons = new StringBuffer();
    sbCons.append("//MAIN PROGRAM CODE HERE");
    sbCons.append(sp.SPUtil.lineBreak());
    sbCons.append("new ");
    sbCons.append(sName);
    sbCons.append("(");
    if (model != null)
      sbCons.append("null");
    sbCons.append(");");

    jmain.setBody(sbCons);
  }

  JMethod jinit = new JMethod(null, "initialize");

  jinit.setBody(new StringBuffer("//PUT
INITIALIZATION HERE"));

  jc.addMethod(jinit);

  xggMain.writeEvents(jc, false);

  bDirty = true;
  return jc.toString();
}

private void makeConstruct(StringBuffer result,
String lbrk) {
  JMethod jcons = new JMethod(null, sName);
  jcons.setConstructor(true);

  XGGModel cmodel = xggMain.getModel();
  jcons.setArgs(new JArg[] { new JArg("model",
cmodel.getVarType(cOpts
    .isMode1_5())) });
  jcons.setBody(new StringBuffer("super(model);"));

  result.append(sp.SPUtil.indent(jcons.toString(),
cOpts.getIndents()));
  result.append(lbrk);
}

private String updateOld() throws IOException,
RecognitionException {
  JavaLexer lexer = new JavaLexer(new
ANTLRFileStream(sPathName + ".java"));
  boolean addConstruct = false;
  boolean setPkg = false;

  CommonTokenStream cstokens = new
CommonTokenStream();
  cstokens.setTokenSource(lexer);

  JavaParser parser = new JavaParser(cstokens);
  parser.MainName = sName;
  parser.compilationUnit();

  String sPackage = xggMain.getPackage();
  if ((sPackage != null) && (!
sPackage.equals(parser.packageName)))
    setPkg = true;
  if ((sPackage == null) && (parser.packageName !
= null))
    setPkg = true;

  if (parser.packageLine != parser.packageLineEnd)
    setPkg = false;

  if (parser.ErrMsg != null) {
    errMsg = parser.ErrMsg;
    sp.SPUtil.errPrint(sPathName + ".java", 0, 0,
parser.ErrMsg);
    throw new RecognitionException();
  }

  String event[] = xggMain.getEventList();
  LinkedList<JMethod> addmeth = new
LinkedList<JMethod>();

  for (int i = 0; i < event.length; i++) {
    Iterator<?> e = parser.LLmethods.iterator();
    boolean found = false;
    while (e.hasNext()) {
      if (((String) e.next()).startsWith(event[i])) {
        found = true;
        break;
      }
    }
    if (!found) {
      String strsplit[] = event[i].split(" ");
      JMethod jm =
xggMain.methodFromEvent(strsplit[1],
Modifier.PUBLIC);
      jm.setBraceType(cOpts.getBraceType());
      jm.setIndents(cOpts.getIndents());
      addmeth.add(jm);
    }
  }

  XGGModel model = xggMain.getModel();
  if ((model != null) && (xggMain.getType() !=
XGGFormat.MTYPE_JAPPLET)) {
    addConstruct = true;
    Iterator<?> e = parser.LLconstructs.iterator();
    while (e.hasNext()) {
      String toParse = (String) e.next();
      if (toParse.trim().length() > 2) {
        addConstruct = false;
        break;
      }
    }
  }

  StringBuffer result = new StringBuffer();

  int line = 0;
  String str = null;
  String lbrk = sp.SPUtil.lineBreak();

  if ((setPkg) && (parser.packageName == null)) {
    result.append("package ");
    result.append(sPackage);
    result.append(';');
    result.append(lbrk);
    result.append(lbrk);
    bDirty = true;
  }

  while ((str = srcFile.readLine()) != null) {
    line++;

    if ((setPkg) && (line == parser.packageLine)) {
      if (sPackage != null) {
        StringBuffer npkgstr = new StringBuffer();
        npkgstr.append(str.substring(0,
parser.packageColumn));
        npkgstr.append("package ");
        npkgstr.append(sPackage);
        npkgstr.append(';');
        if (parser.packageColumnEnd + 1 <
str.length())
          npkgstr.append(str.substring(parser.package
ColumnEnd + 1, str
            .length()));
```

```
        str = npkgstr.toString();
      } else
        str = "";
      bDirty = true;
    }

    if ((line != parser.insertLine)
      && ((line != parser.insertCLine) || (!
addConstruct))) {
      result.append(str);
      result.append(lbrk);
    } else if (line == parser.insertCLine) {
      result.append(str.substring(0,
parser.insertCColumn));
      result.append(lbrk);

      makeConstruct(result, lbrk);
      result.append(lbrk);

      for (int j = 0; j < parser.insertCColumn; j++)
        result.append(' ');
      result.append(str.substring(parser.insertCColum
n));
      result.append(lbrk);

      bDirty = true;
    } else {
      result.append(str.substring(0,
parser.insertColumn));

      if ((parser.insertCLine < 0) && (addConstruct))
        makeConstruct(result, lbrk);

      Iterator<?> e = addmeth.iterator();
      while (e.hasNext()) {
        result.append(lbrk);
        result.append(sp.SPUtil.indent(((JMethod)
e.next()).toString(), cOpts
            .getIndents()));
        result.append(lbrk);
        bDirty = true;
      }

      result.append(str.substring(parser.insertColumn
));
      result.append(lbrk);
    }
  }

  return result.toString();
}
}

**********
FILE: sp/compiler/StreamGobbler.java

package sp.compiler;

import java.io.*;

/**
 * The class used to get the out and err streams of a
runtime process. It
 * prints them to the console.
 */
public class StreamGobbler extends Thread {
private InputStream is;
private Exception err = null;

public StreamGobbler(InputStream anIs) {
  is = anIs;
}

public void validate() throws XGGRunException {
  if (err != null)
    throw new XGGRunException(err.getMessage());
}

public void run() {
  try {
    BufferedReader br = new BufferedReader(new
InputStreamReader(is));
    String line = null;
    while ((line = br.readLine()) != null)
      System.out.println(line);
  } catch (IOException e) {
    err = e;
  }
}
}

**********
FILE: sp/compiler/FileClassLoader.java

package sp.compiler;

import java.io.*;

public class FileClassLoader extends ClassLoader {
private String root;

public FileClassLoader(String rootDir) {
  if (rootDir == null)
    throw new IllegalArgumentException("Null root
directory");
  root = rootDir;
}

@SuppressWarnings("unchecked")
protected Class loadClass(String name, boolean
resolve)
  throws ClassNotFoundException {
  //Since all support classes of loaded class use
same class loader
  //must check subclass cache of classes for things
like Object
  Class c = findLoadedClass(name);
  if (c == null) {
    try {
      c = findSystemClass(name);
    } catch (Exception e) {}
  }

  if (c == null) {
    // Convert class name argument to filename
    // Convert package names into subdirectories
    String filename = name.replace('.',
File.separatorChar) + ".class";

    try {
      byte data[] = loadClassData(filename);
      c = defineClass(name, data, 0, data.length);

      if (c == null)
        throw new ClassNotFoundException(name);
    } catch (IOException e) {
      throw new ClassNotFoundException("Error
reading file: " + filename);
    }
  }

  if (resolve)
    resolveClass(c);
  return c;
}
```

```
private byte[] loadClassData(String filename)
throws IOException {
  File f = new File(root, filename);

  int size = (int) f.length();
  byte buff[] = new byte[size];

  FileInputStream fis = new FileInputStream(f);
  DataInputStream dis = new DataInputStream(fis);

  dis.readFully(buff);
  dis.close();

  return buff;
}
}

**********
FILE: sp/compiler/XGGSupported.java

package sp.compiler;

import sp.format.*;
import sp.format.code.*;
import java.util.*;
import javax.swing.*;

/**
 * Contains the methods used to validate the XGG
format. Since the XGG format
 * is dynamic and not strict, This is used than
common XML validation
 * techniques. It also caches validations for further
uses.
 */
public final class XGGSupported {
private static LinkedList<Class<?>>
supportedElems = new LinkedList<Class<?>>();

private static final String viewTags[] = new String[]
{ "Arc", "Border",
    "Component", "CubicCurve", "Ellipse",
"GeneralPath", "Image", "Line",
    "Paint", "Path", "QuadCurve", "Rectangle",
"RoundRectangle", "Stroke",
    "Text" };

private static final HashMap<String, String>
elementmaps = new HashMap<String, String>();

private static final HashMap<String, Class<?>>
classmaps = new HashMap<String, Class<?>>();

private static final LinkedList<Class<?>>
supportedList = new LinkedList<Class<?>>();

static {
  String str[] = { "Box", "JButton", "JCheckBox",
"JComboBox", "JEditorPane",
      "JFormattedTextField", "JLabel", "JList",
"JMenuBar", "JMenu",
      "JMenuItem", "JCheckBoxMenuItem",
"JRadioButtonMenuItem", "JSeparator",
      "JPasswordField", "JPopupMenu", "JProgressBar",
"JRadioButton",
      "JScrollPane", "JSlider", "JSpinner", "JSplitPane",
"JTabbedPane",
      "JTable", "JTextArea", "JTextField", "JTextPane",
"JToggleButton",
      "JToolBar", "JTree", "JFrame", "JApplet", "JPanel",
"JDesktopPane",
      "JWindow", "JInternalFrame",
"JRadioButtonMenuItem", "JCheckBoxMenuItem" };
  for (int i = 0; i < str.length; i++) {
    try {
      Class<?> cls = Class.forName("javax.swing." +
str[i], false, null);
      supportedElems.add(cls);
      supportedList.add(cls);
      elementmaps.put(str[i], "javax.swing.");
      classmaps.put(str[i], cls);
    } catch (ClassNotFoundException e) {
      if (sp.XGGException.DODEBUG)
        sp.XGGException.debugStr("Uninitialized: " +
str[i]);
    }
  }
}

public static String getCachedLocation(String str) {
  return elementmaps.get(str);
}

public static Class<?> getCachedClass(String str) {
  return classmaps.get(str);
}

public static void addCachedClasses(String str,
String name, Class<?> cls) {
  elementmaps.put(str, name);
  classmaps.put(str, cls);
}

private static boolean isValidClass(Class<?> pcls,
Class<?> cls) {
  return sp.SPUtil.isClassAssignable(pcls, cls);
}

public static boolean isItemType(String str) {
  return ((str.equals("Item")) || (str.equals("Node"))
    || (str.equals("Column")) || (str.equals("Row")));
}

public static boolean legalItem(String str,
XGGElement elem) {
  if (elem == null)
    return false;

  Class<?> cls = elem.getTargetClass();

  if ((isValidClass(JComboBox.class, cls)) ||
(isValidClass(JList.class, cls)))
    return str.equals("Item");
  else if (isValidClass(JTree.class, cls))
    return str.equals("Node");
  else if (isValidClass(JTable.class, cls))
    return (str.equals("Column") || str.equals("Row"));

  return false;
}

public static boolean legalChildItem(String str,
XGGItem pelem) {
  if (pelem == null)
    return true;

  if (pelem.getTagType().equals("Node"))
    return str.equals("Node");
  else if (pelem.getTagType().equals("Row"))
    return str.equals("Item");

  return false;
}

public static boolean isViewType(String str) {
  return sp.SPUtil.binarySearchString(str, viewTags,
0, viewTags.length - 1) != null;
```

```
}

public static boolean legalViewChild(String str,
ViewType parent) {
  if (parent == null)
    return false;

  String pview = parent.getViewType();

  if (str.equals("Path"))
    return pview.equals("GeneralPath");
  else if ((str.equals("Border")) ||
(str.equals("Component")))
    return pview.equals("UI");
  else
    return pview.equals("Border") ||
pview.equals("Component");
}

public static boolean legalElemView(XGGElement
elem) {
  if (elem == null)
    return false;

  Class<?> cls = elem.getTargetClass();

  return ((isValidClass(JButton.class, cls))
    || (isValidClass(JCheckBox.class, cls))
    || (isValidClass(JRadioButton.class, cls))
    || (isValidClass(JToggleButton.class, cls))
    || (isValidClass(JMenuItem.class, cls))
    || (isValidClass(JCheckBoxMenuItem.class, cls))
    || (isValidClass(JRadioButtonMenuItem.class, cls))
    || (isValidClass(JProgressBar.class, cls))
    || (isValidClass(JLabel.class, cls)) ||
(isValidClass(JPanel.class, cls))
    || (isValidClass(JToolBar.class, cls)) ||
(isValidClass(Box.class, cls)));
}

public static boolean legalChildren(Class<?> cls,
XGGElement elem) {
  if (elem == null)
    return true;

  Class<?> parentcls = elem.getTargetClass();

  if ((isValidClass(JFrame.class, parentcls))
    || (isValidClass(JApplet.class, parentcls))
    || (isValidClass(JInternalFrame.class, parentcls)))
{
    if ((isValidClass(JComponent.class, cls))
      || (isValidClass(JMenuBar.class, cls)))
      return true;
  } else if ((isValidClass(JPanel.class, parentcls))
    || (isValidClass(JSplitPane.class, parentcls))
    || (isValidClass(JTabbedPane.class, parentcls))
    || (isValidClass(Box.class, parentcls))
    || (isValidClass(JScrollPane.class, parentcls))
    || (isValidClass(JToolBar.class, parentcls))) {
    if (isValidClass(JComponent.class, cls))
      return true;
  } else if (isValidClass(JMenuBar.class, parentcls)) {
    if (isValidClass(JMenu.class, cls))
      return true;
  } else if ((isValidClass(JMenu.class, parentcls))
    || (isValidClass(JPopupMenu.class, parentcls))) {
    if ((isValidClass(JMenuItem.class, cls))
      || (isValidClass(JSeparator.class, cls)))
      return true;
  } else if (isValidClass(JDesktopPane.class,
parentcls)) {
    if (isValidClass(JInternalFrame.class, cls))
      return true;
  }

  if ((isValidClass(JPopupMenu.class, cls))
    && (isValidClass(JComponent.class, parentcls)))
    return true;

  return false;
}

public static Class<?> isSpecialAttr(String str,
Class<?> cls) {
  CodeHandler cHnd = CodeHandler.getHandler(cls,
null);

  if (cHnd.isSpecialAttr(str))
    return cHnd.getClassAttr(str);

  return null;
}

public static boolean isElementSupported(Class<?>
cls) {
  if (supportedList.contains(cls))
    return true;

  Iterator<?> i = supportedElems.iterator();

  while (i.hasNext()) {
    Class<?> curcls = (Class<?>) i.next();
    if (sp.SPUtil.isClassAssignable(curcls, cls)) {
      supportedList.add(cls);
      return true;
    }
  }

  return false;
}
}

**********
FILE: sp/compiler/XGGCompileException.java

package sp.compiler;

import sp.*;

/**
 * The exception that indicates errors in
compilation. It does nothing by
 * itself and is normally thrown by methods that
catches <code>
 * XGGElementException</code>.
 */
public class XGGCompileException extends
XGGException {
static final long serialVersionUID = 'c' * 'o' * 'm' * 'p'
* 'i' * 'l' * 'e';

private XGGException err;
private CompilerMessage msg;

public XGGCompileException(XGGException anErr,
CompilerMessage aMsg) {
  err = anErr;
  msg = aMsg;
}

public CompilerMessage getCMessage() {
  return msg;
}

public int getLineNumber() {
  if (err != null)
    return err.getLineNumber();
```

```
  return 0;
}

public int getColumnNumber() {
  if (err != null)
    return err.getColumnNumber();
  return 0;
}

public String getError() {
  if (err != null)
    return err.getError();
  return null;
}
}
```

****
FILE: sp/compiler/XGGRunException.java

```
package sp.compiler;

/**
 * Thrown to indicate an error in an executed
runtime program.
 */
public class XGGRunException extends Exception {
static final long serialVersionUID = 'r' * 'u' * 'n';
private String error = null;

public XGGRunException(String anError) {
  error = anError;
}

public String getError() {
  return error;
}
}
```

****
FILE: sp/format/TreeType.java

```
package sp.format;

/**
 * Interface that represents an item in the plugin
tree view. Has methods used
 * by the treeview to draw the items in the tree.
 */
public interface TreeType {
  public String getIconName();

  public Object[] getTreeChildren();

  public Object getTreeParent();
}
```

****
FILE: sp/format/XGGFormat.java

```
package sp.format;

import sp.code.*;
import sp.compiler.*;
import java.io.*;
import java.util.*;
import java.beans.*;
import java.lang.reflect.*;

/**
 * The main class of the internal format, holds the
tree structure and other
 * properties if the XGG file.
 */
public class XGGFormat implements TreeType {

public static final int MTYPE_JFRAME = 1,
MTYPE_JAPPLET = 2, MTYPE_JOBJECT = 3;

private boolean doMain = true;
private boolean mapFlag = false;
private int iMType = 0;
private String sName = null;
private String rootName = null;
private String sPackage = null;
private File file = null;

private XGGElement headElem = null;
private CompilerOpts cOpts = null;
private XGGModel model = null;

private LinkedList<String> imports = new
LinkedList<String>();
private LinkedList<String> importsFile = new
LinkedList<String>();
private LinkedList<JVar> idnames = new
LinkedList<JVar>();
private LinkedList<XGGElement> elements = new
LinkedList<XGGElement>();
private LinkedList<JEvent> events = new
LinkedList<JEvent>();
private LinkedList<XGGAction> actions = new
LinkedList<XGGAction>();
private LinkedList<CompilerMessage> messages =
new LinkedList<CompilerMessage>();

private HashMap<String, Integer> tempvarnames
= new HashMap<String, Integer>();
private HashMap<String, Method> eventnames =
new HashMap<String, Method>();
private HashMap<String, XGGStyle> styles = new
HashMap<String, XGGStyle>();

public XGGFormat(CompilerOpts aCOpts, File aFile)
{
  cOpts = aCOpts;
  file = aFile;

  imports.add("javax.swing.");
  imports.add("java.awt.");
  imports.add("javax.swing.event.");
  imports.add("java.awt.event.");
  importsFile.add("javax.swing.");
}

public void setDoMain(boolean atype) {
  doMain = atype;
}

public boolean isDoMain() {
  return doMain;
}

public void setType(int atype) {
  iMType = atype;
  if (iMType != MTYPE_JFRAME)
    doMain = false;
}

public int getType() {
  return iMType;
}

public void setName(String str) {
  sName = str;
}

public String getName() {
  return sName;
}
```

```
public void setRootName(String str) {
  rootName = str;
}

public File getFile() {
  return file;
}

public CompilerOpts getOptions() {
  return cOpts;
}

public LinkedList<CompilerMessage>
getMessages() {
  return messages;
}

public XGGElement getHeadElement() {
  return headElem;
}

public void setModel(XGGModel aModel) {
  model = aModel;
  if (model.getModelType() ==
XGGModel.MODEL_CLASS) {
    idnames.add(new JVar(Modifier.PROTECTED,
"model", model.getTargetType(),
      "null"));
  } else if ((model.getModelType() ==
XGGModel.MODEL_CLASSLIST)
    || (model.getModelType() ==
XGGModel.MODEL_DBASE)) {
    addImport("java.util");

    if (model.getModelType() ==
XGGModel.MODEL_DBASE) {
      addImport("java.sql");
      addImport("javax.sql");
      idnames.add(new JVar(Modifier.PROTECTED,
"sqlerror", "SQLException",
        "null"));

      idnames.add(new JVar(Modifier.PROTECTED,
"model", model.getTargetType(),
        "null"));
    } else
      idnames.add(new JVar(Modifier.PROTECTED,
"model", cOpts.getGeneric(model
        .getTargetType(), model.getName()), "null"));

    idnames.add(new JVar(Modifier.PROTECTED,
"modelGroup", cOpts.getGeneric(
      "HashMap", "Object, Object"), "new "
      + cOpts.getGeneric("HashMap", "Object,
Object") + "()"));
  }
  if (iMType != MTYPE_JAPPLET)
    idnames.add(new JVar(Modifier.PUBLIC |
Modifier.STATIC | Modifier.FINAL,
      "MODELCLASS", cOpts.getGeneric("Class", "?"),
model.getTargetType()
        + ".class"));
}

public XGGModel getModel() {
  return model;
}

public void setPackage(String str) {
  sPackage = str;
}

public String getPackage() {
  return sPackage;
}

public JEvent addEvent(JEvent evt) {
  Iterator<?> i = events.iterator();
  while (i.hasNext()) {
    JEvent curevt = (JEvent) i.next();

    if (curevt.checkEqual(evt)) {
      curevt.setTemp(false);
      return curevt;
    }
  }

  events.add(evt);
  return evt;
}

public XGGElement getElementByName(String str) {
  Iterator<XGGElement> i = elements.iterator();
  while (i.hasNext()) {
    XGGElement elem = i.next();
    if (str.equals(elem.getName()))
      return elem;
  }
  return null;
}

public void addElement(XGGElement elem,
XGGElement pelem) {
  if (headElem == null) {
    headElem = elem;
    headElem.setHeadElement();
  }

  if (pelem != null)
    pelem.addChildren(elem);

  elements.add(elem);

  getApplyStyle(elem);
}

public void getApplyStyle(XGGElement elem) {
  String elemstyle =
elem.getAttributeValue("styleid");
  if (elemstyle == null)
    elemstyle = sp.SPUtil.STYLE_DEFAULT;
  String styleid = elem.getTargetClass().getName()
+ " " + elemstyle;

  XGGStyle style = (XGGStyle) styles.get(styleid);
  if (style != null) {
    style.applyStyle(elem);
    if (sp.XGGException.DODEBUG)
      sp.XGGException.debugStr("Apply Style "
        + elem.getClassName()
        + '|'
        + style.getStyleId()
        + " to "
        + ((elem.getName() != null) ?
elem.getName() : elem.getClassName()
          + '(' + elem.getLineNumber() + ')'));
  }
}

public boolean addStyle(XGGStyle style) {
  String str = style.getHashName();

  if (styles.get(str) != null)
    return false;

  styles.put(str, style);
  return true;
```

```
}

public void addAction(XGGAction actn) {
  actions.add(actn);
  addImport("javax.swing");
  addImport("java.awt.event");
}

public boolean addImport(String str) {
  //for now always return true, no way to have java
know if a package exists

  str = str + ".";
  if (!importsFile.contains(str)) {
    if (sp.XGGException.DODEBUG)
      sp.XGGException.debugStr("Import added: " +
str);
    importsFile.add(str);
  }
  if (!imports.contains(str))
    imports.add(str);
  return true;
}

public Class<?> checkElement(String str) {
  String cachedloc =
XGGSupported.getCachedLocation(str);
  if (cachedloc != null)
    return XGGSupported.getCachedClass(str);

  Class<?> result =
checkElementViaClassLoader(str, ClassLoader
    .getSystemClassLoader());

  if (result != null)
    return result;

  if (file.getAbsoluteFile().getParent() != null) {
    try {
      String fname = file.getAbsoluteFile().getParent()
+ File.separator;
      result = checkElementViaClassLoader(str, new
FileClassLoader(fname));
    } catch (Exception e) {}
  }

  return result;
}

protected Class<?>
checkElementViaClassLoader(String str,
ClassLoader clsld) {
  Iterator<?> i = imports.iterator();
  Class<?> result = null;

  while (i.hasNext()) {
    try {
      String curstr = (String) i.next();
      result = Class.forName(curstr + str, false, clsld);
      if (result != null) {
        if (importsFile.indexOf(curstr) < 0) {
          if (sp.XGGException.DODEBUG)
            sp.XGGException.debugStr("Import added: "
+ curstr);

          importsFile.add(curstr);
        }
        XGGSupported.addCachedClasses(str, curstr,
result);
        return result;
      }
    } catch (ClassNotFoundException e) {}
  }

  if (sPackage != null) {
    try {
      result = Class.forName(sPackage + '.' + str,
false, clsld);
      if (result != null)
        return result;
    } catch (ClassNotFoundException e) {}
  }

  try {
    result = Class.forName(str, false, clsld);
    if (result != null) {
      String clsstr = result.getName();
      if (clsstr.lastIndexOf('.') >= 0) {
        clsstr = clsstr.substring(0,
clsstr.lastIndexOf('.'));
        addImport(clsstr);
      }
      return result;
    }
  } catch (ClassNotFoundException e) {}

  return result;
}

public boolean addIdNames(String str, String type,
String initval) {
  Iterator<?> i = idnames.iterator();
  while (i.hasNext())
    if (str.equals(((JVar) i.next()).getName()))
      return false;

  Method curstr = (Method) eventnames.get(str);
  if (curstr != null)
    return false;

  idnames.add(new JVar(Modifier.PROTECTED, str,
type, initval));

  return true;
}

public boolean addIdNames(String str, String type)
{
  return addIdNames(str, type, null);
}

public String checkIdName(String str) {
  Iterator<?> i = idnames.iterator();
  while (i.hasNext()) {
    JVar var = (JVar) i.next();
    if (str.equals(var.getName()))
      return var.getType();
  }

  Method curstr = (Method) eventnames.get(str);
  if (curstr != null)
    return "event";

  return null;
}

public boolean addIdNamesEvent(String str, Method
type) {
  Method curstr = (Method) eventnames.get(str);

  if (curstr != null) {
    Class<?> cls1[] = curstr.getParameterTypes();
    Class<?> cls2[] = type.getParameterTypes();

    if (cls1.length != cls2.length)
      return false;

    for (int i = 0; i < cls1.length; i++)
```

```
      if (!cls1[i].getName().equals(cls2[i].getName()))
        return false;
    }

    Iterator<?> i = idnames.iterator();
    while (i.hasNext())
      if (str.equals(((JVar) i.next()).getName()))
        return false;

    eventnames.put(str, type);
    return true;
  }

  public void giveWarning(String str, XGGElement
elem, XGGTag tag) {
    int warnline = (tag != null) ?
tag.getLineNumber() : elem.getLineNumber();
    int warncol = (tag != null) ?
tag.getColumnNumber() :
elem.getColumnNumber();

    String errStr = "In element '" +
elem.getClassName() + "': "
      + (tag != null ? "In Tag " + tag.getTagType() + ":
" : "") + str;

    sp.SPUtil.errPrint(rootName, warnline, warncol,
"Warning: " + rootName + ": "
      + elem.getLineNumber() + "," +
elem.getColumnNumber() + ": " + errStr);

    messages.add(new CompilerMessage(rootName,
warnline, warncol, errStr));
  }

  public void giveWarning(String str, XGGElement
elem) {
    giveWarning(str, elem, null);
  }

  public PropertyDescriptor checkAttr(Class<?>
cName, String atts) {
    BeanInfo biInfo = sp.SPUtil.getBeanInfo(cName);

    PropertyDescriptor pd[] =
biInfo.getPropertyDescriptors();
    PropertyDescriptor pdt =
sp.SPUtil.binarySearchAtts(atts, pd, 0,
      pd.length - 1);
    if (pdt != null)
      return pdt;
    return null;
  }

  public Method checkEventAttr(Class<?> cName,
String atts) {
    BeanInfo biInfo = sp.SPUtil.getBeanInfo(cName);

    EventSetDescriptor ed[] =
biInfo.getEventSetDescriptors();
    for (int i = 0; i < ed.length; i++) {
      Method meth[] = ed[i].getListenerMethods();
      for (int j = 0; j < meth.length; j++) {
        if (atts.equals(meth[j].getName())) {
          String str = ed[i].getListenerType().getName();
          checkElement(str.substring(str.lastIndexOf('.')
+ 1));
          return meth[j];
        }
      }
    }
    return null;
  }

  public String[] getEventList() {
    String result[] = new String[eventnames.size()];

    Iterator<?> e = eventnames.keySet().iterator();
    int i = 0;
    while (e.hasNext()) {
      String str = (String) e.next();
      Method meth = (Method) eventnames.get(str);

      String retstr = "void";
      Class<?> retcls = meth.getReturnType();
      if (retcls != null)
        retstr =
sp.SPUtil.getShortName(retcls.getName());

      result[i] = retstr + " " + str;
      i++;
    }
    return result;
  }

  public void initMapVar() {
    if (!mapFlag) {
      mapFlag = true;

      JVar jv = new JVar(Modifier.PRIVATE,
"__HashMap", cOpts.getGeneric(
        "HashMap", "Object, Object"), "new "
        + cOpts.getGeneric("HashMap", "Object,
Object") + "()");
      idnames.add(jv);

      addImport("java.util");
    }
  }

  public String addHashMap(XGGElement elem) {
    initMapVar();

    StringBuffer result = new StringBuffer();

    result.append("__HashMap.put(");
    result.append('"');
    result.append(elem.getUsableName());
    result.append('"');
    result.append(", ");
    result.append(elem.getUsableName());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    return result.toString();
  }

  public String getConstantValue(String str, Class<?>
targetcls) {
    int last = str.lastIndexOf('.');
    if (last <= 0)
      return null;

    String value = str.substring(0, last);

    Class<?> cls = null;
    try {
      cls = Class.forName(value, false, null);
    } catch (ClassNotFoundException e) {}

    if (cls == null)
      cls = checkElement(value);

    if (cls == null)
      return null;

    Field[] flds = cls.getDeclaredFields();
```

```
  value = str.substring(last + 1);

  for (int i = 0; i < flds.length; i++) {
    if ((value.equals(flds[i].getName()))
      && (sp.SPUtil.isClassAssignable(targetcls,
flds[i].getType()))) {
      int imod = flds[i].getModifiers();
      if ((Modifier.isFinal(imod)) && (!
Modifier.isPrivate(imod))
        && (!Modifier.isProtected(imod)))
        return str;
    }
  }

  return null;
}

public String getNewName(Class<?> cls) {
  String stype = sp.SPUtil.getSubClassName(cls,
"_");

  Integer i = (Integer) tempvarnames.get(stype);

  if (i == null)
    i = new Integer(1);

  tempvarnames.put(stype, new Integer(i.intValue()
+ 1));

  return "__" + stype + i;
}

public void writeImports(JClass jc) {
  if (sPackage != null)
    jc.setPackage(sPackage);

  Iterator<?> i = importsFile.iterator();
  while (i.hasNext()) {
    jc.addImport(i.next() + "*");
  }
}

public void writeVars(JClass jc) {
  Iterator<JVar> i = idnames.iterator();
  while (i.hasNext()) {
    jc.addVar(i.next());
  }
}

public JMethod methodFromEvent(String str, int
imod) {
  JMethod meth = new JMethod((Method)
eventnames.get(str));
  meth.setName(str);
  meth.setModifier(imod);
  return meth;
}

public void writeEvents(JClass jc, boolean
isAbstract) {
  int imod = Modifier.PUBLIC;
  if (isAbstract)
    imod = imod | Modifier.ABSTRACT;
  Iterator<?> e = eventnames.keySet().iterator();
  while (e.hasNext())
    jc.addMethod(methodFromEvent((String)
e.next(), imod));
}

public String writeFile() throws
XGGCompileException {
  JClass jc = new JClass(sName,
headElem.getClassName());

  try {
    JRootWrite(jc, iMType == MTYPE_JAPPLET);
    if (!cOpts.isCheckOnly()) {
      writeImports(jc);
      writeVars(jc);
    }

    if (sp.XGGException.DODEBUG)
      sp.XGGException.debugStr("\n" +
headElem.print(1));

    return jc.toString();
  } catch (XGGElementException e) {
    sp.XGGException xggex = e.getException();
    XGGElement elem = e.getElement();
    if (xggex != null) {
      String errStr = "In element '" +
elem.getClassName() + "': "
        + xggex.getError();

      sp.SPUtil.errPrint(rootName,
xggex.getLineNumber(), xggex
        .getColumnNumber(), "Error: " + rootName +
":" + elem.getLineNumber()
        + "," + elem.getColumnNumber() + ": " +
errStr);

      throw new XGGCompileException(e, new
CompilerMessage(rootName, xggex
        .getLineNumber(), xggex.getColumnNumber(),
errStr));
    }

    throw new XGGCompileException(e, new
CompilerMessage(rootName, 0, 0,
      "Unknown Error!"));
  }
}

private StringBuffer eventsString(JMethod jm)
throws XGGElementException {
  StringBuffer result = new StringBuffer();
  boolean toeventimp = false;
  boolean parseonly = cOpts.isCheckOnly();

  Iterator<JEvent> i = events.iterator();
  while (i.hasNext()) {
    JEvent curevt = i.next();

    curevt.setSpecialName(getNewName(curevt.getT
argetClass()));

    if ((!curevt.isTemp()) && (!parseonly)) {
      JVar arg = new JVar(0, curevt.getSpecialName(),
sp.SPUtil
        .getShortName(curevt.getTargetClass().getNa
me()), null);

      arg.setQInitVal(curevt);
      jm.addVar(arg);
    }
    toeventimp = true;
  }

  Iterator<XGGAction> iact = actions.iterator();
  while (iact.hasNext()) {
    XGGAction curacts = iact.next();

    if (!parseonly) {
      JVar arg = curacts.outputVar();

      try {
        result.append(curacts.outputCode());
      } catch (XGGTagException e) {
```

```
        throw new XGGElementException(headElem,
e);
      }

      jm.addVar(arg);
    }
    toeventimp = true;
  }

  if ((toeventimp) && (!parseonly)) {
    addImport("java.awt.event");
    addImport("javax.swing.event");
  }

  return result;
}

private void JRootWrite(JClass jc, boolean isApplet)
throws XGGElementException {
  if (!cOpts.isPrototype())
    jc.setModifier(Modifier.PUBLIC |
Modifier.ABSTRACT);
  else
    jc.setModifier(Modifier.PUBLIC);

  if (!cOpts.isCheckOnly()) {
    JMethod jcons;
    StringBuffer sbJCons = new StringBuffer();

    if (!isApplet) {
      jcons = new JMethod(null, sName);
      jcons.setConstructor(true);

      sbJCons.append("super(");
      if (headElem.getTargetClass() ==
javax.swing.Box.class)
        sbJCons.append(sp.format.code.BoxCodeHandl
er
          .getBoxOrientation(headElem));
      sbJCons.append(");");
      sbJCons.append(sp.SPUtil.lineBreak());

      if (model != null) {
        jcons.setArgs(new JArg[] { new JArg("model",
model.getVarType(cOpts
          .isMode1_5())) });
        sbJCons.append("this.model = model;");
        sbJCons.append(sp.SPUtil.lineBreak());

        JMethod jcons2 = new JMethod(null, sName);
        jcons2.setConstructor(true);
        jcons2.setBody(new StringBuffer("this(null);"));
        jc.addMethod(jcons2);
      }
    } else {
      jcons = new JMethod(null, "init");
      doMain = false;
    }

    sbJCons.append("initialize();");
    sbJCons.append(sp.SPUtil.lineBreak());
    sbJCons.append("createComponents();");

    jcons.setBody(sbJCons);
    jc.addMethod(jcons);

    JMethod jinit = new JMethod(null, "initialize");

    jc.addMethod(jinit);
  }

  JMethod jm = new JMethod(null,
"createComponents");

  if (!cOpts.isCheckOnly())
    jc.addMethod(jm);

  if (sp.XGGException.DODEBUG)
    sp.XGGException.debugStr("Starting to create
internal format..");

  headElem.assignNames();

  XGGView view = headElem.getView();
  if (view != null) {
    try {
      view.addShapeCodes(jc);
    } catch (sp.XGGException e) {
      throw new XGGElementException(headElem, e);
    }
  }

  if (!cOpts.isPrototype())
    headElem.collectEvents();

  if (!cOpts.isCheckOnly()) {
    jc.setBraceType(cOpts.getBraceType());
    jc.setIndents(cOpts.getIndents());
  } else
    jc.setIndents(0);

  if (cOpts.isCheckOnly())
    ;
  else if (!cOpts.isPrototype())
    writeEvents(jc, true);
  else if (!isApplet) {
    JMethod mainmeth =
JMethod.makeMainMethod();

    StringBuffer sbJMain = new StringBuffer();
    sbJMain.append("new ");
    sbJMain.append(jc.getName());
    sbJMain.append("();");

    mainmeth.setBody(sbJMain);
    jc.addMethod(mainmeth);
  }

  StringBuffer sbCCompsMeth = new StringBuffer();
  StringBuffer sbTemp = eventsString(jm);
  StringBuffer sbTemp2 = headElem.attrToString(jc);
  if (!cOpts.isCheckOnly()) {
    sbCCompsMeth.append(sbTemp);
    sbCCompsMeth.append(sbTemp2);
    jm.setBody(sbCCompsMeth);
  }
}

public String toString() {
  return getName();
}

public Object[] getTreeChildren() {
  LinkedList<Object> ll = new
LinkedList<Object>();

  if (sPackage != null)
    ll.add(new BasicTreeLeaf(sPackage,
"package_obj.gif", this));

  if (headElem != null)
    ll.add(headElem);

  if (model != null)
    ll.add(model);

  ll.add(new TreeImport(importsFile, this));
```

```
  return ll.toArray(new Object[] { ll.size() });
}

public Object getTreeParent() {
  return null;
}

public String getIconName() {
  return "";
}

class TreeImport implements TreeType {
LinkedList<String> llimps;
Object parent;

public TreeImport(LinkedList<String> anLL, Object
aParent) {
  llimps = anLL;
  parent = aParent;
}

public String getIconName() {
  return "th_single.gif";
}

public Object[] getTreeChildren() {
  LinkedList<Object> ll = new
LinkedList<Object>();

  Iterator<String> iter = llimps.iterator();
  while (iter.hasNext())
    ll.add(new BasicTreeLeaf(iter.next(),
"private_co.gif", this));

  return ll.toArray(new Object[ll.size()]);
}

public Object getTreeParent() {
  return parent;
}

public String toString() {
  return "Imports";
}
}

public static class BasicTreeLeaf implements
TreeType {
String name, icon;
Object leafp;

public BasicTreeLeaf(String aName, String anIcon,
Object aParent) {
  name = aName;
  icon = anIcon;
  leafp = aParent;
}

public String getIconName() {
  return icon;
}

public Object[] getTreeChildren() {
  return new Object[0];
}

public Object getTreeParent() {
  return leafp;
}

public String toString() {
  return name;
}
}
```

```
}

**********
FILE: sp/format/XGGTag.java

package sp.format;

import org.xml.sax.*;
import org.xml.sax.helpers.*;

/**
 * The base class for XGG tag elements, contains
tag location and attributes.
 */
public abstract class XGGTag {
protected XGGFormat xggMain = null;
protected AttributesImpl atts = null;
protected Class<?> targetClass = null;

protected XGGTag(AttributesImpl anAtts) {
  atts = anAtts;
}

public Class<?> getTargetClass() {
  return targetClass;
}

public String getClassName() {
  return
sp.SPUtil.getShortName(targetClass.getName());
}

public XGGFormat getMain() {
  return xggMain;
}

public String getTagType() {
  return getClassName();
}

public String getErrorHeader() {
  return "Error in Tag";
}

public String getTagHeader() {
  return "Tag";
}

public boolean isAttributeExist(String qName) {
  return atts.getIndex(qName) >= 0;
}

public void addAttribute(String qName, String value,
String qType) {
  atts.addAttribute(null, null, qName, qType, value);
}

public String updateContentAttribute(String qName,
String value, String extra) {
  String err = null;
  int index = atts.getIndex(qName);

  if (index < 0) {
    addAttribute(qName, value,
String.class.getName());
    addAttribute('_' + qName, "true", "boolean");
    return null;
  }

  int existindex = atts.getIndex('_' + qName);

  if (existindex < 0) {
    err = "Attribute '" + qName + "' is used more
than once."
```

```
      + " The first one is overwritten.";

    atts.setValue(index, value);

    addAttribute('_' + qName, "true", "boolean");
   } else {
    String curval = atts.getValue(index);
    atts.setValue(index, curval + extra + value);
   }

   return err;
 }

 public void addStyleAttribute(String sName, String
qName, String value,
   String qType) {
   atts.addAttribute(null, sName, qName, qType,
value);
 }

 public int countAttributes() {
   return atts.getLength();
 }

 public void installPosition(Locator locator) {
   addAttribute("_line",
Integer.toString(locator.getLineNumber()), "int");
   addAttribute("_col",
Integer.toString(locator.getColumnNumber()), "int");
 }

 public int getLineNumber() {
   String str = atts.getValue("_line");

   if (str != null)
    try {
     return Integer.parseInt(str);
    } catch (NumberFormatException e) {}
   return 0;
 }

 public int getColumnNumber() {
   String str = atts.getValue("_col");

   if (str != null)
    try {
     return Integer.parseInt(str);
    } catch (NumberFormatException e) {}
   return 0;
 }

 public String getAttributeValue(String qName) {
   int index = atts.getIndex(qName);
   if (index >= 0)
    return atts.getValue(index);
   return null;
 }

 public String debugWrite(int spaces, String str) {
   StringBuffer result = new StringBuffer();

   result.append(sp.SPUtil.getShortName(targetClass.
getName()));
   result.append(" > ");
   result.append(str);
   result.append(": ");
   result.append(getLineNumber());
   result.append(',');
   result.append(getColumnNumber());
   spaces += 1;

   for (int i = 0; i < atts.getLength(); i++) {
    if ((!atts.getQName(i).startsWith("_")) && (!
atts.getQName(i).equals("id"))) {
     result.append('\n');
     for (int j = 0; j < spaces; j++)
      result.append(' ');
     result.append(atts.getQName(i));
     result.append('(');
     result.append(atts.getType(i));
     result.append(')');
     result.append('=');
     result.append(atts.getValue(i));
    }
   }
   result.append('\n');

   return result.toString();
 }
}
```

****

FILE: sp/format/XGGElement.java

```
package sp.format;

import sp.code.*;
import sp.compiler.*;
import sp.format.code.*;
import java.util.*;
import java.beans.*;
import java.lang.reflect.*;
import org.xml.sax.helpers.*;

/**
 * Represents an element (a JComponent) in an XGG
file. It contains all the
 * info needed by an element to be written in java
code.
 */
public class XGGElement extends XGGTag
implements TreeType {
private String sName = null;
private String sLocalName = null;
private BeanInfo biInfo = null;
private boolean created = false;
private boolean inloop = false;
private XGGElement parent = null;
private XGGView view = null;

private LinkedList<XGGElement> children = new
LinkedList<XGGElement>();
private LinkedList<XGGItem> items = new
LinkedList<XGGItem>();
private LinkedList<XGGAttribute> crossattr = new
LinkedList<XGGAttribute>();
private LinkedList<JEvent> events = new
LinkedList<JEvent>();

private HashMap<String, XGGBind> binds = new
HashMap<String, XGGBind>();

public XGGElement(XGGFormat anXGG, String
aName, Class<?> aClass,
  AttributesImpl anAtts, BeanInfo aBi) {
  super(anAtts);
  xggMain = anXGG;
  sName = aName;
  targetClass = aClass;
  biInfo = aBi;
}

public String getName() {
  return sName;
}

public String getUsableName() {
  if (sName != null)
```

```
    return sName;
  else
    return sLocalName;
}

public void setHeadElement() {
  sName = "this";
  created = true;
}

public XGGElement getParent() {
  return parent;
}

public void setView(XGGView aView) {
  view = aView;
}

public XGGView getView() {
  return view;
}

public boolean isHead() {
  return (xggMain.getHeadElement() == this);
}

public boolean isCreated() {
  return created;
}

public void setCreated(boolean b) {
  created = b;
}

public boolean isInLoop() {
  return inloop;
}

public void setInLoop(boolean b) {
  inloop = b;
}

public void addChildren(XGGElement xggElem) {
  children.add(xggElem);
  xggElem.parent = this;
  xggElem.inloop = this.inloop;
}

public Object[] getChildren() {
  return children.toArray(new
Object[children.size()]);
}

public void addCrossRef(XGGAttribute xggAtts) {
  crossattr.add(xggAtts);
}

public void addItem(XGGItem anItem) {
  items.add(anItem);
}

public void addBind(String str, XGGBind aBind) {
  binds.put(str, aBind);
}

public String print(int spaces) {
  StringBuffer result = new StringBuffer();

  result.append(debugWrite(spaces,
getUsableName()));

  Iterator<?> i = children.iterator();
  while (i.hasNext()) {
    for (int j = 0; j < spaces; j++)
      result.append(' ');
    result.append(((XGGElement)
i.next()).print(spaces + 1));
  }
  return result.toString();
}

public void assignNames() {
  String varname = sName;

  if ((varname == null) && (!created))
    sLocalName =
xggMain.getNewName(targetClass);

  Iterator<?> i = children.iterator();
  while (i.hasNext()) {
    ((XGGElement) i.next()).assignNames();
  }
}

public void collectEvents() {
  HashMap<String, JEvent> ht = new
HashMap<String, JEvent>();

  for (int i = 0; i < atts.getLength(); i++) {
    String qName = atts.getQName(i);
    if ((qName.equals("id")) || (qName.charAt(0) ==
'_')
      || (qName.equals("text")))
      ;
    else {
      FeatureDescriptor fdesc = getDescriptor(i);
      if (fdesc instanceof EventSetDescriptor) {
        EventSetDescriptor edesc =
(EventSetDescriptor) fdesc;

        if (sp.XGGException.DODEBUG)
          sp.XGGException.debugStr("Event Collect: "
+ getUsableName() + ": "
            + edesc.getName() + "." + qName + " -> "
+ atts.getValue(i));

        JEvent evt = (JEvent) ht.get(edesc.getName());
        if (evt == null) {
          evt = new JEvent(edesc);
          ht.put(edesc.getName(), evt);
        }

        evt.addListenerMethod(atts.getValue(i),
qName);
      }
    }
  }

  Iterator<?> e = ht.values().iterator();
  while (e.hasNext()) {
    JEvent getEvt = xggMain.addEvent((JEvent)
e.next());
    events.add(getEvt);
  }

  Iterator<?> i = children.iterator();
  while (i.hasNext()) {
    ((XGGElement) i.next()).collectEvents();
  }
}

public FeatureDescriptor getDescriptor(int num) {
  String attname = atts.getQName(num);

  PropertyDescriptor pd[] =
biInfo.getPropertyDescriptors();
  PropertyDescriptor pdt =
sp.SPUtil.binarySearchAtts(attname, pd, 0,
```

```
    pd.length - 1);
  if (pdt != null)
    return pdt;

  EventSetDescriptor ed[] =
biInfo.getEventSetDescriptors();
  for (int i = 0; i < ed.length; i++) {
    Method meth[] = ed[i].getListenerMethods();
    for (int j = 0; j < meth.length; j++) {
      if (attname.equals(meth[j].getName())) {
        return ed[i];
      }
    }
  }
  return null;
}

public void printAttribute(XGGAttribute xggatt,
String varname,
  StringBuffer result) throws XGGElementException
{
  try {
    String str = xggatt.getCode();
    boolean cmt = xggatt.isCommentedOut();

    if (getMain().getOptions().isCheckOnly())
      return;

    if ((!cmt) || (xggMain.getOptions().isComments()))
{
      if (cmt)
        result.append("//");

      if (!isHead()) {
        result.append(varname);
        result.append('.');
      }
      result.append(str);
    }
  } catch (XGGForwardRefException e) {
    XGGElement target = e.getElement();
    XGGAttribute targetatt = e.getAttribute();

    targetatt.setCrossRefName(varname);
    target.addCrossRef(targetatt);

    if (sp.XGGException.DODEBUG)
      sp.XGGException.debugStr("Cross reference : "
+ target.getName() + " : "
        + getUsableName() + " -> " +
targetatt.getName());
  } catch (XGGValueException e) {
    throw new XGGElementException(this, e);
  } catch (XGGSpecialAttributeException e) {
    throw new XGGElementException(this, e);
  }
}

public StringBuffer attrToString(JClass jc) throws
XGGElementException {
  StringBuffer result = new StringBuffer();
  StringBuffer mresult = null;
  StringBuffer meresult = null;
  StringBuffer sbTemp = null;
  String varname = sName;
  boolean varcreated = true;
  boolean parseonly =
getMain().getOptions().isCheckOnly();
  XGGModel mdl = null;
  CompilerOpts cOpts = xggMain.getOptions();

  if ((inloop) && (parent != null) && (!parent.inloop)
&& (!parseonly)) {
    mresult = new StringBuffer();
    mresult.append(sp.SPUtil.lineBreak());

    mdl = xggMain.getModel();

    if (mdl.getModelType() ==
XGGModel.MODEL_CLASSLIST) {
      mresult.append(cOpts.getGeneric("Iterator",
mdl.getName()));
      mresult.append(" __iterate = (model != null) ? "
        + "model.iterator() : null;");
      mresult.append(sp.SPUtil.lineBreak());

      mresult.append("while ((__iterate != null) &&
(__iterate.hasNext()))");
      mresult.append(cOpts.openBrace());

      JVar curvar = new JVar(0, "curmodel",
mdl.getName(), '(' + mdl.getName()
        + ")__iterate.next()");
      result.append(curvar);
      result.append(sp.SPUtil.lineBreak());

      curvar = new JVar(0, "curmap",
cOpts.getGeneric("HashMap",
        "Object, Object"), "new "
        + cOpts.getGeneric("HashMap", "Object,
Object") + "()");
      result.append(curvar);
      result.append(sp.SPUtil.lineBreak());

      result.append("modelGroup.put(curmodel,
curmap);");
      result.append(sp.SPUtil.lineBreak());

      result.append("curmap.put(\"model\",curmodel);
");
      result.append(sp.SPUtil.lineBreak());
    } else if (mdl.getModelType() ==
XGGModel.MODEL_DBASE) {
      mresult.append("try");
      mresult.append(cOpts.openBrace());

      meresult = new StringBuffer();

      meresult.append("while ((model!= null) &&
(model.next()))");
      meresult.append(cOpts.openBrace());

      JVar curvar = new JVar(0, "curmap",
cOpts.getGeneric("HashMap",
        "Object, Object"), "new "
        + cOpts.getGeneric("HashMap", "Object,
Object") + "()");
      result.append(curvar);
      result.append(sp.SPUtil.lineBreak());

      result.append("curmap.put(\"row\",new
Integer(model.getRow()));");
      result.append(sp.SPUtil.lineBreak());
    }
  }

  result.append(sp.SPUtil.lineBreak());

  if ((varname == null) && (!created)) {
    varname = sLocalName;
    varcreated = false;
  }

  CodeHandler cHnd =
CodeHandler.getHandler(targetClass, this);

  if (!parseonly) {
    sbTemp = cHnd.commentCode();
```

```
    if (sbTemp != null)
      result.append(sbTemp);
  }

  sbTemp = cHnd.openCode(varname, varcreated);
  if ((sbTemp != null) && (!parseonly))
    result.append(sbTemp);

  if ((inloop) && (sName != null) && (parent != null)
&& (parent.inloop)
     && (!parseonly)) {
    result.append("modelGroup.put(");
    result.append(sName);
    result.append(", curmap);");
    result.append(sp.SPUtil.lineBreak());

    result.append("curmap.put(\"");
    result.append(sName);
    result.append("\", ");
    result.append(sName);
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
  }

  int texti = atts.getIndex("text");
  if (texti >= 0) {
    XGGAttribute xggatt = new XGGAttribute(this,
atts.getQName(texti), atts
       .getValue(texti), getDescriptor(texti));

    printAttribute(xggatt, varname, result);
  }

  for (int i = 0; i < atts.getLength(); i++) {
    String qName = atts.getQName(i);
    if (cHnd.isSpecialAttr(qName)) {
      sbTemp = cHnd.attrCode(qName,
atts.getValue(i));
      if ((sbTemp != null) && (!parseonly))
        result.append(sbTemp);
    } else if ((!qName.equals("id")) &&
(qName.charAt(0) != '_')
      && (!qName.equals("text")) && (!
qName.equals("styleid"))) {
      XGGAttribute xggatt = new XGGAttribute(this,
atts.getQName(i), atts
        .getValue(i), getDescriptor(i));

      printAttribute(xggatt, varname, result);
    }
  }

  Iterator<?> i = items.iterator();
  while (i.hasNext()) {
    XGGItem item = (XGGItem) i.next();

    sbTemp = cHnd.itemCode(item);
    if ((sbTemp != null) && (!parseonly))
      result.append(sbTemp);
  }

  i = events.iterator();
  while (i.hasNext()) {
    JEvent event = (JEvent) i.next();

    if (!isHead()) {
      result.append(varname);
      result.append('.');
    }
    result.append(event.toStringAttr(jc));
  }

  i = crossattr.iterator();
  while (i.hasNext()) {
    XGGAttribute xggatt = (XGGAttribute) i.next();

    printAttribute(xggatt, xggatt.getCrossRefName(),
result);
  }

  boolean structindent = cOpts.isStructIndent();
  int indent = cOpts.getIndents();

  i = children.iterator();
  while (i.hasNext()) {

    XGGElement child = (XGGElement) i.next();

    StringBuffer ccode = child.attrToString(jc);

    if (parseonly)
      ;
    if (!structindent)
      result.append(ccode);
    else {
      result.append(sp.SPUtil.indent(ccode.toString(),
indent));
      result.append(sp.SPUtil.lineBreak());
    }

    sbTemp = cHnd.childCode(child, parent);
    if ((sbTemp != null) && (!parseonly)) {

      if ((!inloop) && (child.inloop)) {
        mdl = xggMain.getModel();

        if (mdl.getModelType() ==
XGGModel.MODEL_CLASSLIST) {
          result.append(cOpts.indent(sbTemp.toString()
));
          result.append(sp.SPUtil.lineBreak());
          result.append('}');
          result.append(sp.SPUtil.lineBreak());
        } else if (mdl.getModelType() ==
XGGModel.MODEL_DBASE) {
          result.append(sp.SPUtil.indent(sbTemp.toStri
ng(), indent * 2));
          result.append(sp.SPUtil.lineBreak());

          result.append(sp.SPUtil.indent("}", indent));
          result.append(sp.SPUtil.lineBreak());
          result.append('}');
          result.append(sp.SPUtil.lineBreak());

          result.append("catch (SQLException e)");
          result.append(cOpts.openBrace());
          result.append(sp.SPUtil.indent("sqlerror = e;",
indent));
          result.append(sp.SPUtil.lineBreak());
          result.append('}');
          result.append(sp.SPUtil.lineBreak());
        }
      } else
        result.append(sbTemp);
    }

  }

  i = binds.keySet().iterator();
  while (i.hasNext()) {
    String str = (String) i.next();
    XGGBind bind = (XGGBind) binds.get(str);

    if (sp.XGGException.DODEBUG)
      sp.XGGException.debugStr("Binding: " +
bind.debugWrite());

    sbTemp = bind.writeBind();
```

```
    if (sbTemp != null)
      result.append(sbTemp);
  }

  sbTemp = cHnd.lateCode();
  if (sbTemp != null)
    result.append(sbTemp);

  if (mresult != null) {
    if (mdl.getModelType() ==
XGGModel.MODEL_CLASSLIST) {
      mresult.append(cOpts.indent(result.toString()));
      mresult.append(sp.SPUtil.lineBreak());
    } else if (mdl.getModelType() ==
XGGModel.MODEL_DBASE) {
      meresult.append(cOpts.indent(result.toString()))
;
      meresult.append(sp.SPUtil.lineBreak());
      mresult.append(cOpts.indent(meresult.toString(
)));
      mresult.append(sp.SPUtil.lineBreak());

    }
    return mresult;
  }
  return result;
}

public Object[] getTreeChildren() {
  LinkedList<Object> ll = new
LinkedList<Object>();

  for (int i = 0; i < atts.getLength(); i++)
    if (atts.getQName(i).charAt(0) != '_')
      ll.add(new ElementAttribute(atts.getQName(i),
atts.getValue(i), atts
        .getType(i), this, atts.getLocalName(i)));

  Iterator<XGGItem> iteri = items.iterator();
  while (iteri.hasNext())
    ll.add(iteri.next());

  if (view != null)
    ll.add(view);

  Iterator<XGGElement> iter = children.iterator();
  while (iter.hasNext())
    ll.add(iter.next());

  return ll.toArray(new Object[ll.size()]);
}

public Object getTreeParent() {
  if (isHead())
    return getMain();
  else
    return getParent();
}

public String toString() {
  if (isHead())
    return getMain().getName() + '(' +
getClassName() + ')';
  if (sName != null)
    return getClassName() + '(' + sName + ')';
  else
    return getClassName();
}

public String getIconName() {
  if (isHead())
    return "application-16x16.gif";
  else
    return "ch_callees.gif";
}

public class ElementAttribute implements TreeType
{
public String name;
public String value;
public String type;
public String styleid;
public XGGElement elem;

public ElementAttribute(String aName, String
aValue, String aType,
  XGGElement anElem, String aStyle) {
  name = aName;
  value = aValue;
  type = aType;
  elem = anElem;
  styleid = aStyle;
}

public String toString() {
  return name + ": " + value;
}

public Object[] getTreeChildren() {
  if ((styleid == null) || (styleid.trim().length() == 0))
    return new Object[] { "type: " + type };
  else
    return new Object[] { "type: " + type, "style: " +
styleid };
}

public Object getTreeParent() {
  return elem;
}

public String getIconName() {
  return "public_co.gif";
}
}
}

**********
FILE: sp/format/XGGStyle.java

package sp.format;

import org.xml.sax.helpers.*;

/**
 * Represents a style in an XGG file. When an XGG
element is created, registered
 * styles are checked if it matches the element's
styleid and type. If it is a
 * match, all styles attributes are added to the
element. It cannot override
 * existing element attributes.
 */
public class XGGStyle extends XGGTag {
String styleid = null;

public XGGStyle(Class<?> aClass, XGGFormat
aMain, AttributesImpl anAtts) {
  super(anAtts);
  targetClass = aClass;
  xggMain = aMain;

  styleid = getAttributeValue("styleid");
  if (styleid == null)
    styleid = sp.SPUtil.STYLE_DEFAULT;
}

public String getStyleId() {
  return styleid;
```

```
}

public String getHashName() {
  return targetClass.getName() + " " + styleid;
}

public boolean isEqual(XGGStyle style) {
  return ((style.targetClass == targetClass) &&
(style.styleid.equals(styleid)));
}

public void applyStyle(XGGElement elem) {
  for (int i = 0; i < atts.getLength(); i++) {
    String str = atts.getQName(i);
    if ((!str.equals("styleid")) && (str.charAt(0) != '_')
      && (!elem.isAttributeExist(str)))
      elem.addStyleAttribute(styleid,
atts.getQName(i), atts.getValue(i), atts
        .getType(i));
  }
}
}

*********************************
FILE: sp/format/XGGModel.java

package sp.format;

/**
 * The base class for a model used in the XGG file.
 */
public abstract class XGGModel implements
TreeType {
public static int MODEL_CLASS = 0,
MODEL_CLASSLIST = 1, MODEL_DBASE = 2;

protected static String mtype[] = { "Class", "List",
"Database" };

protected int type = 0;
protected XGGFormat xggMain;

protected XGGModel(XGGFormat aMain) {
  xggMain = aMain;
}

public int getModelType() {
  return type;
}

public String getTargetType() {
  return "";
}

public String getName() {
  return "";
}

public String getVarName() {
  return "model";
}

public String getVarType(boolean is1_5) {
  return getTargetType();
}

public String toString() {
  return "Model";
}

public String getIconName() {
  return "java_attach.gif";
}

public Object[] getTreeChildren() {
  return new Object[] { new
XGGFormat.BasicTreeLeaf("type: " + mtype[type],
    "private_co.gif", this) };
}

public Object getTreeParent() {
  return xggMain;
}

}

*********************************
FILE: sp/format/XGGClassModel.java

package sp.format;

import sp.format.code.*;
import java.lang.reflect.*;
import java.beans.*;
import java.util.*;

/**
 * A Model from a class file in Java. It can be a single
model or a
 * list of models.
 */
public class XGGClassModel extends XGGModel {
private Class<?> targetClass = null;
private boolean canBind = false;
private String modelvarname = "model";

private HashMap<String, Class<?>> fields = new
HashMap<String, Class<?>>();
private HashMap<String, Class<?>> methods =
new HashMap<String, Class<?>>();
private HashMap<String, PropertyDescriptor> props
= new HashMap<String, PropertyDescriptor>();

public XGGClassModel(XGGFormat aMain, Class<?>
aClass, boolean isList) {
  super(aMain);
  targetClass = aClass;
  if (!isList)
    type = MODEL_CLASS;
  else {
    type = MODEL_CLASSLIST;
    modelvarname = "curmodel";
  }

  Field fieldsarr[] = aClass.getDeclaredFields();

  for (int i = 0; i < fieldsarr.length; i++) {
    if (Modifier.isPublic(fieldsarr[i].getModifiers()))
      fields.put(fieldsarr[i].getName(),
fieldsarr[i].getType());
  }

  Method methsarr[] =
aClass.getDeclaredMethods();

  for (int i = 0; i < methsarr.length; i++) {
    if (Modifier.isPublic(methsarr[i].getModifiers())) {
      if (methsarr[i].getName() ==
"addPropertyChangeListener") {
        Class<?> params[] =
methsarr[i].getParameterTypes();
        if ((params.length == 1) && (params[0] ==
PropertyChangeListener.class))
          canBind = true;
      }
      if ((methsarr[i].getReturnType() != null)
        && (methsarr[i].getParameterTypes().length
== 0))
```

```
      methods.put(methsarr[i].getName(),
methsarr[i].getReturnType());
    }
  }

  BeanInfo biInfo =
sp.SPUtil.getBeanInfo(targetClass);

  PropertyDescriptor pd[] =
biInfo.getPropertyDescriptors();
  for (int i = 0; i < pd.length; i++) {
    if (pd[i].getReadMethod() != null)
      props.put(pd[i].getName(), pd[i]);
  }
}

public String getTargetType() {
  if (type == MODEL_CLASS)
    return
sp.SPUtil.getShortName(targetClass.getName());
  else
    return "AbstractList";
}

public String getVarName() {
  return modelvarname;
}

public String getName() {
  return
sp.SPUtil.getShortName(targetClass.getName());
}

public String getVarType(boolean is1_5) {
  if ((is1_5) && (type == MODEL_CLASSLIST))
    return "AbstractList<" + getName() + '>';
  return getTargetType();
}

public boolean isBindable() {
  return canBind;
}

public static String getSafeValue(String varname,
String str, Class<?> cls,
  boolean mode) {
  return "((" + varname + " != null) ? " + (mode ?
varname + '.' + str : str)
    + " : " + ValueConverter.getNullValue(cls) + ")";
}

public String getVarCode(String str, Class<?>
returncls)
  throws XGGSpecialAttributeException {
  Class<?> cls = (Class<?>) fields.get(str);

  if (cls != null) {
    if (!sp.SPUtil.isClassAssignable(returncls, cls)) {
      String tempstr = null;
      if (returncls == String.class)
        tempstr =
ValueConverter.tryToString(modelvarname + '.' +
str, cls);
      else if ((returncls == Object.class)
        || ((returncls == Number.class) && (cls !=
char.class)))
        tempstr =
ValueConverter.tryToObject(modelvarname + '.' +
str, cls);

      if (tempstr != null)
        return getSafeValue(modelvarname, tempstr,
returncls, false);

      throw new XGGSpecialAttributeException(25,
null, null);
    }

    return getSafeValue(modelvarname, str,
returncls, true);
  } else
    return null;
}

public String getVarMethod(String str, Class<?>
returncls)
  throws XGGSpecialAttributeException {
  Class<?> cls = (Class<?>) methods.get(str);

  if (cls != null) {
    if (!sp.SPUtil.isClassAssignable(returncls, cls)) {
      String tempstr = null;
      if (returncls == String.class)
        tempstr =
ValueConverter.tryToString(modelvarname + '.' +
str + "()",
          cls);
      else if ((returncls == Object.class)
        || ((returncls == Number.class) && (cls !=
char.class)))
        tempstr =
ValueConverter.tryToObject(modelvarname + '.' +
str + "()",
          cls);

      if (tempstr != null)
        return getSafeValue(modelvarname, tempstr,
returncls, false);

      throw new XGGSpecialAttributeException(25,
null, null);
    }

    return getSafeValue(modelvarname, str + "()",
returncls, true);
  } else
    return null;
}

public PropertyDescriptor getBindMethod(String str,
Class<?> returncls)
  throws XGGSpecialAttributeException {
  PropertyDescriptor pd = (PropertyDescriptor)
props.get(str);

  if (pd != null) {
    Class<?> cls = pd.getPropertyType();
    if (!sp.SPUtil.isClassAssignable(returncls, cls))
      throw new XGGSpecialAttributeException(25,
null, null);

    return pd;
  } else
    return null;
}

public Object[] getTreeChildren() {
  return new Object[] {
      new XGGFormat.BasicTreeLeaf("type: " +
mtype[type], "private_co.gif",
        this),
      new XGGFormat.BasicTreeLeaf("class: " +
targetClass.getName(),
        "private_co.gif", this) };
}
}

**********
```

FILE: sp/format/XGGDataModel.java

```
package sp.format;

/**
 * A Model from a database result. It uses the
<code>ResultSet</code> class.
 */
public class XGGDataModel extends XGGModel {
public XGGDataModel(XGGFormat aMain) {
  super(aMain);
  type = MODEL_DBASE;
}

public String getTargetType() {
  return "ResultSet";
}

public String getVarName() {
  return "model";
}

public String getData(String str, Class<?> returncls)
{
  if (returncls == int.class)
    return getVarName() + ".getInt(\"" + str + "\")";
  else if (returncls == byte.class)
    return getVarName() + ".getByte(\"" + str + "\")";
  else if (returncls == float.class)
    return getVarName() + ".getFloat(\"" + str +
"\")";
  else if (returncls == double.class)
    return getVarName() + ".getDouble(\"" + str +
"\")";
  else if (returncls == long.class)
    return getVarName() + ".getLong(\"" + str +
"\")";
  else if (returncls == short.class)
    return getVarName() + ".getShort(\"" + str +
"\")";
  else if (returncls == boolean.class)
    return getVarName() + ".getBoolean(\"" + str +
"\")";
  else if (returncls == char.class)
    return "(char)" + getVarName() + ".getInt(\"" +
str + "\")";
  else if ((returncls == String.class) || (returncls ==
CharSequence.class))
    return getVarName() + ".getString(\"" + str +
"\")";
  else if (returncls == Object.class)
    return getVarName() + ".getObject(\"" + str +
"\")";
  else if (returncls == java.io.InputStream.class)
    return getVarName() + ".getBinaryStream(\"" +
str + "\")";
  else if (returncls == java.sql.Array.class)
    return getVarName() + ".getArray(\"" + str +
"\")";
  else if (returncls == java.math.BigDecimal.class)
    return getVarName() + ".getBigDecimal(\"" + str
+ "\")";
  else if (returncls == java.sql.Blob.class)
    return getVarName() + ".getBlob(\"" + str + "\")";
  else if (returncls == byte[].class)
    return getVarName() + ".getBytes(\"" + str +
"\")";
  else if (returncls == java.io.Reader.class)
    return getVarName() + ".getCharacterStream(\""
+ str + "\")";
  else if (returncls == java.sql.Clob.class)
    return getVarName() + ".getClob(\"" + str + "\")";
  else if (returncls == java.sql.Date.class)
    return getVarName() + ".getDate(\"" + str +
"\")";
  else if (returncls == java.sql.Ref.class)
    return getVarName() + ".getRef(\"" + str + "\")";
  else if (returncls == java.sql.Time.class)
    return getVarName() + ".getTime(\"" + str +
"\")";
  else if (returncls == java.sql.Timestamp.class)
    return getVarName() + ".getTimestamp(\"" + str
+ "\")";
  else if (returncls == java.net.URL.class)
    return getVarName() + ".getURL(\"" + str + "\")";

  return null;
}
}
```

****

FILE: sp/format/XGGAction.java

```
package sp.format;

import sp.code.*;
import sp.format.code.*;
import java.lang.reflect.*;
import org.xml.sax.helpers.*;

/**
 * Represents an action object.
 */
public class XGGAction extends XGGTag
implements TreeType {
private static final String attrs[] = new String[]
{ "actionCommand",
    "default", "longDescription", "shortDescription",
"name", "accelerator",
    "mnemonicKey" };

private static final String attrconsts[] = new String[]
{ "ACTION_COMMAND_KEY",
    "DEFAULT", "LONG_DESCRIPTION",
"SHORT_DESCRIPTION", "NAME",
    "ACCELERATOR_KEY", "MNEMONIC_KEY" };

private static final Class<?> attrcls[] = new Class<?
>[] { String.class,
    String.class, String.class, String.class,
String.class, Integer.class,
    java.awt.event.KeyEvent.class };

private JQuickClass xggqc = null;
private String sName = null;

public XGGAction(XGGFormat aMain, AttributesImpl
anAtts) {
  super(anAtts);
  xggqc = new JQuickClass("AbstractAction");
  xggMain = aMain;

  sName = getAttributeValue("id");

  JMethod jm = new JMethod(Modifier.PUBLIC, null,
"actionPerformed",
    new JArg[] { new JArg("e", "ActionEvent") });
  jm.setBody(new
StringBuffer(getAttributeValue("actionPerformed")
+ "(e);"));
  xggqc.addMethod(jm);
}

private static Class<?> getAttrClass(String str) {
  if ((str.equals("id")) ||
(str.equals("actionPerformed")))
    return String.class;
```

```
  for (int i = 0; i < attrs.length; i++)
    if (str.equals(attrs[i]))
      return attrcls[i];

  return null;
}

public String getTagType() {
  return "Action";
}

public static String validate(AttributesImpl atts) {
  boolean hasid = false;
  boolean hasap = false;

  for (int i = 0; i < atts.getLength(); i++) {
    String str = atts.getQName(i);
    if (getAttrClass(str) == null)
      return "Invalid attribute '" + str + "'";

    if ((!hasid) && (str.equals("id")))
      hasid = true;
    if ((!hasap) && (str.equals("actionPerformed")))
      hasap = true;
  }

  if (!hasid)
    return "Requires attribute 'id'!";

  if (!hasap)
    return "Requires attribute 'actionPerformed'!";

  return null;
}

public JVar outputVar() {
  JVar result = new JVar(sName, "AbstractAction");
  result.setCreated(true);
  result.setQInitVal(xggqc);

  return result;
}

public String outputCode() throws XGGTagException
{
  StringBuffer result = new StringBuffer();

  for (int i = 0; i < attrs.length; i++) {
    String str = getAttributeValue(attrs[i]);

    if (str != null) {
      result.append(sName);
      result.append(".putValue(Action.");
      result.append(attrconsts[i]);
      result.append(", ");

      try {
        result.append(ValueConverter.convertValue(str
, attrcls[i],
          javax.swing.Action.class,
xggMain.getHeadElement()));
      } catch (sp.XGGException e) {
        throw new XGGTagException(this,
XGGConversionException.getNeededError(
          attrs[i], str,
sp.SPUtil.getShortName(attrcls[i].getName()), e));
      }
      result.append(");");
      result.append(sp.SPUtil.lineBreak());
    }
  }

  return result.toString();
}

public Object[] getTreeChildren() {
  return new Object[0];
}

public Object getTreeParent() {
  return null;
}

public String getIconName() {
  return "public_co.gif";
}

public String toString() {
  return "Action (" + getAttributeValue("id") + ')';
}
}

**********
FILE: sp/format/XGGAttribute.java

package sp.format;

import sp.format.code.*;
import java.beans.*;
import java.lang.reflect.*;

/**
 * Represents a single property of an element in an
XGG file, passed as an
 * XML tag attribute. It contains methods to validate
its content and display
 * its corresponding Java code.
 */
public class XGGAttribute {
private String name, value, crossrefName = null;
private FeatureDescriptor fdesc = null;
private XGGElement elem = null;
private boolean bCommentedOut = false;

public XGGAttribute(XGGElement anElem, String
aName, String aValue,
  FeatureDescriptor anFdesc) {
  name = aName;
  value = aValue;
  fdesc = anFdesc;
  elem = anElem;
}

public boolean isCommentedOut() {
  return bCommentedOut;
}

public String getName() {
  return name;
}

public String getCrossRefName() {
  return crossrefName;
}

public void setCrossRefName(String str) {
  crossrefName = str;
}

public FeatureDescriptor getDescriptor() {
  return fdesc;
}

public String convertAttrValue(Class<?> cls) throws
XGGValueException,
  XGGForwardRefException,
XGGSpecialAttributeException {
```

```
  try {
    String temp =
ValueConverter.convertValue(value, cls,
      elem.getTargetClass(), elem, this, true);
    if (temp != null)
      return temp;
  } catch (XGGConversionException e) {
    throw new XGGValueException(name, value,
cls.getName(), e.getError());
  }

  throw new XGGValueException(name, value,
cls.getName());
}

public String getCode() throws XGGValueException,
XGGForwardRefException,
  XGGSpecialAttributeException {
  StringBuffer result = new StringBuffer();

  if (fdesc instanceof PropertyDescriptor) {
    PropertyDescriptor pd = (PropertyDescriptor)
fdesc;
    Method mtd = pd.getWriteMethod();
    result.append(mtd.getName());
    result.append('(');
    result.append(convertAttrValue(pd.getPropertyTy
pe()));
    result.append(')');
  } else if (fdesc instanceof EventSetDescriptor) {
    bCommentedOut = true;
    result.append(name);
    result.append(" = ");
    result.append(value);
  } else {
    bCommentedOut = true;
    elem.getMain().giveWarning(
      "Encountered unknown attribute '" + name + "'
and was ignored.", elem);
  }

  result.append(';');
  result.append(sp.SPUtil.lineBreak());

  return result.toString();
}
}

**********
FILE: sp/format/XGGItem.java

package sp.format;

import java.util.*;
import org.xml.sax.helpers.*;

/**
 * Represents an item in an XGG file. Items are the
children tags of components
 * such as
<code>JComboBox</code>,<code>JTree</code>,
<code>JList</code> and
 * <code>JTable</code>.
 */
public class XGGItem extends XGGTag implements
TreeType {
private XGGElement elem = null;
private String sType = null;

private LinkedList<XGGItem> children = new
LinkedList<XGGItem>();
private XGGItem parent = null;

public XGGItem(String aType, XGGElement aParent,
AttributesImpl anAtts) {
  super(anAtts);
  sType = aType;
  elem = aParent;
  if (elem != null) {
    xggMain = elem.getMain();
    targetClass = elem.getTargetClass();
  }
}

public String getTagType() {
  return sType;
}

public String getErrorHeader() {
  return "Error in Item";
}

public void addItem(XGGItem item) {
  children.add(item);
  item.parent = this;
}

public Iterator<?> getChildren() {
  return children.iterator();
}

public String checkAttributes() {
  for (int i = 0; i < atts.getLength(); i++) {
    String str = atts.getQName(i);
    if (!str.equals("value"))
      return str;
  }
  return null;
}

public Object[] getTreeChildren() {
  return children.toArray(new
Object[children.size()]);
}

public Object getTreeParent() {
  if (parent != null)
    return parent;
  return elem;
}

public String toString() {
  return sType + ": " +
this.getAttributeValue("value");
}

public String getIconName() {
  return "intf_obj.gif";
}
}

**********
FILE: sp/format/XGGSpecialAttribute.java

package sp.format;

import java.util.*;

/**
 * Represents the special input in an attribute of an
XGG file. A special input
 * starts with <code>'{'</code> and ends with
<code>'}'</code>.<br>
 * The special input types:<br>
 * <ul>
 *  <li> {var varname} - points to the variable
named "varname". Note that
```

```
 *  varname can only point to variables created
using the 'id' attribute.
 *  <li> {late varname} - creates a variable named
"varname" that is originally
 *  set to a default value (usually null) that must be
filled out in the
 *  <code>initialize()</code> method.
 *  <li> {modelvar varname} - points to a variable
in the model with type class
 *  or list.
 *  <li> {modelmethod methodname} - points to a
method in the model with type
 *  class or list. Note that it only accepts methods
that return the required
 *  value and with no parameters.
 *  <li> {bind bindtype propertyname} - binds a
bean property of a model with
 *  type class or list to the attribute. <br>
 *  The bindtype can be any of the following:
 *    <ul>
 *      <li>to - when the model property changes,
the elementy property will
 *          follow.
 *      <li>from - when the element property
changes, the model property will
 *          follow.
 *      <li>both - when any one property changes,
the other property will
 *          follow.
 *    </ul>
 *  <li> {data columnname} - retrieves the data for
the columnname in the
 *  current row for the model with type database.
 * </ul>
 */
public class XGGSpecialAttribute {
private String body = null;
private String value = null;
private String type = null;
private int attype = 0;
private Class<?> targetClass = null;

public static final int SPEC_ATTR_VAR = 1,
SPEC_ATTR_LATE = 2,
  SPEC_ATTR_MODELF = 3, SPEC_ATTR_MODELM =
4, SPEC_ATTR_BIND = 5, SPEC_ATTR_BINDFROM =
6, SPEC_ATTR_BINDTO = 7, SPEC_ATTR_DBASE = 8;

public XGGSpecialAttribute(String aBody, Class<?>
aClass) {
  body = aBody;
  targetClass = aClass;
}

public String getType() {
  return type;
}

public String getValue() {
  return value;
}

public Class<?> getTargetClass() {
  return targetClass;
}

public int getAttributeType() {
  return attype;
}

private void checkValue() throws
XGGSpecialAttributeException {
  if (!sp.SPUtil.isLegalVar(value))
    throw new XGGSpecialAttributeException(5, type,
value);

  if (!sp.SPUtil.isUnreservedVar(value))
    throw new XGGSpecialAttributeException(7, type,
value);

}

public void parse() throws
XGGSpecialAttributeException {
  String temp;

  if (body.length() > 2)
    temp = body.substring(1, body.length() - 1);
  else
    temp = body;

  StringTokenizer token = new
StringTokenizer(temp);

  if (!token.hasMoreTokens())
    throw new XGGSpecialAttributeException(1, type,
value);

  try {
    type = token.nextToken();

    if (!token.hasMoreTokens())
      throw new XGGSpecialAttributeException(2,
type, value);

    if (type.equals("var")) {
      attype = SPEC_ATTR_VAR;
      value = token.nextToken();

      checkValue();
    } else if (type.equals("late")) {
      attype = SPEC_ATTR_LATE;
      value = token.nextToken();

      checkValue();
    } else if (type.equals("modelvar")) {
      attype = SPEC_ATTR_MODELF;
      value = token.nextToken();

      if (!sp.SPUtil.isLegalVar(value))
        throw new XGGSpecialAttributeException(5,
type, value);
    } else if (type.equals("modelmethod")) {
      attype = SPEC_ATTR_MODELM;
      value = token.nextToken();

      if (!sp.SPUtil.isLegalVar(value))
        throw new XGGSpecialAttributeException(5,
type, value);
    } else if (type.equals("bind")) {
      String secType = token.nextToken();
      if (!token.hasMoreTokens())
        throw new XGGSpecialAttributeException(2,
type, value);

      if (secType.equals("both"))
        attype = SPEC_ATTR_BIND;
      else if (secType.equals("from"))
        attype = SPEC_ATTR_BINDFROM;
      else if (secType.equals("to"))
        attype = SPEC_ATTR_BINDTO;
      else
        throw new XGGSpecialAttributeException(29,
type, value);

      value = token.nextToken();
```

```
      if (!sp.SPUtil.isLegalVar(value))
        throw new XGGSpecialAttributeException(5,
type, value);
    } else if (type.equals("data")) {
      attype = SPEC_ATTR_DBASE;
      value = token.nextToken();

      if (!sp.SPUtil.isLegalVar(value))
        throw new XGGSpecialAttributeException(5,
type, value);
    } else
      throw new XGGSpecialAttributeException(4,
type, value);

    if (token.hasMoreTokens())
      throw new XGGSpecialAttributeException(3,
type, value);

  } catch (XGGSpecialAttributeException e) {
    throw e;
  } catch (Exception e) {
    throw new XGGSpecialAttributeException(0, type,
value);
  }
}
}

*********************************
FILE: sp/format/XGGBind.java

package sp.format;

import sp.code.*;
import java.beans.*;
import java.lang.reflect.*;

/**
 * The class that provides validation and code
output for data binding
 */
public class XGGBind {
private XGGElement elem = null;
private XGGFormat xggMain = null;
private XGGClassModel model = null;
private PropertyDescriptor pd = null, targetPd =
null;
private int mode = 0;

public XGGBind(XGGElement anElem,
PropertyDescriptor aPd,
  PropertyDescriptor aTargetPd, int aMode) throws
XGGSpecialAttributeException {
  elem = anElem;
  pd = aPd;
  targetPd = aTargetPd;
  mode = aMode;
  if (elem != null) {
    xggMain = elem.getMain();
    model = (XGGClassModel) xggMain.getModel();
  }

  if (mode !=
XGGSpecialAttribute.SPEC_ATTR_BINDTO) {
    if (pd.getWriteMethod() == null)
      throw new XGGSpecialAttributeException(32,
null, null);
    if (targetPd.getReadMethod() == null)
      throw new XGGSpecialAttributeException(34,
null, null);
  }

  if (mode !=
XGGSpecialAttribute.SPEC_ATTR_BINDFROM) {
    if (!model.isBindable())
      throw new XGGSpecialAttributeException(33,
null, null);
  }

}

public boolean isTwoway() {
  return mode ==
XGGSpecialAttribute.SPEC_ATTR_BIND;
}

public boolean isWriteToComp() {
  return mode !=
XGGSpecialAttribute.SPEC_ATTR_BINDTO;
}

public boolean isGetFromComp() {
  return mode !=
XGGSpecialAttribute.SPEC_ATTR_BINDFROM;
}

public String debugWrite() {
  String strtype = "";
  switch (mode) {
  case XGGSpecialAttribute.SPEC_ATTR_BINDTO:
    strtype = "bind to";
    break;
  case XGGSpecialAttribute.SPEC_ATTR_BINDFROM:
    strtype = "bind from";
    break;
  default:
    strtype = "bind both";
  }

  return "target." + targetPd.getName() + ' ' +
strtype + " model."
    + pd.getName() + " in element " +
elem.getUsableName();
}

public StringBuffer writeBind() {
  StringBuffer result = new StringBuffer();
  XGGModel mdl = elem.getMain().getModel();

  xggMain.addImport("javax.swing.event");
  result.append(xggMain.addHashMap(elem));

  result.append("__HashMap.put(\"model\", ");
  result.append(mdl.getVarName());
  result.append(");");
  result.append(sp.SPUtil.lineBreak());

  if (isGetFromComp()) {
    if (elem.getMain().getOptions().isComments()) {
      result.append("//bind target.");
      result.append(targetPd.getName());
      result.append(" to model.");
      result.append(pd.getName());
      result.append(sp.SPUtil.lineBreak());
    }

    result.append("if (");
    result.append(mdl.getVarName());
    result.append(" != null) ");
    result.append(elem.getUsableName());
    result.append(".addChangeListener(");
    result.append(sp.SPUtil.lineBreak());

    JQuickClass jqc = new
JQuickClass("ChangeListener");
    JArg ppargs[] = new JArg[] { new JArg("e",
"ChangeEvent") };
```

```
    jqc.addVar(new JVar(0, "__comp",
elem.getClassName(), "("
      + elem.getClassName() + ")__HashMap.get(\""
+ elem.getUsableName()
      + "\")"));

    jqc.addVar(new JVar(0, "__model",
mdl.getName(), "(" + mdl.getName()
      + ")__HashMap.get(\"model\")"));

    JMethod meth = new JMethod(Modifier.PUBLIC,
null, "stateChanged", ppargs);
    StringBuffer subresult = new StringBuffer();

    subresult.append("__model.");
    subresult.append(pd.getWriteMethod().getName(
));
    subresult.append("(__comp.");
    subresult.append(targetPd.getReadMethod().get
Name());
    subresult.append("());");

    meth.setBody(subresult);
    jqc.addMethod(meth);

    result.append(xggMain.getOptions().indent(jqc.to
String()));
    result.append(sp.SPUtil.lineBreak());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
  }

  if (isWriteToComp()) {
    xggMain.addImport("java.beans");

    if (elem.getMain().getOptions().isComments()) {
      result.append("//bind target.");
      result.append(targetPd.getName());
      result.append(" from model.");
      result.append(pd.getName());
      result.append(sp.SPUtil.lineBreak());
    }

    result.append("if (");
    result.append(mdl.getVarName());
    result.append(" != null) ");
    result.append(mdl.getVarName());
    result.append(".addPropertyChangeListener(");
    result.append(sp.SPUtil.lineBreak());

    JQuickClass jqc = new
JQuickClass("PropertyChangeListener");
    JArg ppargs[] = new JArg[] { new JArg("e",
"PropertyChangeEvent") };

    jqc.addVar(new JVar(0, "__comp",
elem.getClassName(), "("
      + elem.getClassName() + ")__HashMap.get(\""
+ elem.getUsableName()
      + "\")"));

    jqc.addVar(new JVar(0, "__model",
mdl.getName(), "(" + mdl.getName()
      + ")__HashMap.get(\"model\")"));

    JMethod meth = new JMethod(Modifier.PUBLIC,
null, "propertyChange", ppargs);
    StringBuffer subresult = new StringBuffer();

    subresult.append("if (\"");
    subresult.append(pd.getName());
    subresult.append("\".equals(e.getPropertyName()
))");
    subresult.append(sp.SPUtil.lineBreak());
    subresult.append(xggMain.getOptions().indent("_
_comp."));
    subresult.append(targetPd.getWriteMethod().get
Name());
    subresult.append("(__model.");
    subresult.append(pd.getReadMethod().getName()
);
    subresult.append("());");

    meth.setBody(subresult);
    jqc.addMethod(meth);

    result.append(xggMain.getOptions().indent(jqc.to
String()));
    result.append(sp.SPUtil.lineBreak());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
  }
  return result;
}
}

**********
FILE: sp/format/ViewType.java

package sp.format;

/**
 * The interface used by View objects in an XGG file.
 */
public interface ViewType {
  public String getViewType();
}

**********
FILE: sp/format/XGGView.java

package sp.format;

import sp.code.*;
import sp.compiler.*;
import java.util.*;
import java.lang.reflect.*;

/**
 * Represents the custom view of an XGG element.
This class handles the code
 * output of the anonymous class created to
override the drawing routines of
 * a JComponent.
 */
public class XGGView implements ViewType,
TreeType {
private static StringBuffer initCache = null;
private static JMethod drawString = null;
private static JMethod mouseEvent = null;

private boolean hascomp = false, hasbord = false,
hascont = false,
  hasstrvert = false;

private XGGElement elem;

private LinkedList<JShape> comps = new
LinkedList<JShape>();
private LinkedList<JShape> bords = new
LinkedList<JShape>();
private LinkedList<JShape> conts = new
LinkedList<JShape>();

private HashMap<String, Integer> tempvarnames
= new HashMap<String, Integer>();

public XGGView(XGGElement anElem) {
```

```
  elem = anElem;
  elem.getMain().addImport("java.awt.geom");
  elem.getMain().addImport("java.awt.event");
  elem.setView(this);
}

public String addViewElem(JShape js, JShape
parentjs) {
  String errstr = js.validate();
  if (errstr != null)
    return errstr;

  String viewstr = js.getViewType();
  if (viewstr.equals("Component")) {
    comps.add(js);
    hascomp = true;
  } else if (viewstr.equals("Border")) {
    bords.add(js);
    hasbord = true;
  } else
    parentjs.addChild(js);

  String cntstr = js.getAttributeValue("contains");
  if ((cntstr != null) && (cntstr.equals("true"))) {
    hascont = true;
    conts.add(js);
  }
  return null;
}

public String getViewType() {
  return "UI";
}

public void initStrVert() {
  hasstrvert = true;
}

public JMethod makePaintMethod() throws
XGGTagException {
  if (!hascomp)
    return null;

  JMethod result = new
JMethod(Modifier.PROTECTED, null,
"paintComponent",
    new JArg[] { new JArg("g", "Graphics") });

  StringBuffer resultant = new StringBuffer();

  resultant.append(getInitCode());

  Iterator<?> i = comps.iterator();
  while (i.hasNext()) {
    JShape js = (JShape) i.next();
    StringBuffer jscode = js.getPaintCode();
    if (jscode != null) {
      resultant.append(jscode);
      resultant.append(sp.SPUtil.lineBreak());
    }
  }

  result.setBody(resultant);

  return result;
}

public JMethod makeContainsMethod() throws
XGGTagException {
  if ((!hascont) ||
(elem.getMain().getOptions().isCheckOnly()))
    return null;

  JMethod result = new JMethod(Modifier.PUBLIC,
"boolean", "contains",
    new JArg[] { new JArg("x", "int"), new JArg("y",
"int") });

  StringBuffer resultant = new StringBuffer();

  Iterator<?> i = conts.iterator();
  while (i.hasNext()) {
    JShape js = (JShape) i.next();
    StringBuffer jscode = js.getContainsCode();
    if (jscode != null) {
      resultant.append(jscode);
      resultant.append(sp.SPUtil.lineBreak());
    }
  }

  resultant.append("return false;");
  resultant.append(sp.SPUtil.lineBreak());

  result.setBody(resultant);

  return result;
}

public JMethod makeBorderMethod() throws
XGGTagException {
  if (!hasbord)
    return null;

  JMethod result = new
JMethod(Modifier.PROTECTED, null, "paintBorder",
    new JArg[] { new JArg("g", "Graphics") });

  StringBuffer resultant = new StringBuffer();

  resultant.append(getInitCode());

  Iterator<?> i = bords.iterator();
  while (i.hasNext()) {
    JShape js = (JShape) i.next();
    StringBuffer jscode = js.getPaintCode();
    if (jscode != null) {
      resultant.append(jscode);
      resultant.append(sp.SPUtil.lineBreak());
    }
  }

  result.setBody(resultant);

  return result;
}

public JMethod makeDrawStringMethod() {
  if (((!hasstrvert) ||
(elem.getMain().getOptions().isCheckOnly()))
    return null;

  return
initDrawStringMethod(elem.getMain().getOptions());
}

public JMethod makeMouseEventMethod() {
  if (elem.getMain().getOptions().isCheckOnly())
    return null;

  return
initMouseEventMethod(elem.getMain().getOptions()
);
}

public String getNewName(Class<?> cls) {
  String stype = sp.SPUtil.getSubClassName(cls,
"_");
```

```
  Integer i = (Integer) tempvarnames.get(stype);

  if (i == null)
    i = new Integer(1);

  tempvarnames.put(stype, new Integer(i.intValue()
+ 1));

  return "__" + stype + i;
}

public static JMethod
initDrawStringMethod(CompilerOpts cOpts) {
  if (drawString != null)
    return drawString;

  drawString = new JMethod(Modifier.PRIVATE, null,
"drawStringVertical",
    new JArg[] { new JArg("g", "Graphics"), new
JArg("str", "String"),
        new JArg("x", "float"), new JArg("y", "float") });

  StringBuffer sb = new StringBuffer();
  String lbrk = sp.SPUtil.lineBreak();

  sb.append("Graphics2D __g2d = (Graphics2D)g;");
  sb.append(lbrk);
  sb.append("FontMetrics __fm =
g.getFontMetrics(g.getFont());");
  sb.append(lbrk);
  sb.append("float yinc = 0;");
  sb.append(lbrk);
  sb.append("for (int i=0; i<str.length(); i++)");
  sb.append(cOpts.openBrace());
  sb.append(cOpts.indent("__g2d.drawString(str.cha
rAt(i)+\"\",x,y + yinc);"));
  sb.append(lbrk);
  sb.append(cOpts.indent("yinc +=
__fm.getHeight();"));
  sb.append(lbrk);
  sb.append('}');
  sb.append(lbrk);
  drawString.setBody(sb);

  return drawString;
}

public static JMethod
initMouseEventMethod(CompilerOpts cOpts) {
  if (mouseEvent != null)
    return mouseEvent;

  mouseEvent = new JMethod(Modifier.PRIVATE,
null, "addMouseOverDetect", null);

  StringBuffer sb = new StringBuffer();
  String lbrk = sp.SPUtil.lineBreak();

  sb.append("__setup = true;");
  sb.append(lbrk);

  JVar jv = new JVar(0, "miaev", "MouseAdapter",
null);
  JQuickClass jqc = new
JQuickClass("MouseAdapter");
  JArg jargs[] = new JArg[] { new JArg("e",
"MouseEvent") };
  StringBuffer sbTemp = new StringBuffer();

  sbTemp.append("isMouseOver = true;");
  sbTemp.append(lbrk);
  sbTemp.append("repaint();");

  JMethod jm = new JMethod(Modifier.PUBLIC, null,
"mouseEntered", jargs);
  jm.setBody(sbTemp);
  jqc.addMethod(jm);

  sbTemp = new StringBuffer();

  sbTemp.append("isMouseOver = false;");
  sbTemp.append(lbrk);
  sbTemp.append("repaint();");

  jm = new JMethod(Modifier.PUBLIC, null,
"mouseExited", jargs);
  jm.setBody(sbTemp);
  jqc.addMethod(jm);
  jv.setQInitVal(jqc);

  sb.append(jv);
  sb.append(lbrk);
  sb.append("addMouseListener(miaev);");
  mouseEvent.setBody(sb);

  return mouseEvent;
}

private static StringBuffer getInitCode() {
  if (initCache != null)
    return initCache;

  initCache = new StringBuffer();
  initCache.append("if (!__setup)
addMouseOverDetect();");
  initCache.append(sp.SPUtil.lineBreak());
  initCache.append("Graphics2D __g2d =
(Graphics2D)g;");
  initCache.append(sp.SPUtil.lineBreak());
  initCache.append("Paint __drawpnt =
getForeground();");
  initCache.append(sp.SPUtil.lineBreak());
  initCache.append("Paint __fillpnt =
getBackground();");
  initCache.append(sp.SPUtil.lineBreak());
  initCache.append("FontMetrics fontmetrics =
g.getFontMetrics(getFont());");
  initCache.append(sp.SPUtil.lineBreak());
  initCache.append(sp.SPUtil.lineBreak());
  return initCache;
}

public StringBuffer getAdditionalCode() {
  Class<?> cls = elem.getTargetClass();
  if
(sp.SPUtil.isClassAssignable(javax.swing.AbstractBu
tton.class, cls)) {
    StringBuffer result = new StringBuffer();

    result.append(elem.getUsableName());
    result.append(".setContentAreaFilled(false);");
    result.append(sp.SPUtil.lineBreak());

    return result;
  }
  return null;
}

public void addShapeCodes(JClassType jct) throws
XGGTagException {
  jct.addVar(new JVar(Modifier.PRIVATE, "__setup",
"boolean", "false"));
  jct.addVar(new JVar(Modifier.PROTECTED,
"isMouseOver", "boolean", "false"));

  JMethod tempmeth = makePaintMethod();
  if (tempmeth != null)
```

```
    jct.addMethod(tempmeth);

  tempmeth = makeContainsMethod();
  if (tempmeth != null)
    jct.addMethod(tempmeth);

  tempmeth = makeBorderMethod();
  if (tempmeth != null)
    jct.addMethod(tempmeth);

  tempmeth = makeDrawStringMethod();
  if (tempmeth != null)
    jct.addMethod(tempmeth);

  tempmeth = makeMouseEventMethod();
  if (tempmeth != null)
    jct.addMethod(tempmeth);
}

public String toString() {
  return "View";
}

public Object[] getTreeChildren() {
  LinkedList<Object> ll = new
LinkedList<Object>();

  Iterator<JShape> iter = comps.iterator();
  while (iter.hasNext())
    ll.add(iter.next());

  iter = bords.iterator();
  while (iter.hasNext())
    ll.add(iter.next());

  return ll.toArray(new Object[] { ll.size() });
}

public Object getTreeParent() {
  return elem;
}

public String getIconName() {
  return "shutdown-16x16.gif";
}
}

**********
FILE: sp/format/XGGValueException.java

package sp.format;

/**
 * Thrown when an error occured while processing a
value of an attribute
 */
public class XGGValueException extends
sp.XGGException {
static final long serialVersionUID = 'v' * 'a' * 'l' * 'u' *
'e';

private String type, name, value, exError = null;

public XGGValueException(String aName, String
aValue, String aType) {
  super();
  name = aName;
  value = aValue;
  type = aType;
}

public XGGValueException(String aName, String
aValue, String aType,
  String anExError) {
  this(aName, aValue, aType);
  exError = anExError;
}

public String getType() {
  return type;
}

public String getName() {
  return name;
}

public String getValue() {
  return value;
}

public String getExError() {
  return exError;
}

public String getError() {
  return "In attribute '" + name + "': value '" + value
+ "' must be of type "
    + type + "." + (exError != null ? ' ' + exError :
"");
}

public int getLineNumber() {
  return 0;
}

public int getColumnNumber() {
  return 0;
}
}

**********
FILE: sp/format/XGGForwardRefExeption.java

package sp.format;

import sp.*;

/**
 * Thrown to indicate that forward reference is
needed for a certain attribute
 * to have a correct java code output. This is used
for special attribute "var".
 * What happens is that the attribute code is not
outputted until the target
 * variable is processed.
 */
public class XGGForwardRefException extends
XGGException {
static final long serialVersionUID = 'f' * 'o' * 'r' * 'w' *
'a' * 'r' * 'd';

private XGGElement elem = null;
private XGGAttribute atts = null;

public XGGForwardRefException(XGGElement
anElem, XGGAttribute anAtts) {
  super();
  elem = anElem;
  atts = anAtts;
}

public XGGElement getElement() {
  return elem;
}

public XGGAttribute getAttribute() {
  return atts;
}
```

```
public int getLineNumber() {
  return 0;
}

public int getColumnNumber() {
  return 0;
}

public String getError() {
  return null;
}
}

**********
FILE: sp/format/XGGFormatException.java

package sp.format;

import sp.*;

/**
 * Thrown when an error occured while processing
the code output. This is used
 * for the general errors and is normaly wrapped by
other exceptions.
 */
public class XGGFormatException extends
XGGException {
static final long serialVersionUID = 'f' * 'o' * 'r' * 'm'
* 'a' * 't';

private String error = null;

public XGGFormatException(String anError) {
  super();
  error = anError;
}

public String getErrorMessage() {
  return error;
}

public String getError() {
  return "Error in format: " + error;
}

public int getLineNumber() {
  return 0;
}

public int getColumnNumber() {
  return 0;
}
}

**********
FILE: sp/format/XGGElementException.java

package sp.format;

import sp.*;

/**
 * Thrown when an error occured while processing
an XGG element
 */
public class XGGElementException extends
XGGException {
static final long serialVersionUID = 'e' * 'l' * 'e' * 'm';

private XGGElement elem;
private XGGException err;

public XGGElementException(XGGElement AnElem,
XGGException anExc) {
  super();
  elem = AnElem;
  err = anExc;
}

public XGGElement getElement() {
  return elem;
}

public XGGException getException() {
  return err;
}

public int getLineNumber() {
  if (err != null) {
    int temp = err.getLineNumber();
    if (temp > 0)
      return temp;
  }
  return elem.getLineNumber();
}

public int getColumnNumber() {
  if (err != null) {
    int temp = err.getColumnNumber();
    if (temp > 0)
      return temp;
  }
  return elem.getColumnNumber();
}

public String getError() {
  if (err != null)
    return err.getError();
  return null;
}
}

**********
FILE: sp/format/XGGSpecialAttributeException.java

package sp.format;

import sp.*;

/**
 * Thrown when an error occured while processing a
special attribute input
 */
public class XGGSpecialAttributeException extends
XGGException {
static final long serialVersionUID = 's' * 'p' * 'e' * 'c'
* 'a' * 't' * 't'
  * 'r';

private String value = null;
private String type = null;
private int errtype = 0;
private XGGAttribute xggatt = null;
private Class<?> cls = null;

public XGGSpecialAttributeException(int anErrtype,
String aType, String aValue) {
  super();
  type = aType;
  value = aValue;
  errtype = anErrtype;
}

public XGGSpecialAttributeException(int anErrtype,
XGGSpecialAttribute xggs) {
  super();
```

```
  type = xggs.getType();
  value = xggs.getValue();
  cls = xggs.getTargetClass();
  errtype = anErrtype;
}

public void setAttribute(XGGAttribute atts) {
  xggatt = atts;
}

public int getErrorType() {
  return errtype;
}

public String getError() {
  String str = "";
  if (xggatt != null)
    str = "In attribute '" + xggatt.getName() + "': ";
  switch (errtype) {
  case 1:
    return str + "No arguments!";
  case 2:
    return str + "Incomplete arguments for " + type
+ "!";
  case 3:
    return str + "Too many arguments for " + type +
" (" + value + ").";
  case 4:
    return str + "Invalid special argument type '" +
type + "'.";
  case 5:
    return str + "Invalid special argument value '" +
value + "'.";
  case 6:
    return str + "Cross reference not allowed!";
  case 7:
    return str + "Value '" + value + "' is reserved.
Use another one.";
  case 9:
    return str + "Cannot cross reference variables
not in the same loop!";
  case 10:
    return str + "Id '" + value + "' does not exist!";
  case 11:
    return str + "Id '" + value + "' is incompatible
with attribute type!"
      + ((cls != null) ? '(' + cls.getName() + ')' : "");
  case 20:
    return str + "Cannot use '" + type + "'. No
assigned model!";
  case 21:
    return str + "Cannot use '" + type + "'. Model is a
database result!";
  case 22:
    return str + "Cannot use '" + type + "'. Model is a
java class!";
  case 23:
    return str + type + " '" + value
      + "' does not exist! modelvar should point "
      + "to a public variable/field";
  case 24:
    return str + type + " '" + value + "' does not
exist! modelmethod should "
      + "point to a non-void method with no
parameters.";
  case 25:
    return str + type + " '" + value + "' is
incompatible with attribute type!"
      + ((cls != null) ? '(' + cls.getName() + ')' : "");
  case 28:
    return str + "Cannot use model of type 'list' or
'database' outside "
      + "of model loop!";
  case 29:
    return str + "Invalid bind type. (both, from, to)
accepted. ";
  case 30:
    return str + "Property '" + value + "' of the model
does not exist!";
  case 31:
    return str + "The attribute cannot be binded!";
  case 32:
    return str + "The model property cannot be
written! (no get method)";
  case 33:
    return str + "The model does not support 'bind
to' and 'bind both'!"
      + "(model does not have the
addPropertyChangeListener("
      + "PropertyChangeListener) method)";
  case 34:
    return str + "The attribute value cannot be
retrieved! (no get method)";
  case 40:
    return str + "Cannot use '" + type + "'. Model is a
class!";
  case 41:
    return str + "'" + type + "' cannot be used on
attribute's required class!"
      + ((cls != null) ? '(' + cls.getName() + ')' : "");
  default:
    return str + "Unknown error in special
argument.";
  }
}

public int getLineNumber() {
  return 0;
}

public int getColumnNumber() {
  return 0;
}
}

**********
FILE: sp/format/XGGTagException.java

package sp.format;

import sp.*;

/**
 * Thrown when an error occured while processing a
non-element tag
 * (item, view, actions etc.)
 */
public class XGGTagException extends
XGGException {
static final long serialVersionUID = 't' * 'a' * 'g';
XGGTag tag = null;
XGGException err = null;

public XGGTagException(XGGTag aTag,
XGGException anErr) {
  tag = aTag;
  err = anErr;
}

public String getError() {
  return tag.getErrorHeader() + " '" +
tag.getTagType() + "' at "
    + tag.getLineNumber() + ',' +
tag.getColumnNumber() + ' '
    + ((err != null) ? ": " + err.getError() : "");
}

public int getLineNumber() {
```

```
  return tag.getLineNumber();
}

public int getColumnNumber() {
  return tag.getColumnNumber();
}
}

**********
FILE: sp/format/code/CodeHandler.java

package sp.format.code;

import sp.*;
import sp.code.*;
import sp.format.*;
import java.util.*;
import java.lang.reflect.*;

/**
 * The base class for handling how an element
should output its code. This
 * does not include the handling for defautl
properties as the <code>
 *  XGGAttribute</code> class handles this.
 */
public abstract class CodeHandler {
private static boolean isValidClass(Class<?> pcls,
Class<?> cls) {
  return sp.SPUtil.isClassAssignable(pcls, cls);
}

public static CodeHandler getHandler(Class<?> cls,
XGGElement anElem) {
  if ((isValidClass(javax.swing.JPanel.class, cls))
    || (isValidClass(javax.swing.JDesktopPane.class,
cls))
    || (isValidClass(javax.swing.JToolBar.class, cls)))
    return new PanelCodeHandler(cls, anElem);
  else if ((isValidClass(javax.swing.JFrame.class, cls))
    || (isValidClass(javax.swing.JApplet.class, cls))
    || (isValidClass(javax.swing.JInternalFrame.class,
cls)))
    return new FrameCodeHandler(cls, anElem);
  else if (isValidClass(javax.swing.JComboBox.class,
cls))
    return new ComboBoxCodeHandler(cls, anElem);
  else if (isValidClass(javax.swing.JList.class, cls))
    return new ListCodeHandler(cls, anElem);
  else if (isValidClass(javax.swing.JTree.class, cls))
    return new TreeCodeHandler(cls, anElem);
  else if (isValidClass(javax.swing.JTable.class, cls))
    return new TableCodeHandler(cls, anElem);
  else if (isValidClass(javax.swing.Box.class, cls))
    return new BoxCodeHandler(cls, anElem);
  else if (isValidClass(javax.swing.JSplitPane.class,
cls))
    return new SplitPaneCodeHandler(cls, anElem);
  else if
(isValidClass(javax.swing.JTabbedPane.class, cls))
    return new TabbedPaneCodeHandler(cls,
anElem);
  else if (isValidClass(javax.swing.JScrollPane.class,
cls))
    return new ScrollPaneCodeHandler(cls, anElem);
  else if
((isValidClass(javax.swing.JRadioButton.class, cls))
    ||
(isValidClass(javax.swing.JRadioButtonMenuItem.cla
ss, cls)))
    return new RadioButtonCodeHandler(cls,
anElem);
  else if (isValidClass(javax.swing.JMenuBar.class,
cls))
    return new MenuBarCodeHandler(cls, anElem);
  else if ((isValidClass(javax.swing.JMenu.class, cls))
    || (isValidClass(javax.swing.JPopupMenu.class,
cls)))
    return new MenuCodeHandler(cls, anElem);
  else if (isValidClass(javax.swing.JSeparator.class,
cls))
    return new SeparatorCodeHandler(cls, anElem);

  return new DefaultCodeHandler(cls, anElem);
}

protected XGGElement elem = null;
protected Class<?> cls = null;

private static HashMap<Class<?>, Class<?>>
ModelDataCache = new HashMap<Class<?>,
Class<?>>();

protected CodeHandler(Class<?> aClass,
XGGElement anElem) {
  cls = aClass;
  elem = anElem;
}

public boolean isSpecialAttr(String str) {
  return getClassAttr(str) != null;
}

public Class<?> getClassAttr(String str) {
  if (str.equals("constraint"))
    return int.class;
  else if ((str.indexOf("constraint.") == 0) &&
(str.length() > 10)) {
    String gbstr = str.substring(11);

    if ((gbstr.equals("anchor")) || (gbstr.equals("fill"))
      || (gbstr.equals("gridheight")) ||
(gbstr.equals("gridwidth"))
      || (gbstr.equals("gridx")) ||
(gbstr.equals("gridy"))
      || (gbstr.equals("insets")) ||
(gbstr.equals("ipadx"))
      || (gbstr.equals("ipady")) ||
(gbstr.equals("insets")))
      return int.class;
    else if ((gbstr.equals("weightx")) ||
(gbstr.equals("weighty")))
      return double.class;
    else if (gbstr.equals("insets"))
      return java.awt.Insets.class;
  }

  return null;
}

public static Class<?> getModelClass(XGGElement
elem) {
  Class<?> pclass = elem.getTargetClass();
  Field mdlfld = null;
  Class<?> result = null;

  if (ModelDataCache.containsKey(pclass)) {
    result = (Class<?>) ModelDataCache.get(pclass);
    return result;
  }

  try {
    mdlfld = pclass.getField("MODELCLASS");

    if (mdlfld != null) {
      if ((mdlfld.getType() != Class.class)
        || (mdlfld.getModifiers() != (Modifier.PUBLIC |
Modifier.STATIC | Modifier.FINAL)))
```

```
        mdlfld = null;
      }
    } catch (NoSuchFieldException e) {} catch
(SecurityException e) {}

    if (mdlfld == null) {
      ModelDataCache.put(pclass, null);
      return null;
    } else {
      try {
        result = (Class<?>) mdlfld.get(null);
      } catch (Exception e) {
        elem.getMain().giveWarning(
          "Cannot retrieve MODELCLASS data in the" + "
object.", elem);
      }
    }

    ModelDataCache.put(pclass, result);
    return result;
  }

  public StringBuffer commentCode() {
    if (elem.getMain().getOptions().isComments()) {
      StringBuffer result = new StringBuffer();

      result.append("//");
      result.append(elem.getClassName());
      result.append(" '");
      result.append(elem.getUsableName());
      result.append("' with start tag ending at (");
      result.append(elem.getLineNumber());
      result.append(", ");
      result.append(elem.getColumnNumber());
      result.append(')');
      result.append(sp.SPUtil.lineBreak());

      return result;
    }
    return null;
  }

  public StringBuffer openCode(String initstr, boolean
tocreate)
    throws XGGElementException {
    if (!elem.isCreated()) {
      StringBuffer result = new StringBuffer();

      JVar jv = new JVar(initstr,
sp.SPUtil.getShortName(cls.getName()));
      jv.setCreated(tocreate);
      jv.setModifier(0);

      JQuickClass jqc = new
JQuickClass(sp.SPUtil.getShortName(cls.getName()))
;

      Class<?> mdlclass = getModelClass(elem);
      String modelstr =
elem.getAttributeValue("model.data");

      if (mdlclass == null) {
        if (modelstr != null)
          throw new XGGElementException(elem, new
XGGFormatException(
            "Element cannot have the model.data "
              + "attribute. Only XGG generated objects
with models can have it."));
      } else {
        if (modelstr == null)
          throw new XGGElementException(elem, new
XGGFormatException(
            "Element needs the model.data "
              + "attribute. It is the model needed by the
object."));

        try {
          modelstr =
ValueConverter.convertValue(modelstr, mdlclass,
mdlclass,
            elem);
        } catch (XGGException e) {
          throw new XGGElementException(elem,
XGGConversionException
            .getNeededError("model.data", modelstr,
mdlclass.getName(), e));
        }
      }

      if (modelstr != null)
        jqc.setPassValues(modelstr);

      XGGView view = elem.getView();

      if (view == null)
        jqc.setHasBody(false);
      else {
        try {
          view.addShapeCodes(jqc);
        } catch (XGGException e) {
          throw new XGGElementException(elem, e);
        }
      }

      jv.setQInitVal(jqc);

      result.append(jv.toString());
      result.append(sp.SPUtil.lineBreak());

      elem.setCreated(true);

      if (view != null) {
        StringBuffer sbTemp =
view.getAdditionalCode();
        if (sbTemp != null)
          result.append(sbTemp);
      }

      return result;
    }

    return null;
  }

  public StringBuffer attrCode(String qName, String
qVal)
    throws XGGElementException {
    return null;
  }

  public StringBuffer lateCode() throws
XGGElementException {
    return null;
  }

  public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
    throws XGGElementException {
    return null;
  }

  public StringBuffer itemCode(XGGItem item) throws
XGGElementException {
    return null;
  }
}
```

```
*********************************
FILE: sp/ format/code/DefaultCodeHandler.java

package sp.format.code;

import sp.code.*;
import sp.format.*;
import java.lang.reflect.*;

/**
 * The code handler assigned to all elements
without a special code handler
 * assigned to them.
 */
public class DefaultCodeHandler extends
CodeHandler {
private boolean haspopup = false;

protected DefaultCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

public Class<?> getClassAttr(String str) {
  if (str.equals("model.loop"))
    return boolean.class;
  else if (str.equals("hasmain"))
    return boolean.class;
  else if (str.equals("model.data"))
    return Object.class;
  else if (str.equals("cursor"))
    return java.awt.Cursor.class;
  else if ((str.indexOf("tab.") == 0) && (str.length()
> 3)) {
    String gbstr = str.substring(4);

    if ((gbstr.equals("title")) || (gbstr.equals("tip")))
      return String.class;
    else if (gbstr.equals("icon"))
      return javax.swing.Icon.class;
  }

  return super.getClassAttr(str);
}

public StringBuffer attrCode(String qName, String
qVal)
  throws XGGElementException {
  if (qName.equals("hasmain")) {
    if (elem.getParent() != null)
      throw new XGGElementException(elem, new
XGGFormatException(
          "Only the Head element can have the
'hasmain' attribute."));
    else if (elem.getTargetClass() ==
javax.swing.JApplet.class)
      throw new XGGElementException(elem, new
XGGFormatException(
          "JApplets cannot have the 'hasmain'
attribute."));
    else {
      try {
        String mainstr =
ValueConverter.convertValue(qVal, boolean.class,
elem
            .getTargetClass(), elem);

        if (mainstr.equals("false"))
          elem.getMain().setDoMain(false);
      } catch (sp.XGGException e) {
        throw new XGGElementException(elem,
XGGConversionException
            .getNeededError("hasmain", qVal, "boolean",
e));
      }
    }
  } else if (qName.equals("cursor")) {
    StringBuffer result = new StringBuffer();

    if (!elem.isHead()) {
      result.append(elem.getUsableName());
      result.append('.');
    }
    result.append("setCursor(");
    try {
      result.append(ValueConverter.convertValue(qVa
l, java.awt.Cursor.class,
          java.awt.Cursor.class, elem));
    } catch (sp.XGGException e) {
      throw new XGGElementException(elem,
XGGConversionException
          .getNeededError("cursor", qVal,
java.awt.Cursor.class.getName(), e));
    }
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    return result;
  }

  return null;
}

public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
  throws XGGElementException {
  Class<?> ccls = celem.getTargetClass();

  if
(sp.SPUtil.isClassAssignable(javax.swing.JPopupMen
u.class, ccls)) {
    if (haspopup)
      throw new XGGElementException(elem, new
XGGFormatException(
          "Component can only have 1 JPopupMenu."));

    StringBuffer result = new StringBuffer();
    sp.compiler.CompilerOpts cOpts =
elem.getMain().getOptions();

    if (cOpts.isMode1_5()) {
      if (!elem.isHead()) {
        result.append(elem.getUsableName());
        result.append('.');
      }
      result.append("setComponentPopupMenu(");
      result.append(celem.getUsableName());
      result.append(");");
      result.append(sp.SPUtil.lineBreak());
      return result;
    }

    result.append(elem.getMain().addHashMap(cele
m));

    if (!elem.isHead()) {
      result.append(elem.getUsableName());
      result.append('.');
    }
    result.append("addMouseListener(");
    result.append(sp.SPUtil.lineBreak());

    elem.getMain().addImport("java.awt.event");
    JQuickClass jqc = new
JQuickClass("MouseAdapter");
    JArg ppargs[] = new JArg[] { new JArg("e",
"MouseEvent") };
```

```
    jqc.addVar(new JVar(0, "__popup", "JPopupMenu",
      "(JPopupMenu)__HashMap.get(\"" +
celem.getUsableName() + "\")"));

    JMethod meth = new JMethod(Modifier.PUBLIC,
null, "mousePressed", ppargs);
    meth.setBody(new
StringBuffer("maybeShowPopup(e);"));
    jqc.addMethod(meth);

    meth = new JMethod(Modifier.PUBLIC, null,
"mouseReleased", ppargs);
    meth.setBody(new
StringBuffer("maybeShowPopup(e);"));
    jqc.addMethod(meth);

    meth = new JMethod(Modifier.PRIVATE, null,
"maybeShowPopup", ppargs);
    StringBuffer sbPop = new StringBuffer();
    sbPop.append("if (e.isPopupTrigger())");
    sbPop.append(sp.SPUtil.lineBreak());
    sbPop.append(cOpts
      .indent("__popup.show(e.getComponent(),e.get
X(),e.getY());"));
    sbPop.append(sp.SPUtil.lineBreak());

    meth.setBody(sbPop);
    jqc.addMethod(meth);

    result.append(cOpts.indent(jqc.toString()));
    result.append(sp.SPUtil.lineBreak());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
    haspopup = true;
    return result;
  }

  return null;
}
}
```

****

FILE: sp/ format/code/SplitPaneCodeHandler.java

```
package sp.format.code;

import sp.format.*;

/**
 * The code handler assigned to a JSplitPane, allows
up to 2 children and places
 * them to the left/up and right/down side in order.
 */
public class SplitPaneCodeHandler extends
DefaultCodeHandler {
int children = 0;

protected SplitPaneCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

public Class<?> getClassAttr(String str) {
  return super.getClassAttr(str);
}

private final boolean isHorizontal(String str) {
  return ((str == null) ||
(str.endsWith("HORIZONTAL_SPLIT")) || (str
    .equals("1")));
}

public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
  throws XGGElementException {
  if (children == 2)
    throw new XGGElementException(elem, new
XGGFormatException(
      "JSplitPane can only have 2 children."));

  String orstr =
elem.getAttributeValue("orientation");

  StringBuffer result = new StringBuffer();

  Class<?> ccls = celem.getTargetClass();

  children++;
  if
(sp.SPUtil.isClassAssignable(javax.swing.JComponen
t.class, ccls)) {
    result.append(elem.getUsableName());
    result.append('.');

    if (children == 1) {
      if (isHorizontal(orstr))
        result.append("setLeftComponent(");
      else
        result.append("setTopComponent(");
    } else {
      if (isHorizontal(orstr))
        result.append("setRightComponent(");
      else
        result.append("setBottomComponent(");
    }

    result.append(celem.getUsableName());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
    return result;
  }
  return null;
}
}
```

****

FILE: sp/ format/code/TabbedPaneCodeHandler.java

```
package sp.format.code;

import sp.format.*;

/**
 * The code handler assigned to a
TabbedPaneCodeHandler. Calls the addTab method
 * and can use custom attributes like tab.title and
tab.item
 */
public class TabbedPaneCodeHandler extends
DefaultCodeHandler {
protected TabbedPaneCodeHandler(Class<?>
aClass, XGGElement anElem) {
  super(aClass, anElem);
}

public Class<?> getClassAttr(String str) {
  return super.getClassAttr(str);
}

private String getTabAttr(XGGElement celem, String
str, Class<?> targetcls)
  throws XGGElementException {
  String temp = null;
  temp = celem.getAttributeValue("tab." + str);
  if (temp != null) {
    try {
      return ValueConverter.convertValue(temp,
targetcls,
```

```
        javax.swing.JSplitPane.class, celem);
    } catch (sp.XGGException e) {
      throw new XGGElementException(celem,
XGGConversionException
        .getNeededError("tab." + str, temp,
targetcls.getName(), e));
    }
  } else
    return "null";
}

public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
  throws XGGElementException {
  StringBuffer result = new StringBuffer();

  Class<?> ccls = celem.getTargetClass();

  if
(sp.SPUtil.isClassAssignable(javax.swing.JComponen
t.class, ccls)) {
    result.append(elem.getUsableName());
    result.append('.');

    result.append("addTab(");

    result.append(getTabAttr(celem, "title",
String.class));
    result.append(", ");

    result.append(getTabAttr(celem, "icon",
javax.swing.Icon.class));
    result.append(", ");

    result.append(celem.getUsableName());
    result.append(", ");

    result.append(getTabAttr(celem, "tip",
String.class));
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
    return result;
  }
  return null;
}
}

**********
FILE: sp/ format/code/BoxCodeHandler.java

package sp.format.code;

import sp.code.*;
import sp.format.*;

/**
 * The code handler assigned to a Box. Adds support
to how its constructor is
 * called.
 */
public class BoxCodeHandler extends
PanelCodeHandler {
protected BoxCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

public Class<?> getClassAttr(String str) {
  if (str.equals("axis"))
    return int.class;
  return super.getClassAttr(str);
}

public static String getBoxOrientation(XGGElement
elem)
  throws XGGElementException {
  String result = "BoxLayout.X_AXIS";
  String temp = elem.getAttributeValue("axis");
  if (temp != null) {
    try {
      result = ValueConverter.convertValue(temp,
int.class,
        javax.swing.BoxLayout.class, elem);
    } catch (sp.XGGException e) {
      throw new XGGElementException(elem,
XGGConversionException
        .getNeededError("axis", temp, "int", e));
    }
  }

  return result;
}

public StringBuffer openCode(String initstr, boolean
tocreate)
  throws XGGElementException {
  if (getModelClass(elem) != null)
    return super.openCode(initstr, tocreate);

  if (!elem.isCreated()) {
    StringBuffer result = new StringBuffer();

    JVar jv = new JVar(initstr,
sp.SPUtil.getShortName(cls.getName()));
    jv.setCreated(tocreate);
    jv.setModifier(0);

    StringBuffer initval = new StringBuffer();

    initval.append("new ");
    initval.append(sp.SPUtil.getShortName(cls.getNa
me()));
    initval.append('(');
    initval.append(getBoxOrientation(elem));
    initval.append(')');

    jv.setInitVal(initval.toString());
    result.append(jv.toString());
    result.append(sp.SPUtil.lineBreak());

    elem.setCreated(true);

    return result;
  }

  return null;
}
}

**********
FILE: sp/ format/code/ScrollPaneCodeHandler.java

package sp.format.code;

import sp.format.*;

/**
 * The code handler assigned to a JScrollPane. Can
accept up to 1 child.
 */
public class ScrollPaneCodeHandler extends
DefaultCodeHandler {
int children = 0;

protected ScrollPaneCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
```

```
}

public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
  throws XGGElementException {
  if (children == 1)
    throw new XGGElementException(elem, new
XGGFormatException(
      "JScrollPane can only have 1 child."));

  StringBuffer result = new StringBuffer();

  Class<?> ccls = celem.getTargetClass();

  children++;
  if
(sp.SPUtil.isClassAssignable(javax.swing.JComponen
t.class, ccls)) {
    result.append(elem.getUsableName());
    result.append(".setViewportView(");
    result.append(celem.getUsableName());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
    return result;
  }
  return null;
}
}

**********
FILE: sp/ format/code/PanelCodeHandler.java

package sp.format.code;

import sp.format.*;
import sp.compiler.*;

/**
 * The code handler assigned to a JPanel and some
Container withut a special
 * code handler (like JDesktopPane). Can contain
children.
 */
public class PanelCodeHandler extends
DefaultCodeHandler {
private static String gridbags[] = new String[]
{ "gridx", "gridy",
    "gridwidth", "gridheight", "weightx", "weighty",
"anchor", "fill",
    "insets", "ipadx", "ipady" };
private static Class<?> gridbagcls[] = new Class<?
>[] { int.class, int.class,
    int.class, int.class, double.class, double.class,
int.class, int.class,
    java.awt.Insets.class, int.class, int.class };

private String layoutvar = null;

protected PanelCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

public Class<?> getClassAttr(String str) {
  if (str.equals("layout"))
    return java.awt.LayoutManager.class;
  else if ((str.indexOf("gridbag.") == 0) &&
(str.length() > 7)) {
    String gbstr = str.substring(8);
    if ((gbstr.equals("columnWeights")) ||
(gbstr.equals("rowWeights")))
      return double[].class;
    else if ((gbstr.equals("columnWidths")) ||
(gbstr.equals("rowHeights")))
      return int[].class;
  }
  return super.getClassAttr(str);
}

public StringBuffer attrCode(String qName, String
qVal)
  throws XGGElementException {
  if (qName.equals("layout")) {
    StringBuffer result = new StringBuffer();
    String layoutstr = null;

    try {
      layoutstr = ValueConverter.convertValue(qVal,
        java.awt.LayoutManager.class,
elem.getTargetClass(), elem);
    } catch (sp.XGGException e) {
      throw new XGGElementException(elem,
XGGConversionException
        .getNeededError("layout", qVal,
java.awt.LayoutManager.class.getName(),
          e));
    }

    if (layoutstr.startsWith("new GridBagLayout")) {
      layoutvar =
elem.getMain().getNewName(java.awt.GridBagLayo
ut.class);
      result.append("GridBagLayout ");
      result.append(layoutvar);
      result.append(" = ");
      result.append(layoutstr);
      result.append(";");
      result.append(sp.SPUtil.lineBreak());

      getGridBagAttr(result, "rowHeights", int[].class);
      getGridBagAttr(result, "columnWidths",
int[].class);
      getGridBagAttr(result, "rowWeights",
double[].class);
      getGridBagAttr(result, "columnWeights",
double[].class);
    }

    if (!elem.isHead()) {
      result.append(elem.getUsableName());
      result.append('.');
    }
    result.append("setLayout(");
    if (layoutvar == null)
      result.append(layoutstr);
    else
      result.append(layoutvar);
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    return result;
  }
  return super.attrCode(qName, qVal);
}

public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
  throws XGGElementException {
  Class<?> ccls = celem.getTargetClass();

  if
(sp.SPUtil.isClassAssignable(javax.swing.JPopupMen
u.class, ccls))
    return childCode(celem, pelem);

  if
(sp.SPUtil.isClassAssignable(javax.swing.JComponen
t.class, ccls)) {
```

```
    StringBuffer result = new StringBuffer();

    if
(sp.SPUtil.isClassAssignable(javax.swing.JMenuItem.
class, ccls)) {
      elem.getMain().giveWarning(
        "Adding a JMenuItem/JMenu in a this " +
"container is not recommended",
        elem);
    } else if
(sp.SPUtil.isClassAssignable(javax.swing.JInternalFra
me.class,
      ccls)) {
      if (!
sp.SPUtil.isClassAssignable(javax.swing.JDesktopPa
ne.class, elem
        .getTargetClass()))
        elem.getMain().giveWarning(
          "Adding a JInternalFrame in a this "
            + " container is not recommended", elem);
    } else if
(sp.SPUtil.isClassAssignable(javax.swing.JDesktopPa
ne.class, elem
      .getTargetClass())) {
      elem.getMain().giveWarning(
        "Adding a component other than "
          + "JInternalFrame in this container is not
recommended", elem);
    }

    if ((layoutvar != null) &&
(layoutvar.startsWith("__GridBag"))) {
      CompilerOpts cOpts =
elem.getMain().getOptions();

      String gbconstvar = layoutvar + "_C";

      result.append('{');
      result.append(sp.SPUtil.lineBreak());

      result.append(cOpts.indent(gridbagcode(gbcons
tvar, celem)));

      result.append(sp.SPUtil.lineBreak());
      result.append("}");
      result.append(sp.SPUtil.lineBreak());

    }

    if (!elem.isHead()) {
      result.append(elem.getUsableName());
      result.append('.');
    }
    result.append("add(");
    result.append(celem.getUsableName());
    if (celem.isAttributeExist("constraint")) {
      result.append(", ");
      result.append(celem.getAttributeValue("constrai
nt"));
    }
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
    return result;
  }

  return null;
}

private void getGridBagAttr(StringBuffer result,
String attr, Class<?> targetcls)
  throws XGGElementException {
  String str = elem.getAttributeValue("gridbag." +
attr);
  if (str != null) {
    try {
      result.append(layoutvar);
      result.append('.');
      result.append(attr);
      result.append(" = ");
      result.append(ValueConverter.convertValue(str,
targetcls,
        java.awt.GridBagLayout.class, elem));
      result.append(';');
      result.append(sp.SPUtil.lineBreak());
    } catch (sp.XGGException e) {
      throw new XGGElementException(elem,
XGGConversionException
        .getNeededError("gridbag." + attr, str,
targetcls.getName(), e));
    }
  }
}

private String gridbagcode(String gbcname,
XGGElement celem)
  throws XGGElementException {
  StringBuffer result = new StringBuffer();

  result.append("GridBagConstraints ");
  result.append(gbcname);
  result.append(" = new GridBagConstraints();");
  result.append(sp.SPUtil.lineBreak());

  for (int i = 0; i < gridbags.length; i++) {
    String strval =
celem.getAttributeValue("constraint." +
gridbags[i]);
    if (strval != null) {
      result.append(gbcname);
      result.append('.');
      result.append(gridbags[i]);
      result.append(" = ");

      try {
        result.append(ValueConverter.convertValue(str
val, gridbagcls[i],
          java.awt.GridBagConstraints.class, elem));
      } catch (sp.XGGException e) {
        throw new XGGElementException(elem,
XGGConversionException
          .getNeededError("constraint." + gridbags[i],
strval, gridbagcls[i]
            .getName(), e));
      }

      result.append(';');
      result.append(sp.SPUtil.lineBreak());
    }
  }

  result.append(layoutvar);
  result.append(".setConstraints(");
  result.append(celem.getUsableName());
  result.append(',');
  result.append(gbcname);
  result.append(");");
  result.append(sp.SPUtil.lineBreak());

  return result.toString();
}
}

*********************************
FILE: sp/ format/code/FrameCodeHandler.java

package sp.format.code;
```

```
import sp.format.*;

/**
 * The code handler assigned to a JFrame, JApplet
and JInternalFrame. Adds more
 * attributes.
 */
public class FrameCodeHandler extends
PanelCodeHandler {
private boolean hasMenubar = false;

protected FrameCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

public Class<?> getClassAttr(String str) {
  if (str.equals("size")) {
    return java.awt.Dimension.class;
  } else if (str.equals("theme")) {
    return String.class;
  }
  return super.getClassAttr(str);
}

public StringBuffer attrCode(String qName, String
qVal)
  throws XGGElementException {
  if (qName.equals("size")) {
    StringBuffer result = new StringBuffer();

    if (!elem.isHead()) {
      result.append(elem.getUsableName());
      result.append('.');
    }
    result.append("setSize(");

    try {
      result.append(ValueConverter.convertValue(qVa
l, java.awt.Dimension.class,
        elem.getTargetClass(), elem));
    } catch (sp.XGGException e) {
      throw new XGGElementException(elem,
XGGConversionException
        .getNeededError("size", qVal,
java.awt.Dimension.class.getName(), e));
    }

    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    return result;
  } else if (qName.equals("theme")) {
    if (elem.isHead()) {
      StringBuffer result = new StringBuffer();

      int indents =
elem.getMain().getOptions().getIndents();

      result.append(sp.SPUtil.lineBreak());
      result.append("try");
      result.append(elem.getMain().getOptions().open
Brace());

      StringBuffer subresult = new StringBuffer();

      subresult.append("UIManager.setLookAndFeel(")
;

      try {
        String resultstr =
ValueConverter.convertValue(qVal, String.class,
elem
          .getTargetClass(), elem);
        if (resultstr.equals("\".system\""))
          resultstr =
"UIManager.getSystemLookAndFeelClassName()";
        else if (resultstr.equals("\".cross\""))
          resultstr =
"UIManager.getCrossPlatformLookAndFeelClassNam
e()";
        subresult.append(resultstr);
      } catch (sp.XGGException e) {
        throw new XGGElementException(elem,
XGGConversionException
          .getNeededError("theme", qVal,
String.class.getName(), e));
      }

      subresult.append(");");
      subresult.append(sp.SPUtil.lineBreak());

      subresult.append("SwingUtilities.updateCompon
entTreeUI(this);");
      subresult.append(sp.SPUtil.lineBreak());

      result.append(sp.SPUtil.indent(subresult.toStrin
g(), indents));
      result.append(sp.SPUtil.lineBreak());
      result.append('}');
      result.append(sp.SPUtil.lineBreak());
      result.append("catch(Exception __e) {}");
      result.append(sp.SPUtil.lineBreak());
      result.append(sp.SPUtil.lineBreak());

      return result;
    } else
      throw new XGGElementException(elem, new
XGGFormatException(
        "Only the Head element can have the theme
attribute."));
  } else
    return super.attrCode(qName, qVal);
}

public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
  throws XGGElementException {
  Class<?> ccls = celem.getTargetClass();

  if
(sp.SPUtil.isClassAssignable(javax.swing.JMenuBar.c
lass, ccls)) {
    StringBuffer result = new StringBuffer();

    if (hasMenubar)
      throw new XGGElementException(celem, new
XGGFormatException(elem
        .getClassName()
        + " can only have 1 menu bar."));

    if (!elem.isHead()) {
      result.append(elem.getUsableName());
      result.append('.');
    }
    result.append("setJMenuBar(");
    result.append(celem.getUsableName());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    hasMenubar = true;
    return result;
  }

  if
(sp.SPUtil.isClassAssignable(javax.swing.JComponen
t.class, ccls))
```

```
    return super.childCode(celem, pelem);

  return null;
}

private void fillAttr(StringBuffer result, String
attrname, String replacement) {
  if (!elem.isAttributeExist(attrname)) {
    if (!elem.isHead()) {
      result.append(elem.getUsableName());
      result.append('.');
    }
    result.append(replacement);
    result.append(sp.SPUtil.lineBreak());
  }
}

public StringBuffer lateCode() throws
XGGElementException {
  if (!elem.getClassName().equals("JApplet")) {
    StringBuffer result = new StringBuffer();

    result.append(sp.SPUtil.lineBreak());
    fillAttr(result, "size", "pack();");
    fillAttr(result, "visible", "setVisible(true);");
    if ((elem.getClassName().equals("JFrame")) &&
(elem.getMain().isDoMain()))
      fillAttr(result, "defaultCloseOperation",
        "setDefaultCloseOperation(JFrame.EXIT_ON_CL
OSE);");

    return result;
  } else if (elem.isAttributeExist("size")) {
    elem.getMain().giveWarning(
      "Attribute 'size', although supported, is" + "
unused in JApplet", elem);
  }

  return null;
}
}

**********
FILE: sp/ format/code/MenuBarCodeHandler.java

package sp.format.code;

import sp.format.*;

/**
 * The code handler assigned to a JMenuBar. Adds
only JMenus.
 */
public class MenuBarCodeHandler extends
DefaultCodeHandler {
protected MenuBarCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
  throws XGGElementException {
  StringBuffer result = new StringBuffer();

  Class<?> ccls = celem.getTargetClass();

  if
(sp.SPUtil.isClassAssignable(javax.swing.JMenu.class
, ccls)) {
    result.append(elem.getUsableName());
    result.append(".add(");
    result.append(celem.getUsableName());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
    return result;
  }
  return null;
}
}

**********
FILE: sp/ format/code/MenuCodeHandler.java

package sp.format.code;

import sp.format.*;

/**
 * The code handler assigned to a JMenu. Adds only
JMenuItems and JSeparators.
 */
public class MenuCodeHandler extends
DefaultCodeHandler {
protected MenuCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

public StringBuffer childCode(XGGElement celem,
XGGElement pelem)
  throws XGGElementException {
  StringBuffer result = new StringBuffer();

  Class<?> ccls = celem.getTargetClass();

  if
(sp.SPUtil.isClassAssignable(javax.swing.JMenuItem.
class, ccls)) {
    result.append(elem.getUsableName());
    result.append(".add(");
    result.append(celem.getUsableName());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
    return result;
  } else if
(sp.SPUtil.isClassAssignable(javax.swing.JSeparator.
class, ccls)) {
    result.append(elem.getUsableName());
    if (celem.countAttributes() == 0)
      result.append(".addSeparator();");
    else {
      result.append(".add(");
      result.append(celem.getUsableName());
      result.append(");");
    }
    result.append(sp.SPUtil.lineBreak());
    return result;
  }
  return null;
}
}

**********
FILE: sp/ format/code/RadioButtonCodeHandler.java

package sp.format.code;

import sp.format.*;

/**
 * The code handler assigned to a JRadioButton.
Sets up the ButtonGroup to be
 * used by the JRadioButtons.
 */
public class RadioButtonCodeHandler extends
DefaultCodeHandler {
```

```
protected RadioButtonCodeHandler(Class<?>
aClass, XGGElement anElem) {
  super(aClass, anElem);
}

public Class<?> getClassAttr(String str) {
  if (str.equals("group")) {
    return javax.swing.ButtonGroup.class;
  }

  return super.getClassAttr(str);
}

public StringBuffer attrCode(String qName, String
qVal)
  throws XGGElementException {
  if (qName.equals("group")) {
    StringBuffer result = new StringBuffer();

    String varname = null;
    try {
      varname = ValueConverter.convertValue(qVal,
        javax.swing.ButtonGroup.class,
elem.getTargetClass(), elem);
    } catch (sp.XGGException e) {
      throw new XGGElementException(elem,
XGGConversionException
        .getNeededError("group", qVal,
javax.swing.ButtonGroup.class.getName(),
          e));
    }

    result.append(varname);
    result.append(".add(");
    result.append(elem.getUsableName());
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    return result;
  }
  return super.attrCode(qName, qVal);
}
}

**********
FILE: sp/ format/code/ComboBoxCodeHandler.java

package sp.format.code;

import sp.format.*;

/**
 * The code handler assigned to a JComboBox.
Handles items.
 */
public class ComboBoxCodeHandler extends
DefaultCodeHandler {
private int value = 1;

protected ComboBoxCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

public StringBuffer itemCode(XGGItem item) throws
XGGElementException {
  StringBuffer result = new StringBuffer();

  String valuestr = item.getAttributeValue("value");
  if (valuestr == null) {
    valuestr = "Item " + value;
    value++;
  }

  result.append(elem.getUsableName());
  result.append(".addItem(");
  try {
    result.append(ValueConverter.convertValue(valu
estr, Object.class, elem
      .getTargetClass(), elem));
  } catch (sp.XGGException e) {
    throw new XGGElementException(elem, new
XGGTagException(item,
      XGGConversionException.getNeededError("valu
e", valuestr, Object.class
        .getName(), e)));
  }
  result.append(");");
  result.append(sp.SPUtil.lineBreak());

  return result;
}
}

**********
FILE: sp/ format/code/ListCodeHandler.java

package sp.format.code;

import sp.code.*;
import sp.compiler.*;
import sp.format.*;

/**
 * The code handler assigned to a JList. Handles
items.
 */
public class ListCodeHandler extends
DefaultCodeHandler {
private int value = 1;
private String listname = null;
private CompilerOpts cOpts;

protected ListCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
  if (anElem != null)
    cOpts = anElem.getMain().getOptions();
}

public StringBuffer itemCode(XGGItem item) throws
XGGElementException {
  StringBuffer result = new StringBuffer();

  String valuestr = item.getAttributeValue("value");
  if (valuestr == null) {
    valuestr = "Item " + value;
    value++;
  }

  if (listname == null) {
    listname =
elem.getMain().getNewName(java.util.Vector.class);

    String str = cOpts.getGeneric("Vector", "Object");
    JVar jv = new JVar(0, listname, str, "new " + str +
"()");

    result.append(jv.toString());
    result.append(sp.SPUtil.lineBreak());
  }

  result.append(listname);
  result.append(".add(");
  try {
    result.append(ValueConverter.convertValue(valu
estr, Object.class, elem
      .getTargetClass(), elem));
```

```
  } catch (sp.XGGException e) {
    throw new XGGElementException(elem, new
XGGTagException(item,
      XGGConversionException.getNeededError("valu
e", valuestr, Object.class
        .getName(), e)));
  }
  result.append(");");
  result.append(sp.SPUtil.lineBreak());

  return result;
}

public StringBuffer lateCode() throws
XGGElementException {
  if (listname != null) {
    elem.getMain().addImport("java.util");

    StringBuffer result = new StringBuffer();

    result.append(elem.getUsableName());
    result.append(".setListData(");
    result.append(listname);
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    return result;
  }
  return null;
}
}

**********
FILE: sp/ format/code/TreeCodeHandler.java

package sp.format.code;

import sp.format.*;
import java.util.*;

/**
 * The code handler assigned to a JTree. Handles
item nodes.
 */
public class TreeCodeHandler extends
DefaultCodeHandler {
private int root = 0;
private int value = 1;
private int childrennum = 1;
private String treemodel = null;

protected TreeCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
}

private String cycleTree(StringBuffer result,
XGGItem item, XGGItem parentitem)
  throws XGGElementException {
  String name = getNodeName(item);
  String valuestr = null;

  try {
    valuestr =
ValueConverter.convertValue(getNodeValue(item),
Object.class,
      elem.getTargetClass(), elem);
  } catch (sp.XGGException e) {
    throw new XGGElementException(elem, new
XGGTagException(item,
      XGGConversionException.getNeededError("valu
e", getNodeValue(item),
        Object.class.getName(), e)));
  }

  Iterator<?> i = item.getChildren();
  if (i.hasNext()) {
    result.append("DefaultMutableTreeNode ");
    result.append(name);
    result.append(" = new
DefaultMutableTreeNode(");
    result.append(valuestr);
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    while (i.hasNext()) {
      String childname = cycleTree(result, (XGGItem)
i.next(), item);

      result.append(name);
      result.append(".add(");
      result.append(childname);
      result.append(");");
      result.append(sp.SPUtil.lineBreak());
    }
  } else
    name = "new DefaultMutableTreeNode(" +
valuestr + ')';

  return name;
}

private String getNodeName(XGGItem item) {
  String result = elem.getUsableName() + "_C" +
childrennum;
  childrennum++;
  return result;
}

private String getNodeValue(XGGItem item) {
  String result = item.getAttributeValue("value");
  if (result == null) {
    result = "Item " + value;
    value++;
  }
  return result;
}

public StringBuffer itemCode(XGGItem item) throws
XGGElementException {
  if (root > 0)
    throw new XGGElementException(elem, new
XGGTagException(item,
      new XGGFormatException("JTree can only have
1 root node.")));
  root++;

  StringBuffer result = new StringBuffer();

  String rootname = cycleTree(result, item, null);

  if (treemodel == null) {
    treemodel = elem.getMain().getNewName(
      javax.swing.tree.DefaultTreeModel.class);

    result.append("DefaultTreeModel ");
    result.append(treemodel);
    result.append(" = new DefaultTreeModel(");
    result.append(rootname);
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
  }

  return result;
}

public StringBuffer lateCode() throws
XGGElementException {
```

```
  if (treemodel != null) {
    elem.getMain().addImport("javax.swing.tree");

    StringBuffer result = new StringBuffer();

    result.append(elem.getUsableName());
    result.append(".setModel(");
    result.append(treemodel);
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

    return result;
  }
  return null;
}
}

**********
FILE: sp/ format/code/TableCodeHandler.java

package sp.format.code;

import sp.code.*;
import sp.format.*;
import sp.compiler.*;
import java.util.*;

/**
 * The code handler assigned to a JTree. Handles
row values and column creation.
 */
public class TableCodeHandler extends
DefaultCodeHandler {
private int value = 1;
private boolean columnok = true;

private String tablemodel = null;
private CompilerOpts cOpts;

protected TableCodeHandler(Class<?> aClass,
XGGElement anElem) {
  super(aClass, anElem);
  if (anElem != null)
    cOpts = anElem.getMain().getOptions();
}

private String getColumnValue(XGGItem item) {
  String result = item.getAttributeValue("value");
  if (result == null) {
    result = "Column " + value;
    value++;
  }
  return result;
}

private String convertValue(String str, XGGItem
item)
  throws XGGElementException {
  try {
    return ValueConverter.convertValue(str,
Object.class,
      elem.getTargetClass(), elem);
  } catch (sp.XGGException e) {
    throw new XGGElementException(elem, new
XGGTagException(item,
      XGGConversionException.getNeededError("valu
e", str, Object.class
        .getName(), e)));
  }
}

public StringBuffer itemCode(XGGItem item) throws
XGGElementException {
  StringBuffer result = new StringBuffer();

  if (tablemodel == null) {
    tablemodel = elem.getMain().getNewName(
      javax.swing.table.DefaultTableModel.class);

    result.append("DefaultTableModel ");
    result.append(tablemodel);
    result.append(" = new DefaultTableModel();");
    result.append(sp.SPUtil.lineBreak());
  }

  if (item.getTagType().equals("Column")) {
    if (!columnok)
      throw new XGGElementException(elem, new
XGGTagException(item,
        new XGGFormatException("Columns go first
before Rows in JTable.")));

    result.append(tablemodel);
    result.append(".addColumn(");
    result.append(convertValue(getColumnValue(ite
m), item));
    result.append(");");
    result.append(sp.SPUtil.lineBreak());
  } else {
    columnok = false;

    String listname =
elem.getMain().getNewName(java.util.Vector.class);

    String str = cOpts.getGeneric("Vector", "Object");
    JVar jv = new JVar(0, listname, str, "new " + str +
"()");

    result.append(jv.toString());
    result.append(sp.SPUtil.lineBreak());

    Iterator<?> i = item.getChildren();
    while (i.hasNext()) {
      XGGItem child = (XGGItem) i.next();
      String valuestr =
child.getAttributeValue("value");
      if (valuestr == null)
        valuestr = "";

      result.append(listname);
      result.append(".add(");
      result.append(convertValue(valuestr, item));
      result.append(");");
      result.append(sp.SPUtil.lineBreak());
    }

    result.append(tablemodel);
    result.append(".addRow(");
    result.append(listname);
    result.append(");");
    result.append(sp.SPUtil.lineBreak());

  }

  return result;
}

public StringBuffer lateCode() throws
XGGElementException {
  if (tablemodel != null) {
    elem.getMain().addImport("javax.swing.table");

    StringBuffer result = new StringBuffer();

    result.append(elem.getUsableName());
    result.append(".setModel(");
    result.append(tablemodel);
    result.append(");");
```

```
    result.append(sp.SPUtil.lineBreak());

    return result;
   }
  return null;
}
}

**********
FILE: sp/ format/code/SeparatorCodeHandler.java

package sp.format.code;

import sp.format.*;

/**
 * The code handler assigned to a JSeparator.
 */
public class SeparatorCodeHandler extends
CodeHandler {
  protected SeparatorCodeHandler(Class<?> aClass,
XGGElement anElem)  {
    super(aClass,anElem);
  }

  public StringBuffer openCode(String initstr,
boolean tocreate)
   throws XGGElementException {
    if (elem.countAttributes() == 0)
      return null;
   return super.openCode(initstr,tocreate);
  }
}

**********
FILE: sp/ format/code/ValueConverter.java

package sp.format.code;

import sp.format.*;
import sp.format.attribute.*;
import sp.compiler.*;
import org.antlr.runtime.*;
import java.util.*;
import java.beans.*;

/**
 * The class that handles the default attribute value
validation and conversion.
 * This can handle constants and detect special
inputs. It can also parse
 * expressions, although there is no guarantee that
it will work in a .java file
 * due o the fact that it does not know if
fields/variables used are existing.
 */
public final class ValueConverter {
public static String convertValue(String value,
Class<?> cls,
  Class<?> extracls, XGGElement elem) throws
XGGConversionException,
  XGGForwardRefException,
XGGSpecialAttributeException {
  return convertValue(value, cls, extracls, elem, null,
false);
}

public static String getNullValue(Class<?> cls) {
  if ((cls == int.class) || (cls == byte.class) || (cls ==
long.class)
    || (cls == short.class))
    return "0";
  else if (cls == float.class)
    return "0.0f";
  else if (cls == double.class)
    return "0.0";
  else if (cls == boolean.class)
    return "false";
  else if (cls == char.class)
    return "(char)0";
  else if ((cls == String.class) || (cls ==
CharSequence.class)
    || (cls == Object.class))
    return "\"\"";

  return "null";
}

private static String intConvert(String str) {
  try {
    return Integer.toString(Integer.parseInt(str));
  } catch (NumberFormatException e) {
    if ((str.length() == 3) && (str.charAt(0) == '\'')
      && (str.charAt(2) == '\''))
      return "(int)'" +
sp.SPUtil.escapeChar(str.charAt(1)) + "'";
  }
  return null;
}

private static String floatConvert(String str) {
  try {
    return Float.toString(Float.parseFloat(str)) + 'f';
  } catch (NumberFormatException e) {}
  return null;
}

private static String doubleConvert(String str) {
  try {
    return Double.toString(Double.parseDouble(str));
  } catch (NumberFormatException e) {}
  return null;
}

public static String convertValue(String value,
Class<?> cls,
  Class<?> extracls, XGGElement elem,
XGGAttribute xggattr, boolean allowcross)
  throws XGGConversionException,
XGGForwardRefException,
  XGGSpecialAttributeException {
  XGGFormat xggMain = elem.getMain();

  String result = value;

  if ((result.indexOf('{') == 0)
    && (result.indexOf('}') == result.length() - 1)) {
    try {
      XGGSpecialAttribute specattr = new
XGGSpecialAttribute(result, cls);
     specattr.parse();

     int type = specattr.getAttributeType();
     if (type ==
XGGSpecialAttribute.SPEC_ATTR_VAR) {
       String specstr = specattr.getValue();
       XGGElement target =
xggMain.getElementByName(specstr);

       if (target == null) {
         String varstr =
xggMain.checkIdName(specstr);

         if (varstr != null) {
           Class<?> varcls =
xggMain.checkElement(varstr);

           if (varcls == null)
```

```
            throw new
XGGSpecialAttributeException(10, specattr);
          else if (!sp.SPUtil.isClassAssignable(cls,
varcls))
            throw new
XGGSpecialAttributeException(11, specattr);

          return specstr;
        }

        throw new XGGSpecialAttributeException(10,
specattr);
      } else if (target.isInLoop() != elem.isInLoop())
        throw new XGGSpecialAttributeException(9,
specattr);
      else if (!sp.SPUtil.isClassAssignable(cls,
target.getTargetClass()))
        throw new XGGSpecialAttributeException(11,
specattr);
      else if (!target.isCreated()) {
        if (allowcross)
          throw new XGGForwardRefException(target,
xggattr);
        else
          throw new XGGSpecialAttributeException(6,
specattr);
      }

      return specstr;
    } else if (type ==
XGGSpecialAttribute.SPEC_ATTR_LATE) {
      String strval = specattr.getValue();
      xggMain.addIdNames(strval,
sp.SPUtil.getShortName(cls.getName()),
        getNullValue(cls));

      return strval;
    } else if (type ==
XGGSpecialAttribute.SPEC_ATTR_MODELF) {
      XGGClassModel xmodel =
checkClassModel(xggMain.getModel(), specattr,
        elem);

      try {
        String strval =
xmodel.getVarCode(specattr.getValue(), cls);

        if (strval == null)
          throw new
XGGSpecialAttributeException(23, specattr);

        return strval;
      } catch (XGGSpecialAttributeException e) {
        throw new
XGGSpecialAttributeException(e.getErrorType(),
specattr);
      }
    } else if (type ==
XGGSpecialAttribute.SPEC_ATTR_MODELM) {
      XGGClassModel xmodel =
checkClassModel(xggMain.getModel(), specattr,
        elem);

      try {
        String strval =
xmodel.getVarMethod(specattr.getValue(), cls);

        if (strval == null)
          throw new
XGGSpecialAttributeException(24, specattr);

        return strval;
      } catch (XGGSpecialAttributeException e) {
        throw new
XGGSpecialAttributeException(e.getErrorType(),
specattr);
      }
    } else if ((type ==
XGGSpecialAttribute.SPEC_ATTR_BIND)
      || (type ==
XGGSpecialAttribute.SPEC_ATTR_BINDTO)
      || (type ==
XGGSpecialAttribute.SPEC_ATTR_BINDFROM)) {
      XGGClassModel xmodel =
checkClassModel(xggMain.getModel(), specattr,
        elem);

      try {
        PropertyDescriptor pd = xmodel
          .getBindMethod(specattr.getValue(), cls);
        FeatureDescriptor fdesc = null;

        if (pd == null)
          throw new
XGGSpecialAttributeException(30, specattr);

        if (xggattr == null)
          throw new
XGGSpecialAttributeException(31, specattr);
        else {
          fdesc = xggattr.getDescriptor();
          if ((fdesc == null) || (!(fdesc instanceof
PropertyDescriptor)))
            throw new
XGGSpecialAttributeException(31, specattr);
        }

        XGGBind bind = new XGGBind(elem, pd,
(PropertyDescriptor) fdesc,
          specattr.getAttributeType());

        elem.addBind(fdesc.getName(), bind);

        return
XGGClassModel.getSafeValue(xmodel.getVarName()
, pd
          .getReadMethod().getName()
          + "()", cls, true);
      } catch (XGGSpecialAttributeException e) {
        throw new
XGGSpecialAttributeException(e.getErrorType(),
specattr);
      }
    } else if (type ==
XGGSpecialAttribute.SPEC_ATTR_DBASE) {
      XGGModel xmodel = xggMain.getModel();
      if (xmodel == null)
        throw new XGGSpecialAttributeException(20,
specattr);

      int modeltype = xmodel.getModelType();
      if (modeltype != XGGModel.MODEL_DBASE)
        throw new XGGSpecialAttributeException(40,
specattr);

      if (!elem.isInLoop())
        throw new XGGSpecialAttributeException(28,
specattr);

      XGGDataModel dmodel = (XGGDataModel)
xmodel;
      try {
        String strval =
dmodel.getData(specattr.getValue(), cls);

        if (strval == null)
```

```
          throw new
XGGSpecialAttributeException(41, specattr);

        return strval;
       } catch (XGGSpecialAttributeException e) {
        throw new
XGGSpecialAttributeException(e.getErrorType(),
specattr);
       }
      }
    } catch (XGGSpecialAttributeException e) {
     e.setAttribute(xggattr);
     throw e;
    }
  }

  if (value.startsWith("\\{"))
    result = value.replaceFirst("\\\\\\{", "{");

  String temp = null;

  if (result.indexOf('.') > 0)
    temp = xggMain.getConstantValue(result, cls);

  if (temp != null)
    return temp;
  else {
    temp =
xggMain.getConstantValue(sp.SPUtil.getShortName(
extracls.getName())
      + '.' + result, cls);
    if (temp != null)
      return temp;
  }

  if (cls == int.class) {
    temp = intConvert(result);
    if (temp != null)
      return temp;
    throw new XGGConversionException("Conversion
error!");
  } else if (cls == byte.class)
    try {
      return Byte.toString(Byte.parseByte(result));
    } catch (NumberFormatException e) {
      throw new
XGGConversionException("Conversion error!");
    }
  else if (cls == float.class) {
    temp = floatConvert(result);
    if (temp != null)
      return temp;
    throw new XGGConversionException("Conversion
error!");
  } else if (cls == double.class) {
    temp = doubleConvert(result);
    if (temp != null)
      return temp;
    throw new XGGConversionException("Conversion
error!");
  } else if (cls == long.class)
    try {
      return Long.toString(Long.parseLong(result));
    } catch (NumberFormatException e) {
      throw new
XGGConversionException("Conversion error!");
    }
  else if (cls == short.class)
    try {
      return Short.toString(Short.parseShort(result));
    } catch (NumberFormatException e) {
      throw new
XGGConversionException("Conversion error!");
    }
  else if (cls == boolean.class) {
    if ((!result.equals("true")) && (!
result.equals("false")))
      throw new
XGGConversionException("Conversion error!");
    return result;
  } else if (cls == char.class) {
    if ((result.length() != 3)||(result.charAt(0)!='\'')||
     (result.charAt(2)!='\'')) {
      temp = intConvert(result);
      if (temp != null)
        return "(char)"+temp;
    }
   if (result.length() != 1)
      throw new
XGGConversionException("Conversion error!");
    return '\'' + sp.SPUtil.escapeStr(result) + '\'';
  } else if ((cls == String.class) || (cls ==
CharSequence.class))
    return '"' + sp.SPUtil.escapeStr(result) + '"';
  else if ((cls == Object.class) || (cls ==
Number.class)) {
    temp = intConvert(result);
    if (temp != null)
      return "new Integer(" + temp + ')';

    temp = floatConvert(result);
    if (temp != null)
      return "new Float(" + temp + ')';

    temp = doubleConvert(result);
    if (temp != null)
      return "new Double(" + temp + ')';

    if ((result.equals("true")) ||
(result.equals("false")))
      return "new Boolean(" + result + ')';

    if (cls == Object.class)
      return '"' + sp.SPUtil.escapeStr(result) + '"';
    else
      throw new
XGGConversionException("Conversion error!");
  }

  value = value.trim();

  AbstractParser ap = AbstractParser.getParser(cls,
value, elem);

  if (ap != null) {
    ap.parse();
    if (ap.hasError())
      throw new
XGGConversionException(ap.getError());

    return ap.getAnswer();
  }

  throw new XGGConversionException(
    "Unsupported Type. Use constants, existing
variables or late binding "
      + "{late [variable name]} instead.");
}

public static String[] parseArgs(String str) throws
XGGConversionException {
  int stkPrtsis = 0;
  boolean quote1 = false;
  boolean quote2 = false;
  boolean quoteused = false;
  LinkedList<String> ll = new LinkedList<String>();
  StringBuffer curSB = new StringBuffer();
```

```
  for (int i = 0; i < str.length(); i++) {
    char c = str.charAt(i);
    if (c == ',') {
      if ((stkPrtsis == 0) && (!quote1) && (!quote2)) {
        ll.add(curSB.toString());
        quoteused = false;
        curSB.delete(0, curSB.length());
      } else
        curSB.append(c);
    } else if (c == '(') {
      if ((!quote1) && (!quote2))
        stkPrtsis++;
      curSB.append(c);
    } else if (c == ')') {
      if ((!quote1) && (!quote2)) {
        stkPrtsis--;
        if (stkPrtsis < 0)
          throw new XGGConversionException("Invalid
Token ')'. No matching '('");
      }
      curSB.append(c);
    } else if (stkPrtsis > 0)
      curSB.append(c);
    else if (c == '\'') {
      if ((!quote1) && (!quote2)) {
        quote1 = true;
        if (quoteused)
          throw new XGGConversionException(
            "Invalid Token '''. Only 1 quote per
argument");
        if (curSB.length() > 0)
          throw new XGGConversionException(
            "Invalid Token '''. Use quotes only at
beginning of argument");
        quoteused = true;
      } else if (quote1)
        quote1 = false;
      else
        curSB.append(c);
    } else if (c == '"') {
      if ((!quote1) && (!quote2)) {
        quote2 = true;
        if (quoteused)
          throw new XGGConversionException(
            "Invalid Token '\"'. Only 1 quote per
argument");
        if (curSB.length() > 0)
          throw new XGGConversionException(
            "Invalid Token '\"'. Use quotes only at
beginning of argument");
        quoteused = true;
      } else if (quote2)
        quote2 = false;
      else
        curSB.append(c);
    } else {
      if (quoteused)
        throw new XGGConversionException("Invalid
Token '" + c
          + "'. All characters must be inside quotes");
      curSB.append(c);
    }
  }

  if (stkPrtsis != 0)
    throw new XGGConversionException("Invalid
Token '('. No matching ')'");
  if (quote1)
    throw new XGGConversionException("Invalid
Token '''. No matching '''");
  if (quote2)
    throw new XGGConversionException("Invalid
Token '\"'. No matching '\"'");

  if ((ll.size() > 0) || (curSB.length() > 0))
    ll.add(curSB.toString());

  return ll.toArray(new String[ll.size()]);
}

public static String checkExpression(String value,
String attrstr)
  throws XGGFormatException {
  JavaLexer lexer = new JavaLexer(new
ANTLRStringStream("class a{void b(){if("
    + value + ");}}"));
  CommonTokenStream tokens = new
CommonTokenStream();
  tokens.setTokenSource(lexer);

  try {
    JavaParser parser = new JavaParser(tokens);
    parser.compilationUnit();

    if (parser.ErrMsg != null) {
      String err = "Error in expression.";
      if (attrstr != null)
        throw new XGGFormatException("At attribute
'" + attrstr + "': " + err);
      throw new XGGFormatException(err);
    }
  } catch (RecognitionException e) {
    throw new XGGFormatException("At attribute '"
+ attrstr
      + "': Unknown error in parsing");
  }

  return value;
}

public static String getWrapper(Class<?> cls) {
  if (cls == int.class)
    return "Integer";
  else if (cls == byte.class)
    return "Byte";
  else if (cls == float.class)
    return "Float";
  else if (cls == double.class)
    return "Double";
  else if (cls == long.class)
    return "Long";
  else if (cls == short.class)
    return "Short";
  else if (cls == boolean.class)
    return "Boolean";
  else if (cls == char.class)
    return "Character";

  return null;
}

public static String tryToString(String value, Class<?
> cls) {
  String str = getWrapper(cls);
  if (str != null)
    return str + ".toString(" + value + ')';

  return null;
}

public static String tryToObject(String value,
Class<?> cls) {
  String str = getWrapper(cls);
  if (str != null)
    return "new " + str + '(' + value + ')';

  return null;
}
```

```
private static XGGClassModel
checkClassModel(XGGModel xmodel,
  XGGSpecialAttribute specattr, XGGElement elem)
  throws XGGSpecialAttributeException {
  if (xmodel == null)
    throw new XGGSpecialAttributeException(20,
specattr);

  int type = xmodel.getModelType();

  if (type == XGGModel.MODEL_DBASE)
    throw new XGGSpecialAttributeException(21,
specattr);

  if ((type == XGGModel.MODEL_CLASSLIST) && (!
elem.isInLoop()))
    throw new XGGSpecialAttributeException(28,
specattr);

  return (XGGClassModel) xmodel;
}
}

**********
FILE: sp/ format/code/XGGConversionException.java

package sp.format.code;

import sp.*;
import sp.format.*;

/**
 * Thrown when an error occured while trying to
convert an attribute input to an
 * object.
 */
public class XGGConversionException extends
XGGException {
static final long serialVersionUID = 'c' * 'o' * 'n' * 'v'
* 'e' * 'r' * 't';
private String error = null;

public XGGConversionException(String anError) {
  super();
  error = anError;
}

public int getLineNumber() {
  return 0;
}

public int getColumnNumber() {
  return 0;
}

public String getError() {
  return error;
}

public static XGGException getNeededError(String
name, String type,
  String clsname, XGGException e) {
  if (e instanceof XGGConversionException)
    return new XGGValueException(name, type,
clsname, e.getError());
  return e;
}
}

**********
FILE: sp/format/attribute/AbstractParser.java

package sp.format.attribute;

import sp.format.*;

/**
 * The base class for special parsers used by
<code>ValueConverter</code>.
 */
public abstract class AbstractParser {
protected String toParse = null;
protected String answer = null;
protected String errorStr = null;
protected XGGFormat xggMain = null;
protected XGGElement elem = null;

public static AbstractParser getParser(Class<?> cls,
String str,
  XGGElement anElem) {
  if (cls == javax.swing.border.Border.class)
    return new BorderParser(str, anElem);
  else if (cls == java.awt.LayoutManager.class)
    return new LayoutParser(str, anElem);
  else if (cls == java.awt.Font.class)
    return new FontParser(str, anElem);
  else if (cls == java.awt.Dimension.class)
    return new DimensionParser(str, anElem);
  else if (cls == java.awt.Color.class)
    return new ColorParser(str, anElem);
  else if (cls == javax.swing.ButtonGroup.class)
    return new ButtonGroupParser(str, anElem);
  else if (cls == javax.swing.Icon.class)
    return new IconParser(str, anElem);
  else if (cls == java.awt.Insets.class)
    return new InsetsParser(str, anElem);
  else if (cls == javax.swing.KeyStroke.class)
    return new KeyStrokeParser(str, anElem);
  else if (cls == java.awt.Paint.class)
    return new PaintParser(str, anElem);
  else if (cls == java.awt.Stroke.class)
    return new StrokeParser(str, anElem);
  else if (cls == java.awt.Image.class)
    return new ImageParser(str, anElem);
  else if (cls == java.awt.Cursor.class)
    return new CursorParser(str, anElem);
  else if (cls == int[].class)
    return new IntArrayParser(str, anElem);
  else if (cls == double[].class)
    return new DoubleArrayParser(str, anElem);
  else if (WrapperParser.getWrappedClass(cls) !=
null)
    return new WrapperParser(cls, str, anElem);
  return null;
}

protected AbstractParser(String str, XGGElement
anElem) {
  toParse = str;
  elem = anElem;
  xggMain = elem.getMain();
}

public void parse() {
  answer = null;
  errorStr = null;
}

public boolean hasError() {
  return errorStr != null;
}

public String getError() {
  return errorStr;
}

public String getAnswer() {
```

```
  return answer;
}
}

*********************************
FILE: sp/format/attribute/FontParser.java

package sp.format.attribute;

import sp.format.*;

/**
 * The parser for the <code>Font</code> class.
Treats the argument as a string
 * and calls <code>Font.decode</code>.
 */
public class FontParser extends AbstractParser {
protected FontParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

public void parse() {
  super.parse();
  toParse = toParse.trim();

  answer = "Font.decode(\"" +
sp.SPUtil.escapeStr(toParse) + "\")";
  xggMain.addImport("java.awt");
}
}

*********************************
FILE: sp/format/attribute/LayoutParser.java

package sp.format.attribute;

import sp.format.*;
import sp.format.code.*;

/**
 * The parser for the
<code>LayoutManager</code> class. Treats the
argument as
 * a constructor call (without the
<code>new</code> keyword).
 */
public class LayoutParser extends AbstractParser {
private static final String[] LAYOUTNAMES =
{ "BorderLayout", "BoxLayout",
    "CardLayout", "FlowLayout", "GridBagLayout",
"GridLayout", "OverlayLayout",
    "SpringLayout" };

protected LayoutParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

private boolean checkLegal(String str) {
  return sp.SPUtil.binarySearchString(str,
LAYOUTNAMES, 0,
    LAYOUTNAMES.length - 1) != null;
}

public String[] paramcheck(String str, Class<?> cls,
Class<?>[] cparams) {
  String result[] = new String[cparams.length];

  String restarr[] = str.split(",");

  for (int i = 0; i < cparams.length; i++) {
    try {
      result[i] = ValueConverter
          .convertValue(restarr[i], cparams[i], cls, elem);
    } catch (sp.XGGException e) {
      errorStr = "Layout Syntax: On Argument " + (i +
1) + " should be " + '('
          + cparams[i].getName() + ')' + ": " +
e.getError();
      return null;
    }

    if (result[i] == null) {
      errorStr = "Layout Syntax: On Argument " + (i +
1);
      return null;
    }
  }

  return result;
}

public void parse() {
  super.parse();

  String value = toParse.trim();

  int inittok = value.indexOf('(');

  if (inittok <= 0) {
    errorStr = "Layout Syntax: '(' missing";
    return;
  }

  String mainname = value.substring(0, inittok);

  if (!checkLegal(mainname)) {
    errorStr = "Layout Syntax: not a layout";
    return;
  }

  if (value.charAt(value.length() - 1) != ')') {
    errorStr = "Layout Syntax: ')' missing";
    return;
  }

  value = value.substring(0, value.length() - 1);

  String resttokstr = value.substring(inittok + 1);

  String restarr[] = null;
  try {
    restarr = ValueConverter.parseArgs(resttokstr);
  } catch (XGGConversionException e) {
    errorStr = "Layout Syntax: " + e.getError();
    return;
  }
  int resttokcnt = restarr.length;

  if ((resttokcnt == 1) &&
(restarr[0].trim().equals("")))
    resttokcnt = 0;

  String params[] = null;

  if (mainname.equals("BorderLayout")) {
    xggMain.addImport("java.awt");
    switch (resttokcnt) {
    case 0:
      answer = "new BorderLayout()";
      return;
    case 2:
      params = paramcheck(resttokstr,
java.awt.BorderLayout.class, new Class[] {
          int.class, int.class });
      break;
    }
```

```
  } else if (mainname.equals("BoxLayout")) {
    xggMain.addImport("javax.swing");
    if (resttokcnt == 1) {
      String tempparams[] = paramcheck(resttokstr,
javax.swing.BoxLayout.class,
        new Class[] { int.class });
      if (tempparams != null) {
        Class<?> cls = elem.getTargetClass();

        try {
          if (cls.getMethod("getContentPane", new
Class<?>[] {}) == null)
            params = new String[]
{ elem.getUsableName(), tempparams[0] };
          else
            params = new String[]
{ elem.getUsableName() + ".getContentPane()",
              tempparams[0] };
        } catch (Exception e) {
          params = new String[]
{ elem.getUsableName(), tempparams[0] };
        }
      }
    }
  } else if (mainname.equals("CardLayout")) {
    xggMain.addImport("java.awt");
    switch (resttokcnt) {
    case 0:
      answer = "new CardLayout()";
      return;
    case 2:
      params = paramcheck(resttokstr,
java.awt.CardLayout.class, new Class[] {
          int.class, int.class });
      break;
    }
  } else if (mainname.equals("FlowLayout")) {
    xggMain.addImport("java.awt");
    switch (resttokcnt) {
    case 0:
      answer = "new FlowLayout()";
      return;
    case 1:
      params = paramcheck(resttokstr,
java.awt.FlowLayout.class,
        new Class[] { int.class });
      break;
    case 3:
      params = paramcheck(resttokstr,
java.awt.FlowLayout.class, new Class[] {
          int.class, int.class, int.class });
      break;
    }
  } else if (mainname.equals("GridBagLayout")) {
    xggMain.addImport("java.awt");
    if (resttokcnt == 0) {
      answer = "new GridBagLayout()";
      return;
    }
  } else if (mainname.equals("GridLayout")) {
    xggMain.addImport("java.awt");
    switch (resttokcnt) {
    case 0:
      answer = "new GridLayout()";
      return;
    case 2:
      params = paramcheck(resttokstr,
java.awt.GridLayout.class, new Class[] {
          int.class, int.class });
      break;
    case 4:
      params = paramcheck(resttokstr,
java.awt.GridLayout.class, new Class[] {
          int.class, int.class, int.class, int.class });
      break;
    }
  } else if (mainname.equals("OverlayLayout")) {
    xggMain.addImport("javax.swing");
    if (resttokcnt == 0) {
      Class<?> cls = elem.getTargetClass();

      try {
        if (cls.getMethod("getContentPane", new
Class<?>[] {}) == null)
          answer = "new OverlayLayout(" +
elem.getUsableName() + ")";
        else
          answer = "new OverlayLayout(" +
elem.getUsableName()
            + ".getContentPane())";
      } catch (Exception e) {
        answer = "new OverlayLayout(" +
elem.getUsableName() + ")";
      }
      return;
    }
  } else if (mainname.equals("SpringLayout")) {
    xggMain.addImport("java.awt");
    if (resttokcnt == 0) {
      answer = "new SpringLayout()";
      return;
    }
  }

  if (params != null) {
    StringBuffer resultant = new StringBuffer();

    resultant.append("new ");
    resultant.append(mainname);
    resultant.append('(');

    for (int i = 0; i < params.length; i++) {
      if (i != 0)
        resultant.append(", ");
      resultant.append(params[i]);
    }

    resultant.append(')');

    answer = resultant.toString();
  }

  if (answer == null) {
    if (errorStr == null)
      errorStr = "Layout Syntax: no layout with
matching arguments";
  }
}
}

**********
FILE: sp/format/attribute/BorderParser.java

package sp.format.attribute;

import sp.format.*;
import sp.format.code.*;
import java.lang.reflect.*;
import javax.swing.*;

/**
 * The parser for the <code>Border</code> class.
Treats the argument as a
 * constructor call (without the <code>new</code>
keyword).
 */
public class BorderParser extends AbstractParser {
```

```
private static final Method[] BorderMeths =
BorderFactory.class.getMethods();

protected BorderParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

private boolean checkLegal(String str) {
  for (int i = 0; i < BorderMeths.length; i++) {
    if (BorderMeths[i].getName().equals("create" +
str))
      return true;
  }
  return false;
}

public void parse() {
  super.parse();

  String value = toParse.trim();

  if (value.equals("null")) {
    answer = "null";
    return;
  }

  int inittok = value.indexOf('(');

  if (inittok <= 0) {
    errorStr = "Border Syntax: '(' missing";
    return;
  }

  String mainname = value.substring(0, inittok);

  if (!checkLegal(mainname)) {
    errorStr = "Border Syntax: not a border";
    return;
  }

  Class<?> cls = null;
  try {
    cls = Class.forName("javax.swing.border." +
mainname, false, null);
    xggMain.addImport("javax.swing.border");
  } catch (ClassNotFoundException e) {
    errorStr = "Error: Initializing '" + mainname + "'";
    return;
  }

  if (value.charAt(value.length() - 1) != ')') {
    errorStr = "Border Syntax: ')' missing";
    return;
  }

  value = value.substring(0, value.length() - 1);

  String resttokstr = value.substring(inittok + 1);

  String restarr[] = null;
  try {
    restarr = ValueConverter.parseArgs(resttokstr);
  } catch (XGGConversionException e) {
    errorStr = "Border Syntax: " + e.getError();
    return;
  }
  int resttokcnt = restarr.length;

  if ((resttokcnt == 1) &&
(restarr[0].trim().equals("")))
    resttokcnt = 0;

  for (int i = 0; i < BorderMeths.length; i++) {
    Class<?> params[] =
BorderMeths[i].getParameterTypes();
    if ((params.length == resttokcnt)
      && (BorderMeths[i].getName().equals("create"
+ mainname))) {
      errorStr = null;

      String rparams[] = new String[params.length];
      for (int j = 0; j < params.length; j++) {
        try {
          rparams[j] =
ValueConverter.convertValue(restarr[j], params[j],
cls,
              elem);
        } catch (sp.XGGException e) {
          errorStr = "Border Syntax: On Argument " +
(j + 1) + " should be "
              + '(' + params[j].getName() + ')' + ": " +
e.getError();
          break;
        }

        if (rparams[j] == null) {
          errorStr = "Border Syntax: On Argument " +
(j + 1);
          break;
        }
      }

      if (errorStr != null)
        continue;
      StringBuffer resultant = new StringBuffer();
      resultant.append("BorderFactory.create");
      resultant.append(mainname);
      resultant.append('(');

      for (int j = 0; j < rparams.length; j++) {
        if (rparams[j] == null) {
          resultant = null;
          break;
        } else {
          if (j != 0)
            resultant.append(", ");
          resultant.append(rparams[j]);
        }
      }

      if (resultant != null) {
        resultant.append(')');
        answer = resultant.toString();
        break;
      }
    }
  }

  if (answer == null) {
    if (errorStr == null)
      errorStr = "Border Syntax: no border with
matching arguments";
  }
}
}

*********************************
FILE: sp/format/attribute/ColorParser.java

package sp.format.attribute;

import sp.format.*;

/**
 * The parser for the <code>Color</code> class.
Checks if the argument
```

```
 * corresponds to the constants in Color class
(blue,red etc.). If not
 * successful, then treats the argument as a as a
string and calls
 * <code>Color.decode</code>.
 */
public class ColorParser extends AbstractParser {
protected String className = "Color";

protected ColorParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

public void parse() {
  String tintstr = null;

  super.parse();

  if (toParse.endsWith(".darker")) {
    tintstr = ".darker()";
    toParse = toParse.substring(0, toParse.length() -
7);
  } else if (toParse.endsWith(".brighter")) {
    tintstr = ".brighter()";
    toParse = toParse.substring(0, toParse.length() -
9);
  }

  answer = xggMain.getConstantValue("Color." +
toParse, java.awt.Color.class);
  if (answer != null) {
    if (tintstr != null)
      answer += tintstr;
    xggMain.addImport("java.awt");
    return;
  }

  try {
    java.awt.Color.decode(toParse);
    xggMain.addImport("java.awt");
    answer = "Color.decode(\"" +
sp.SPUtil.escapeStr(toParse) + "\")";
    if (tintstr != null)
      answer += tintstr;
    return;
  } catch (NumberFormatException e) {}

  errorStr = className
    + " Syntax: Color must be represented either by
#RRGGBB format "
    + "or by using the available color constants in
the Color class. (.darker "
    + "and .brighter can be suffixed)";
}
}

**********
FILE: sp/format/attribute/PaintParser.java

package sp.format.attribute;

import sp.format.*;
import sp.format.code.*;

/**
 * The parser for the <code>Paint</code> class.
Parses the argument as a
 * Color. If not successful then Treats the argument
as a constructor call for
 * <code>GradientPaint</code> (without the
<code>new</code> keyword).
 */
public class PaintParser extends ColorParser {
private static Class<?> gradientcls[] = new Class<?
>[] { float.class,
    float.class, java.awt.Color.class, float.class,
float.class,
    java.awt.Color.class, boolean.class };

protected PaintParser(String str, XGGElement
anElem) {
  super(str, anElem);
  className = "Paint";
}

public void parse() {
  answer = null;
  errorStr = null;

  String value = toParse.trim();

  if (value.startsWith("GradientPaint(") &&
value.endsWith(")")) {
    value = value.substring(14, value.length() - 1);

    String restarr[] = null;
    try {
      restarr = ValueConverter.parseArgs(value);
    } catch (XGGConversionException e) {
      errorStr = "Paint Syntax: " + e.getError();
      return;
    }
    int resttokcnt = restarr.length;

    if ((resttokcnt != 6) && (resttokcnt != 7)) {
      errorStr = "Paint Syntax: Error in GradientPaint
syntax. Must be "
        +
"GradientPaint(float,float,Color,float,float,Color[,bool
ean]).";
      return;
    }

    String resultarr[] = new String[resttokcnt];

    for (int i = 0; i < resttokcnt; i++) {
      try {
        if (gradientcls[i] == float.class)
          resultarr[i] =
ValueConverter.checkExpression(restarr[i], null);
        else
          resultarr[i] =
ValueConverter.convertValue(restarr[i],
            gradientcls[i], java.awt.Color.class, elem);
      } catch (sp.XGGException e) {
        errorStr = "Layout Syntax: On Argument " + (i
+ 1) + " should be "
          + '(' + gradientcls[i].getName() + ')' + ": " +
e.getError();
        return;
      }

      if (resultarr[i] == null) {
        errorStr = "Layout Syntax: On Argument " + (i
+ 1);
        return;
      }
    }

    StringBuffer resultant = new StringBuffer();
    resultant.append("new GradientPaint(");

    for (int i = 0; i < resultarr.length; i++) {
      if (i != 0)
        resultant.append(", ");
      resultant.append(resultarr[i]);
    }
```

```
    resultant.append(')');

    xggMain.addImport("java.awt");
    answer = resultant.toString();
    return;
  } else if ((value.indexOf('(') >= 0) ||
(value.indexOf(')') >= 0)) {
    errorStr = "Paint Syntax: Error in GradientPaint
syntax. Must be "
      +
"GradientPaint(float,float,Color,float,float,Color[,bool
ean]).";
    return;
  }

  super.parse();
}
}

**********
FILE: sp/format/attribute/DimensionParser.java

package sp.format.attribute;

import sp.format.*;
import sp.format.code.*;

/**
 * The parser for the <code>Dimension</code>
class. Has format "int1,int2".
 */
public class DimensionParser extends
AbstractParser {
protected int NUM_PARAMS = 2;
protected String clsName = "Dimension";
protected Class<?> clsUsed =
java.awt.Dimension.class;

protected DimensionParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

public void parse() {
  super.parse();
  toParse = toParse.trim();

  String params[] = toParse.split(",");

  if (params.length != NUM_PARAMS) {
    errorStr = clsName + " Syntax: invalid number of
arguments.";
  }

  String nparams[] = new String[params.length];
  for (int i = 0; i < params.length; i++) {
    try {
      nparams[i] =
ValueConverter.convertValue(params[i], int.class,
clsUsed,
        elem);
    } catch (sp.XGGException e) {
      errorStr = clsName + " Syntax: On Argument "
+ (i + 1) + " should be "
        + "(int)" + ": " + e.getError();
      return;
    }
  }

  StringBuffer resultant = new StringBuffer();

  resultant.append("new ");
  resultant.append(clsName);
  resultant.append('(');

  for (int i = 0; i < nparams.length; i++) {
    if (i != 0)
      resultant.append(", ");
    resultant.append(nparams[i]);
  }

  resultant.append(')');

  answer = resultant.toString();
  xggMain.addImport("java.awt");
}
}

**********
FILE: sp/format/attribute/InsetsParser.java

package sp.format.attribute;

import sp.format.*;

/**
 * The parser for the <code>Insets</code> class.
 * Has format "int1,int2,int3,int4".
 */
public class InsetsParser extends DimensionParser {
protected InsetsParser(String str, XGGElement
anElem) {
  super(str,anElem);
  NUM_PARAMS = 4;
  clsName = "Insets";
  clsUsed = java.awt.Insets.class;
}
}

**********
FILE: sp/format/attribute/IconParser.java

package sp.format.attribute;

import sp.format.*;

/**
 * The parser for the <code>Icon</code> class.
Treates the argument as a path
 * and calls the <code>ImageIcon(String)</code>
constructor.
 */
public class IconParser extends AbstractParser {
protected String addstr = "";

protected IconParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

public void parse() {
  super.parse();
  toParse = toParse.trim();

  char schar = toParse.charAt(0);
  if ((schar == '\\') || (schar == '/'))
    answer = "new ImageIcon(\"" +
sp.SPUtil.escapeStr(toParse) + "\")" + addstr;
  else {
    String resstr = "getClass().getResource(\"" +
sp.SPUtil.escapeStr(toParse)
      + "\")";

    answer = "((" + resstr + "!= null) ? new
ImageIcon(" + resstr + ')'
      + addstr + " : new ImageIcon(\"\")" + addstr +
')';
```

```
  }
}
}

**********
FILE: sp/format/attribute/ButtonGroupParser.java

package sp.format.attribute;

import sp.format.*;

/**
 * The parser for the <code>ButtonGroup</code>
class. Treats the argument as a
 * variable name and creates or returns the existing
the ButtonParser.
 */
public class ButtonGroupParser extends
AbstractParser {
protected ButtonGroupParser(String str,
XGGElement anElem) {
  super(str, anElem);
}

public void parse() {
  super.parse();
  xggMain.addImport("javax.swing");

  if ((sp.SPUtil.isLegalVar(toParse)) &&
(sp.SPUtil.isUnreservedVar(toParse))) {
    if (!xggMain.addIdNames(toParse, "ButtonGroup",
"new ButtonGroup()")) {
      String usedstr =
xggMain.checkIdName(toParse);
      if ((usedstr == null) || (!
usedstr.equals("ButtonGroup"))) {
        errorStr = "ButtonGroup Syntax: duplicate
variable name.";
        return;
      }
    }
    answer = toParse;
  } else
    errorStr = "ButtonGroup Syntax: invalid variable
name for ButtonGroup.";
}
}

**********
FILE: sp/format/attribute/KeyStrokeParser.java

package sp.format.attribute;

import sp.format.*;
import javax.swing.*;

/**
 * The parser for the <code>KeyStroke</code>
class. Treats the argument as a
 * string and calls
<code>KeyStroke.getKeyStroke</code>.
 */
public class KeyStrokeParser extends
AbstractParser {
protected KeyStrokeParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

public void parse() {
  super.parse();
  toParse = toParse.trim();
  if (KeyStroke.getKeyStroke(toParse) == null) {
    errorStr = "KeyStroke Syntax: invalid format
(examples of correct input:"
      + " 'INSERT', 'control DELETE', 'alt shift X', 'alt
shift released X'"
      + ", 'typed a' )";
    return;
  }
  answer = "KeyStroke.getKeyStroke(\"" +
sp.SPUtil.escapeStr(toParse) + "\")";
}
}

**********
FILE: sp/format/attribute/SrrokeParser.java

package sp.format.attribute;

import sp.format.*;
import sp.format.code.*;

/**
 * The parser for the <code>Stroke</code> class.
Treats the argument as a
 * constructor call for <code>GradientPaint</code>
(without the
 * <code>new</code> keyword).
 */
public class StrokeParser extends AbstractParser {
private static Class<?> bstrokcls[] = new Class[]
{ float.class, int.class,
    int.class, float.class, float[].class, float.class };

protected StrokeParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

public void parse() {
  super.parse();

  String value = toParse.trim();

  int resttokcnt = 0;
  String restarr[] = value.split(",");
  if (value.length() > 0)
    resttokcnt = restarr.length;

  if (resttokcnt == 0) {
    xggMain.addImport("java.awt");
    answer = "new BasicStroke()";
  } else if ((resttokcnt == 1) || (resttokcnt == 3) ||
(resttokcnt == 4)
    || (resttokcnt >= 6)) {
    String resultarr[] = value.split(",");

    for (int i = 0; i < restarr.length; i++) {
      try {
        if (i > 3)
          resultarr[i] = restarr[i];
        else if (bstrokcls[i] == float.class)
          resultarr[i] =
ValueConverter.checkExpression(restarr[i], null);
        else
          resultarr[i] =
ValueConverter.convertValue(restarr[i], bstrokcls[i],
            java.awt.BasicStroke.class, elem);
      } catch (sp.XGGException e) {
        errorStr = "Layout Syntax: On Argument " + (i
+ 1) + " should be "
          + '(' + bstrokcls[i].getName() + ')' + ": " +
e.getError();
        return;
      }
```

```
    if (resultarr[i] == null) {
      errorStr = "Layout Syntax: On Argument " + (i
+ 1);
      return;
    }
  }

  StringBuffer resultant = new StringBuffer();
  resultant.append("new BasicStroke(");

  for (int i = 0; i < resultarr.length; i++) {
    if (i != 0)
      resultant.append(", ");
    resultant.append(resultarr[i]);
  }

  resultant.append(')');

  xggMain.addImport("java.awt");
  answer = resultant.toString();
 } else
  errorStr = "Stroke Syntax: Wrong number of
arguments (" + resttokcnt + ')';
}
}

**********
FILE: sp/format/attribute/ImageParser.java

package sp.format.attribute;

import sp.format.*;

/**
 * The parser for the <code>Image</code> class.
Typically the same as ImageIcon
 * parser except that method
<code>getImage()</code> is used.
 */
public class ImageParser extends IconParser {
protected ImageParser(String str, XGGElement
anElem) {
  super(str,anElem);
  xggMain.addImport("java.awt");
  addstr = ".getImage()";
}
}

**********
FILE: sp/format/attribute/CursorParser.java

package sp.format.attribute;

import sp.format.*;
import sp.format.code.*;

/**
 * The parser for the <code>Cursor</code> class.
Treats the argument as an
 * integer and calls constructor
<code>Cursor(int)</code>.
 */
public class CursorParser extends AbstractParser {
protected CursorParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

public void parse() {
  super.parse();
  toParse = toParse.trim();

  try {
    String intstr =
ValueConverter.convertValue(toParse, int.class,
      java.awt.Cursor.class, elem);

    xggMain.addImport("java.awt");
    answer = "new Cursor(" + intstr + ")";
  } catch (sp.XGGException e) {
    errorStr = "Cursor Syntax: input must be an
integer or a constant in"
      + " the Cursor class";
  }
}
}

**********
FILE: sp/format/attribute/ntArrayParser.java

package sp.format.attribute;

import sp.format.*;
import sp.format.code.*;

/**
 * The parser for the <code>int[]</code> class.
Has format "int1,int2,..,int#".
 */
public class IntArrayParser extends AbstractParser {
protected String clsName = "int";
protected Class<?> clsused = int.class;

protected IntArrayParser(String str, XGGElement
anElem) {
  super(str, anElem);
}

public void parse() {
  super.parse();
  toParse = toParse.trim();

  String params[] = toParse.split(",");

  String nparams[] = new String[params.length];
  for (int i = 0; i < params.length; i++) {
    try {
      nparams[i] =
ValueConverter.convertValue(params[i], clsused,
elem
        .getTargetClass(), elem);
    } catch (sp.XGGException e) {
      errorStr = "Array Syntax: On Argument " + (i +
1) + " should be " + "("
        + clsName + ")" + ": " + e.getError();
      return;
    }
  }

  StringBuffer resultant = new StringBuffer();

  resultant.append("new ");
  resultant.append(clsName);
  resultant.append("[]{");

  for (int i = 0; i < nparams.length; i++) {
    if (i != 0)
      resultant.append(", ");
    resultant.append(nparams[i]);
  }

  resultant.append('}');

  answer = resultant.toString();
}
}
```

```
*********************************
FILE: sp/format/attribute/DoubleArrayParser.java

package sp.format.attribute;

import sp.format.*;

/**
 * The parser for the <code>double[]</code> class.
 * Has format "double1,double2,..,double#".
 */
public class DoubleArrayParser extends
IntArrayParser {
protected DoubleArrayParser(String str,
XGGElement anElem) {
  super(str,anElem);
  clsName = "double";
  clsused = double.class;
}
}

*********************************
FILE: sp/format/attribute/WrapperParser.java

package sp.format.attribute;

import sp.format.*;
import sp.format.code.*;

/**
 * The parser for the Wrapper
(<code>Integer</code>,<code>ouble</code>,
etc.)
 * classes.
 */
public class WrapperParser extends AbstractParser
{
Class<?> cls = null;

protected WrapperParser(Class<?> aCls, String str,
XGGElement anElem) {
  super(str, anElem);
  cls = aCls;
}

public static Class<?> getWrappedClass(Class<?>
aCls) {
  if (aCls == Integer.class)
    return int.class;
  else if (aCls == Byte.class)
    return byte.class;
  else if (aCls == Float.class)
    return float.class;
  else if (aCls == Double.class)
    return double.class;
  else if (aCls == Long.class)
    return long.class;
  else if (aCls == Short.class)
    return short.class;
  else if (aCls == Boolean.class)
    return boolean.class;
  else if (aCls == Character.class)
    return char.class;

  return null;
}

public void parse() {
  super.parse();

  toParse = toParse.trim();

  Class<?> wcls = getWrappedClass(cls);

  if (wcls == null) {
    errorStr = "Primitive type Wrapper Syntax:
Unknown Parsing Error!";
    return;
  }

  String nval = null;
  try {
    nval = ValueConverter.convertValue(toParse,
wcls, cls, elem);
  } catch (sp.XGGException e) {
    errorStr =
sp.SPUtil.getShortName(cls.getName())
      + " Syntax: Value should be " + wcls.getName()
+ ": " + e.getError();
    return;
  }

  answer = ValueConverter.tryToObject(nval, wcls);

  if (answer == null) {
    errorStr =
sp.SPUtil.getShortName(cls.getName())
      + " Syntax: Unknown Parsing Error - Empty
Value!";
    return;
  }
}
}

*********************************
FILE: sp/gui/SPFrontEnd.xml
NOTE: compile this with the XGG Tool (this
program) to produce the file
sp/gui/SPFrontEndGUI.java

<JFrame title="XGG Front End" size="600,450"
 defaultCloseOperation="WindowConstants.HIDE_O
N_CLOSE"
 windowClosing="endProgram">

<Package path="sp.gui"/>
<JPanel layout="BorderLayout(2,2)"
border="EmptyBorder(5,5,5,5)">
  <JPanel constraint="BorderLayout.CENTER"
layout="BorderLayout()">
    <Box constraint="BorderLayout.NORTH"
axis="Y_AXIS">

      <JPanel layout="BorderLayout(5,5)">
        <JLabel text="File: "
constraint="BorderLayout.WEST"
          labelFor="{var jtfFile}"
displayedMnemonic="'F'"/>
        <JTextField id="jtfFile"
constraint="BorderLayout.CENTER" />
        <JButton text="Browse"
constraint="BorderLayout.EAST"
          mnemonic="'W'"
actionPerformed="doAction_Browse"/>
      </JPanel>

      <JPanel layout="GridBagLayout()"
gridbag.columnWeights=".1,.9">

        <JLabel text="Javac Arguments: "
constraint.gridwidth="RELATIVE"
        constraint.anchor="WEST" labelFor="{var
jtfCArg}" displayedMnemonic="'J'"
        constraint.insets="2,0,2,0"/>
        <JTextField id="jtfCArg" name="-J"
constraint.gridwidth="REMAINDER"
        constraint.fill="BOTH"
constraint.insets="2,0,2,0"/>
```

```
        <JLabel text="Java Arguments: "
constraint.gridwidth="RELATIVE"
         constraint.anchor="WEST" labelFor="{var
jtfIArg}" displayedMnemonic="'A'"
         constraint.insets="2,0,2,0"/>
        <JTextField id="jtfIArg" name="-JR"
constraint.gridwidth="REMAINDER"
         constraint.fill="BOTH"
constraint.insets="2,0,2,0"/>

        <JLabel text="Run Arguments: "
constraint.gridwidth="RELATIVE"
         constraint.anchor="WEST" labelFor="{var
jtfRArg}" displayedMnemonic="'R'"
         constraint.insets="2,0,2,0"/>
        <JTextField id="jtfRArg" name="-JA"
constraint.gridwidth="REMAINDER"
         constraint.fill="BOTH"
constraint.insets="2,0,2,0"/>

      </JPanel>

      <JPanel layout="GridLayout(1,2)">
        <JPanel layout="FlowLayout(LEFT,0,0)">
          <JLabel text="Bracing Style: "/>
          <JRadioButton actionCommand=" "
text="KNF" toolTipText="'{' before linefeed"
            group="bgBracetype" selected="true"/>
          <JRadioButton actionCommand="-B"
text="Allman" toolTipText="linefeed before '}'"
            group="bgBracetype"/>
        </JPanel>

        <JPanel layout="FlowLayout(LEFT,0,0)">
          <JLabel text="Indent: "/>
          <JRadioButton actionCommand="-i 2"
text="2" group="bgIndent"/>
          <JRadioButton actionCommand=" " text="4"
group="bgIndent" selected="true"/>
          <JRadioButton actionCommand="-i 8"
text="8" group="bgIndent"/>
        </JPanel>
      </JPanel>

      <JPanel layout="FlowLayout(LEFT,0,0)">
        <JLabel text="GUI File: "/>
        <JRadioButton actionCommand="-p" text="Do
nothing" group="bgGUI"/>
        <JRadioButton actionCommand="-pt"
text="Prototype" toolTipText="No event files"
group="bgGUI"/>
        <JRadioButton actionCommand=" "
text="Compile" group="bgGUI" selected="true"/>
        <JRadioButton actionCommand="-dc"
text="Compile and delete source"
group="bgGUI"/>
      </JPanel>

      <JPanel layout="FlowLayout(LEFT,0,0)">
        <JLabel text="Event File: "/>
        <JRadioButton actionCommand="-ne"
text="Do nothing" group="bgEvent"/>
        <JRadioButton actionCommand="-nce"
text="Create/Update source" group="bgEvent"/>
        <JRadioButton actionCommand=" "
text="Compile" group="bgEvent"
selected="true"/>
        <JRadioButton actionCommand="-r"
text="Compile and Run" group="bgEvent"/>
      </JPanel>

      <JPanel layout="GridLayout(2,2)">
        <JCheckBox id="cb_v" text="Verbose Mode"
mnemonic="'V'" name="-v"/>
        <JCheckBox id="cb_1_5" text="Generate 1.5
code" mnemonic="'G'" name="-1.5"/>
        <JCheckBox id="cb_nh" text="Code has
Comments" selected="true" mnemonic="'H'"
name="-nh"/>
        <JCheckBox id="cb_si" text="Code has
Indention in Structure"  mnemonic="'I'" name="-
si"/>
      </JPanel>
    </Box>

    <JPanel layout="BorderLayout()"
constraint="BorderLayout.CENTER">

      <JButton id="btnAct"
constraint="BorderLayout.NORTH"
cursor="HAND_CURSOR"
        text="Start!" font="Arial-BOLD-16"
actionPerformed="doAction_Start"
        mnemonic="'S'">
        <UI>

          <Component condition="!
getModel().isArmed()">
            <Paint type="fill" value="#EEEEFF"/>
          </Component>

          <Component condition="isMouseOver">
            <Paint type="fill" value="#AAAAFF"/>
          </Component>

          <Component
condition="getModel().isArmed()">
            <Paint type="fill"
              value="GradientPaint(0,0,white,getWidth(),
getHeight(),black,true)"/>
          </Component>

          <Component condition="!isEnabled()">
            <Paint type="fill"
              value="Color.black"/>
          </Component>

          <Component>
            <Stroke value="2"/>
            <RoundRectangle do="both" x="10" y="0"
width="getWidth()-20"
              height="getHeight()-1" contains="true"
archeight="20"
              arcwidth="20"/>
            <Text orientation="horizontal"
valueexp="getText()"
              x="getWidth()/2-
fontmetrics.stringWidth(getText())/2"
              y="2*getHeight()/3"/>
            <Stroke />
          </Component>

          <Border />
        </UI>
      </JButton>

      <JScrollPane
viewportBorder="TitledBorder(Console Output)"
horizontalScrollBarPolicy="ScrollPaneConstants.HO
RIZONTAL_SCROLLBAR_ALWAYS"
verticalScrollBarPolicy="ScrollPaneConstants.VERTI
CAL_SCROLLBAR_ALWAYS"
        constraint="BorderLayout.CENTER"
focusable="false">
        <JTextArea id="jtaConsole" editable="false"/>
      </JScrollPane>
```

```
    </JPanel>
  </JPanel>
</JPanel>
</JFrame>

*********************************
FILE: sp/gui/SPFrontEnd.java

package sp.gui;

import sp.compiler.*;
import java.io.*;
import java.awt.*;
import java.awt.dnd.*;
import java.awt.event.*;
import java.awt.datatransfer.*;
import java.util.*;
import javax.swing.*;
import javax.swing.text.*;

public class SPFrontEnd extends SPFrontEndGUI {
StreamGUIGobbler outGobbler = null, errGobbler =
null;
SPFileConfig fileconf = null;
JPopupMenu ccpMenu = new JPopupMenu();

public SPFrontEnd() {
  super();

  JMenuItem menuItem = null;

  menuItem = new JMenuItem(new
DefaultEditorKit.CutAction());
  menuItem.setText("Cut");
  menuItem.setMnemonic(KeyEvent.VK_T);
  ccpMenu.add(menuItem);
  menuItem = new JMenuItem(new
DefaultEditorKit.CopyAction());
  menuItem.setText("Copy");
  menuItem.setMnemonic(KeyEvent.VK_C);
  ccpMenu.add(menuItem);
  menuItem = new JMenuItem(new
DefaultEditorKit.PasteAction());
  menuItem.setText("Paste");
  menuItem.setMnemonic(KeyEvent.VK_P);
  ccpMenu.add(menuItem);

  jtfFile.setComponentPopupMenu(ccpMenu);
  jtfCArg.setComponentPopupMenu(ccpMenu);
  jtfIArg.setComponentPopupMenu(ccpMenu);
  jtfRArg.setComponentPopupMenu(ccpMenu);

  FileDropTarget fdt = new FileDropTarget();
  new DropTarget(jtfFile, fdt);
  new DropTarget(jtaConsole, fdt);
  new DropTarget(getContentPane(), fdt);

  PipedOutputStream pos = new
PipedOutputStream();
  outGobbler = new StreamGUIGobbler(pos,
jtaConsole);
  System.setOut(new PrintStream(pos));
  outGobbler.start();

  pos = new PipedOutputStream();
  errGobbler = new StreamGUIGobbler(pos,
jtaConsole);
  System.setErr(new PrintStream(pos));
  errGobbler.start();

  fileconf = SPFileConfig.load();
  if (fileconf == null)
    fileconf = new SPFileConfig();
  else {
    jtfCArg.setText(fileconf.cargStr);
    jtfIArg.setText(fileconf.iargStr);
    jtfRArg.setText(fileconf.rargStr);

    setOption(bgBracetype, fileconf.bstyle);
    setOption(bgIndent, fileconf.indent);

    cb_v.setSelected(fileconf.verbose);
    cb_1_5.setSelected(fileconf.mode1_5);
    cb_nh.setSelected(fileconf.comments);
    cb_si.setSelected(fileconf.sindent);
  }
}

public static void main(String[] args) {
  SwingUtilities.invokeLater(new Runnable() {
    public void run() {
      new SPFrontEnd();
    }
  });
}

public void setOption(ButtonGroup bgrp, String str)
{
  Enumeration<AbstractButton> enumer =
bgrp.getElements();
  while (enumer.hasMoreElements()) {
    AbstractButton abtn = enumer.nextElement();
    if (abtn.getActionCommand().equals(str)) {
      abtn.setSelected(true);
      break;
    }
  }
}

public void buttonGroupToArgs(LinkedList<String>
ll, ButtonGroup bgrp) {
  String str =
bgrp.getSelection().getActionCommand();

  String argstr[] = str.split(" ");
  for (int i = 0; i < argstr.length; i++) {
    if (argstr[i].trim().length() > 0)
      ll.add(argstr[i]);
  }
}

public void checkBoxToArgs(LinkedList<String> ll,
JCheckBox jc, boolean isarg) {
  if (jc.isSelected() == isarg)
    ll.add(jc.getName());
}

public void textFieldToArgs(LinkedList<String> ll,
JTextField jtf) {

}

public String[] createArguments() {
  LinkedList<String> ll = new LinkedList<String>();
  StringBuffer argsb = null;

  // ll.add("--debug");

  String tempstr = jtfFile.getText();
  if (tempstr != null)
    tempstr = tempstr.trim();

  if ((tempstr != null) && (tempstr.length() > 0)) {
    ll.add(tempstr);
  }

  checkBoxToArgs(ll, cb_v, true);
```

```
  checkBoxToArgs(ll, cb_1_5, true);
  checkBoxToArgs(ll, cb_nh, false);
  checkBoxToArgs(ll, cb_si, true);
  buttonGroupToArgs(ll, bgBracetype);
  buttonGroupToArgs(ll, bgIndent);
  buttonGroupToArgs(ll, bgGUI);
  buttonGroupToArgs(ll, bgEvent);

  String rargStr = jtfRArg.getText();
  if (rargStr != null) {
    rargStr = rargStr.trim();
    if (rargStr.length() > 0) {
      ll.add(jtfRArg.getName());
      ll.add(rargStr);
    }
  }

  textFieldToArgs(ll, jtflArg);

  if ((tempstr != null) && (tempstr.length() > 0)) {

    argsb = new StringBuffer("-cp \"");
    argsb.append(System.getProperty("java.class.pat
h"));

    File f = new File(tempstr);
    String fdir = f.getAbsolutePath();
    int sepindex = fdir.lastIndexOf(File.separator);
    if (sepindex > 0)
      fdir = fdir.substring(0, sepindex);
    else
      fdir = null;

    if (fdir != null) {
      argsb.append(File.pathSeparatorChar);
      argsb.append(fdir);
    }
    argsb.append("\" ");

    ll.add("-J");
    if (jtfCArg.getText() != null)
      ll.add(argsb.toString() + jtfCArg.getText());
    else
      ll.add(argsb.toString());

    ll.add("-JR");
    if (jtflArg.getText() != null)
      ll.add(argsb.toString() + jtflArg.getText());
    else
      ll.add(argsb.toString());
  }

  if (ll.size() == 0)
    ll.add("-help");

  String result[] = ll.toArray(new String[ll.size()]);
  return result;
}

public void doAction_Start(ActionEvent e) {
  new Thread(new FrontEndCompiler()).start();
}

public void doAction_Browse(ActionEvent e) {
  JFileChooser chooser = new JFileChooser();
  chooser.setFileFilter(new
javax.swing.filechooser.FileFilter() {
    public boolean accept(File f) {
      if (f.isDirectory()) {
        return true;
      }

      String fname = f.getName();
      int dotpos = fname.lastIndexOf('.');
      if (dotpos > 0) {
        String extension = fname.substring(dotpos,
fname.length());
        if ((!extension.equals(".xml")) && (!
extension.equals(".xgg")))
          return false;
      } else
        return false;
      return true;
    }

    public String getDescription() {
      return "XGG Files (*.xgg,*.xml)";
    }
  });
  int returnVal = chooser.showOpenDialog(this);
  if (returnVal == JFileChooser.APPROVE_OPTION) {
    jtfFile.setText(chooser.getSelectedFile().getAbsol
utePath());
  }

}

class FrontEndCompiler implements Runnable {
public void run() {
  btnAct.setEnabled(false);

  jtaConsole.setText("");
  System.out.println("Starting the XGG Compiler..");
  int exitcode = 0;
  try {
    new CompilerArgs(createArguments()).start();
  } catch (sp.ExitException err) {
    exitcode = err.getExitID();
  }

  try {
    Thread.sleep(200);
  } catch (InterruptedException e) {}

  System.out.println("Exited at code:" + exitcode);
  btnAct.setEnabled(true);
}
}

class FileDropTarget implements
DropTargetListener {
public void dragEnter(DropTargetDragEvent dtde)
{}

public void dragExit(DropTargetEvent dte) {}

public void dragOver(DropTargetDragEvent dtde)
{}

public void
dropActionChanged(DropTargetDragEvent dtde) {}

public void drop(DropTargetDropEvent dtde) {
  try {
    Transferable tr = dtde.getTransferable();
    DataFlavor[] flavors =
tr.getTransferDataFlavors();

    for (int i = 0; i < flavors.length; i++) {
      if (flavors[i].isFlavorJavaFileListType()) {
        dtde.acceptDrop(DnDConstants.ACTION_COPY_
OR_MOVE);

        java.util.List list = (java.util.List)
tr.getTransferData(flavors[i]);
        jtfFile.setText(list.get(0).toString());
      }
```

```
    }
  } catch (Exception e) {
    e.printStackTrace();
    dtde.rejectDrop();
  }
}
}

public void endProgram(WindowEvent e) {
  fileconf.cargStr = jtfCArg.getText();
  fileconf.iargStr = jtfIArg.getText();
  fileconf.rargStr = jtfRArg.getText();
  fileconf.bstyle =
bgBracetype.getSelection().getActionCommand();
  fileconf.indent =
bgIndent.getSelection().getActionCommand();
  fileconf.verbose = cb_v.isSelected();
  fileconf.mode1_5 = cb_1_5.isSelected();
  fileconf.comments = cb_nh.isSelected();
  fileconf.sindent = cb_si.isSelected();

  fileconf.save();
  System.exit(0);
}
}

**********
FILE: sp/gui/StreamGUIGobbler.java

package sp.gui;

import java.io.*;
import javax.swing.*;
import sp.compiler.*;

/**
 * The class used to get the out and err streams of
the console and redirects
 * them to a textarea.
 */
public class StreamGUIGobbler extends Thread {
private PipedOutputStream pos;
private JTextArea jtaOutput;
private Exception err = null;

public StreamGUIGobbler(PipedOutputStream aPos,
JTextArea aTextArea) {
  pos = aPos;
  jtaOutput = aTextArea;
}

public Exception getException() {
  return err;
}

public void run() {
  try {
    PipedInputStream is = new
PipedInputStream(pos);
    BufferedReader br = new BufferedReader(new
InputStreamReader(is));

    String line = null;
    while ((line = br.readLine()) != null) {
      jtaOutput.append(line + "\n");
      jtaOutput.setCaretPosition(jtaOutput.getText().l
ength());

      while (!br.ready()) {
        try {
          Thread.sleep(10);
        } catch (InterruptedException e) {}
      }
    }
  } catch (IOException e) {
    e.printStackTrace();
    err = e;
  }
}
}

**********
FILE: sp/gui/SPFileConfig.java

package sp.gui;

import java.io.*;

package sp.gui;

import java.io.*;

public class SPFileConfig implements Serializable {
static final long serialVersionUID = 'f' * 'i' * 'l' * 'e' *
'c' * 'o' * 'n'* 'f';

public String cargStr = "";
public String iargStr = "";
public String rargStr = "";
public String bstyle = " ";
public String indent = " ";
public boolean verbose = false;
public boolean mode1_5 = false;
public boolean comments = true;
public boolean sindent = true;

public void save() {
  try {
    ObjectOutputStream oos = new
ObjectOutputStream(new FileOutputStream(
      "xgggui.dat"));

    oos.writeObject(this);
  } catch (FileNotFoundException e) {} catch
(IOException e) {}
}

public static SPFileConfig load() {
  try {
    ObjectInputStream ois = new
ObjectInputStream(new FileInputStream(
      "xgggui.dat"));

    SPFileConfig result = (SPFileConfig)
ois.readObject();

    return result;
  } catch (FileNotFoundException e) {} catch
(ClassNotFoundException e) {} catch (IOException
e) {}

  return null;
}
}

**********
FILE: Java.g
NOTE: compile the  file Java.g w/ the ANTLR
(http://www.antlr.org/) tool to be able to create the
files sp/compiler/JavaParser.java and
sp/compiler/JavaLexer.java

grammar Java;
options {k=2; backtrack=true; memoize=true;}

@header {
package sp.compiler;
}
```

```
@lexer::header {
package sp.compiler;
}

@lexer::members {
protected boolean enumIsKeyword = true;
}

@parser::members {
public String ErrMsg = null;
public String packageName = null;
public String MainName = "";
public java.util.Stack CurClass = new
java.util.Stack();
public java.util.LinkedList LLmethods = new
java.util.LinkedList();
public java.util.LinkedList LLconstructs = new
java.util.LinkedList();
public int insertLine = 0;
public int insertColumn = 0;
public int insertCLine = -1;
public int insertCColumn = 0;
public int packageLine = -1;
public int packageColumn = 0;
public int packageLineEnd = -1;
public int packageColumnEnd = 0;

public void emitErrorMessage(String msg)
{
  ErrMsg = msg;
}
}
```

NOTE: #the rest are not included since it is supplied by the ANTLR

****
FILE: sp/xggplugin/Activator.java

```
package sp.xggplugin;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

/**
 * The activator class controls the plug-in life cycle
 */
public class Activator extends AbstractUIPlugin {

// The plug-in ID
public static final String PLUGIN_ID =
"sp.XGGPlugin";

// The shared instance
private static Activator plugin;

/**
 * The constructor
 */
public Activator() {}

public void start(BundleContext context) throws
Exception {
  super.start(context);
  plugin = this;
}

public void stop(BundleContext context) throws
Exception {
  plugin = null;
  super.stop(context);
}

public static Activator getDefault() {
  return plugin;
}

public static ImageDescriptor
getImageDescriptor(String path) {
  return imageDescriptorFromPlugin(PLUGIN_ID,
path);
}
}
```

****
FILE: sp/xggplugin/editors/ColorManager.java

```
package package sp.xggplugin.editors;

import java.util.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.widgets.Display;

public class ColorManager {

protected Map<RGB, Color> fColorTable = new
HashMap<RGB, Color>(10);

public void dispose() {
  Iterator<Color> e = fColorTable.values().iterator();
  while (e.hasNext())
    e.next().dispose();
}

public Color getColor(RGB rgb) {
  Color color = (Color) fColorTable.get(rgb);
  if (color == null) {
    color = new Color(Display.getCurrent(), rgb);
    fColorTable.put(rgb, color);
  }
  return color;
}
}
```

****
FILE: sp/xggplugin/editors/EclipseClassLoader.java

```
package sp.xggplugin.editors;

import java.io.*;
import org.eclipse.core.runtime.*;

public class EclipseClassLoader extends
ClassLoader {
private IPath root;

public EclipseClassLoader(IPath rootDir) {
  root = Platform.getLocation().append(rootDir);
}

@SuppressWarnings("unchecked")
protected Class loadClass(String name, boolean
resolve)
  throws ClassNotFoundException {
  Class c = findLoadedClass(name);
  if (c == null) {
    try {
      c = findSystemClass(name);
    } catch (Exception e) {}
  }

  if (c == null) {
    String filename = name.replace('.',
File.separatorChar) + ".class";

    try {
```

```
      byte data[] = loadClassData(filename);
      c = defineClass(name, data, 0, data.length);
      if (c == null)
        throw new ClassNotFoundException(name);
    } catch (IOException e) {
      throw new ClassNotFoundException("Error
reading file: " + filename);
    }
  }

  if (resolve)
    resolveClass(c);

  return c;
}

private byte[] loadClassData(String filename)
throws IOException {
  IPath npath = root.append(filename);
  File f = npath.toFile();

  int size = (int) f.length();

  byte buff[] = new byte[size];
  FileInputStream fis = new FileInputStream(f);
  DataInputStream dis = new DataInputStream(fis);

  dis.readFully(buff);

  dis.close();

  return buff;
}
}
```

****

FILE: sp/xggplugin/editors/IXMLColorConstants.java

```
package sp.xggplugin.editors;

import org.eclipse.swt.graphics.RGB;

public interface IXMLColorConstants {
RGB XML_COMMENT = new RGB(128, 0, 0);
RGB PROC_INSTR = new RGB(128, 128, 128);
RGB STRING = new RGB(0, 128, 0);
RGB DEFAULT = new RGB(0, 0, 0);
RGB TAG = new RGB(0, 0, 128);
}
```

****

FILE: sp/xggplugin/editors/MultiPageEditorContributor.java

```
package sp.xggplugin.editors;

import org.eclipse.jface.action.*;
import org.eclipse.ui.*;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.ui.ide.IDEActionFactory;
import
org.eclipse.ui.part.MultiPageEditorActionBarContrib
utor;
import org.eclipse.ui.texteditor.ITextEditor;
import
org.eclipse.ui.texteditor.ITextEditorActionConstants;

/**
 * Manages the installation/deinstallation of global
actions for multi-page
 * editors. Responsible for the redirection of global
actions to the active
 * editor. Multi-page contributor replaces the
contributors for the individual
 * editors in the multi-page editor.
 */
public class MultiPageEditorContributor extends
  MultiPageEditorActionBarContributor {
private IEditorPart activeEditorPart;
private Action compileAction;
private Action commentAction;
private Action protoAction;

/**
 * Creates a multi-page contributor.
 */
public MultiPageEditorContributor() {
  super();
  createActions();
}

protected IAction getAction(ITextEditor editor,
String actionID) {
  return (editor == null ? null :
editor.getAction(actionID));
}

public void setActivePage(IEditorPart part) {
  if (activeEditorPart == part)
    return;

  activeEditorPart = part;

  IActionBars actionBars = getActionBars();
  if (actionBars != null) {

    ITextEditor editor = (part instanceof
ITextEditor) ? (ITextEditor) part
      : null;

    actionBars.setGlobalActionHandler(ActionFactory.
DELETE.getId(), getAction(
      editor, ITextEditorActionConstants.DELETE));
    actionBars.setGlobalActionHandler(ActionFactory.
UNDO.getId(), getAction(
      editor, ITextEditorActionConstants.UNDO));
    actionBars.setGlobalActionHandler(ActionFactory.
REDO.getId(), getAction(
      editor, ITextEditorActionConstants.REDO));
    actionBars.setGlobalActionHandler(ActionFactory.
CUT.getId(), getAction(
      editor, ITextEditorActionConstants.CUT));
    actionBars.setGlobalActionHandler(ActionFactory.
COPY.getId(), getAction(
      editor, ITextEditorActionConstants.COPY));
    actionBars.setGlobalActionHandler(ActionFactory.
PASTE.getId(), getAction(
      editor, ITextEditorActionConstants.PASTE));
    actionBars.setGlobalActionHandler(ActionFactory.
SELECT_ALL.getId(),
      getAction(editor,
ITextEditorActionConstants.SELECT_ALL));
    actionBars.setGlobalActionHandler(ActionFactory.
FIND.getId(), getAction(
      editor, ITextEditorActionConstants.FIND));
    actionBars.setGlobalActionHandler(IDEActionFact
ory.BOOKMARK.getId(),
      getAction(editor,
IDEActionFactory.BOOKMARK.getId()));
    actionBars.updateActionBars();
  }
}

private void createActions() {
  compileAction = new Action() {
    public void run() {
```

```
      if (activeEditorPart instanceof
XGGMultiPageEditor)
        ((XGGMultiPageEditor)
activeEditorPart).doXGGCompile(true);
      else if (activeEditorPart instanceof XGGEditor)
        ((XGGEditor)
activeEditorPart).doXGGCompile(true);
    }
  };
  compileAction.setText("Compile XGG");
  compileAction.setToolTipText("Compiles the XGG
File to Java Swing code");
  compileAction.setImageDescriptor(PluginUtils
    .getImageDescriptor("exportapp_wiz.gif"));

  commentAction = new Action("Comments",
Action.AS_CHECK_BOX) {
    public void run() {
      if (activeEditorPart instanceof
XGGMultiPageEditor)
        ((XGGMultiPageEditor)
activeEditorPart).setComments(isChecked());
      else if (activeEditorPart instanceof XGGEditor)
        ((XGGEditor)
activeEditorPart).setComments(isChecked());
    }
  };
  commentAction.setChecked(true);
  commentAction.setToolTipText("Comments are
added to the generated code");
  commentAction.setImageDescriptor(PluginUtils
    .getImageDescriptor("comment_edit.gif"));

  protoAction = new Action("Prototype",
Action.AS_CHECK_BOX) {
    public void run() {
      if (activeEditorPart instanceof
XGGMultiPageEditor)
        ((XGGMultiPageEditor)
activeEditorPart).setPrototype(isChecked());
      else if (activeEditorPart instanceof XGGEditor)
        ((XGGEditor)
activeEditorPart).setPrototype(isChecked());
    }
  };
  protoAction
    .setToolTipText("Event file is not generated and
GUI file has main method");
  protoAction.setImageDescriptor(PluginUtils
    .getImageDescriptor("templateprop_co.gif"));
}

public void contributeToMenu(IMenuManager
manager) {
  IMenuManager menu = new
MenuManager("&XGG");
  manager.prependToGroup(IWorkbenchActionConst
ants.MB_ADDITIONS, menu);
  menu.add(compileAction);
  menu.add(new Separator());
  menu.add(commentAction);
  menu.add(protoAction);
}

public void contributeToToolBar(IToolBarManager
manager) {
  manager.add(new Separator());
  manager.add(compileAction);
  manager.add(commentAction);
  manager.add(protoAction);
}
}

*********************************
```

FILE:
sp/xggplugin/editors/NonRuleBasedDamagerRepaire
r.java

```
package sp.xggplugin.editors;

import org.eclipse.jface.text.*;
import
org.eclipse.jface.text.presentation.IPresentationDa
mager;
import
org.eclipse.jface.text.presentation.IPresentationRep
airer;
import org.eclipse.core.runtime.Assert;
import org.eclipse.swt.custom.StyleRange;

public class NonRuleBasedDamagerRepairer
implements IPresentationDamager,
  IPresentationRepairer {

/** The document this object works on */
protected IDocument fDocument;
/** The default text attribute if non is returned as
data by the current token */
protected TextAttribute fDefaultTextAttribute;

/**
 * Constructor for NonRuleBasedDamagerRepairer.
 */
public
NonRuleBasedDamagerRepairer(TextAttribute
defaultTextAttribute) {
  Assert.isNotNull(defaultTextAttribute);

  fDefaultTextAttribute = defaultTextAttribute;
}

/**
 * @see
IPresentationRepairer#setDocument(IDocument)
 */
public void setDocument(IDocument document) {
  fDocument = document;
}

/**
 * Returns the end offset of the line that contains
the specified offset or if
 * the offset is inside a line delimiter, the end offset
of the next line.
 *
 * @param offset
 *          the offset whose line end offset must be
computed
 * @return the line end offset for the given offset
 * @exception BadLocationException
 *              if offset is invalid in the current document
 */
protected int endOfLineOf(int offset) throws
BadLocationException {

  IRegion info =
fDocument.getLineInformationOfOffset(offset);
  if (offset <= info.getOffset() + info.getLength())
    return info.getOffset() + info.getLength();

  int line = fDocument.getLineOfOffset(offset);
  try {
    info = fDocument.getLineInformation(line + 1);
    return info.getOffset() + info.getLength();
  } catch (BadLocationException x) {
    return fDocument.getLength();
  }
}
```

```
/**
 * @see
IPresentationDamager#getDamageRegion(ITypedR
egion, DocumentEvent,
 *      boolean)
 */
public IRegion getDamageRegion(ITypedRegion
partition, DocumentEvent event,
  boolean documentPartitioningChanged) {
  if (!documentPartitioningChanged) {
    try {

      IRegion info =
fDocument.getLineInformationOfOffset(event.getOff
set());
      int start = Math.max(partition.getOffset(),
info.getOffset());

      int end = event.getOffset()
        + (event.getText() == null ? event.getLength()
: event.getText()
          .length());

      if (info.getOffset() <= end && end <=
info.getOffset() + info.getLength()) {
        // optimize the case of the same line
        end = info.getOffset() + info.getLength();
      } else
        end = endOfLineOf(end);

      end = Math.min(partition.getOffset() +
partition.getLength(), end);
      return new Region(start, end - start);

    } catch (BadLocationException x) {}
  }

  return partition;
}

/**
 * @see
IPresentationRepairer#createPresentation(TextPres
entation, ITypedRegion)
 */
public void createPresentation(TextPresentation
presentation,
  ITypedRegion region) {
  addRange(presentation, region.getOffset(),
region.getLength(),
    fDefaultTextAttribute);
}

/**
 * Adds style information to the given text
presentation.
 *
 * @param presentation
 *          the text presentation to be extended
 * @param offset
 *          the offset of the range to be styled
 * @param length
 *          the length of the range to be styled
 * @param attr
 *          the attribute describing the style of the
range to be styled
 */
protected void addRange(TextPresentation
presentation, int offset, int length,
  TextAttribute attr) {
  if (attr != null)
    presentation.addStyleRange(new
StyleRange(offset, length, attr
        .getForeground(), attr.getBackground(),
attr.getStyle()));
}
}

**********
FILE: sp/xggplugin/editors/PluginUtils.java

package sp.xggplugin.editors;

import java.net.MalformedURLException;
import java.net.URL;

import org.eclipse.jface.resource.ImageDescriptor;

import sp.xggplugin.Activator;

public class PluginUtils {
  public static ImageDescriptor
getImageDescriptor(String name)  {
    String iconPath = "icons/";
    try  {
      URL installURL =
Activator.getDefault().getBundle().getEntry("/");
      URL url = new URL(installURL, iconPath +
name);
      return ImageDescriptor.createFromURL(url);
    } catch (MalformedURLException e)  {
      return
ImageDescriptor.getMissingImageDescriptor();
    }
  }
}

**********
FILE: sp/xggplugin/editors/TagRule.java

package sp.xggplugin.editors;

import org.eclipse.jface.text.rules.*;

public class TagRule extends MultiLineRule {

public TagRule(IToken token) {
  super("<", ">", token);
}

protected boolean
sequenceDetected(ICharacterScanner scanner,
char[] sequence,
  boolean eofAllowed) {
  int c = scanner.read();
  if (sequence[0] == '<') {
    if (c == '?') {
      // processing instruction - abort
      scanner.unread();
      return false;
    }
    if (c == '!') {
      scanner.unread();
      // comment - abort
      return false;
    }
  } else if (sequence[0] == '>') {
    scanner.unread();
  }
  return super.sequenceDetected(scanner,
sequence, eofAllowed);
}
}

**********
FILE: sp/xggplugin/editors/XGGEclipseFormat.java
```

```
package sp.xggplugin.editors;

import sp.compiler.*;
import sp.format.*;
import java.io.*;

import org.eclipse.jdt.core.*;
import org.eclipse.core.resources.*;

/**
 * Extension of the XGGFormat class to support the
file system of Eclipse
 */
public class XGGEclipseFormat extends XGGFormat
{
IJavaProject javaproj = null;
EclipseClassLoader ecl = null;

public XGGEclipseFormat(CompilerOpts opts, File
file, IProject aProject) {
  super(opts, file);
  javaproj = JavaCore.create(aProject);
  try {
    ecl = new
EclipseClassLoader(javaproj.getOutputLocation());
  } catch (JavaModelException e) {}
}

public Class<?> checkElement(String str) {
  Class<?> result = super.checkElement(str);
  if (result != null)
    return result;

  if (ecl != null)
    return this.checkElementViaClassLoader(str, ecl);
  return null;}
}

**********
FILE: sp/xggplugin/editors/XGGEclipseParser.java

package sp.xggplugin.editors;

import java.io.*;

import sp.compiler.*;
import sp.format.*;
import org.eclipse.core.runtime.*;
import org.eclipse.core.resources.*;

/**
 * Extension of the XGGParser class to support the
file system of Eclipse
 */
public class XGGEclipseParser extends XGGParser {
IFile mainFile = null;
IPath curPath = null;

public XGGEclipseParser(XGGFormat anXGG, String
str, boolean isVerbose,
  IFile aFile) {
  super(anXGG, str, isVerbose);
  mainFile = aFile;

  IPath mainPath =
mainFile.getFullPath().removeFileExtension();
  curPath = mainPath.removeLastSegments(1);
  curPath = curPath.removeFirstSegments(1);
}

protected InputStream getStyleFile(String fname)
throws FileNotFoundException,
  IOException {
  File stylefile = new File(fname);
  if ((stylefile.isAbsolute()) || (fname.charAt(0) ==
File.separatorChar))
    super.getStyleFile(fname);

  IPath stylePath = curPath.append(fname);

  IFile styleIFile =
mainFile.getProject().getFile(stylePath);

  try {
    return styleIFile.getContents();
  } catch (CoreException e) {
    throw new IOException(e.getMessage());
  }
}
}

**********
FILE: sp/xggplugin/editors/XGGEditor.java

package sp.xggplugin.editors;

import org.eclipse.core.resources.*;
import org.eclipse.core.filebuffers.*;
import org.eclipse.ui.editors.text.TextEditor;
import
org.eclipse.ui.views.contentoutline.IContentOutlineP
age;

import sp.format.*;
import sp.compiler.*;

import org.xml.sax.*;
import org.eclipse.core.runtime.*;

import java.io.*;
import java.util.*;

import org.antlr.runtime.*;
import javax.xml.parsers.*;

public class XGGEditor extends TextEditor {
private SAXParserFactory factorySAX = null;
private SAXParser sax = null;

private XGGFormat xggMain = null;
private CompilerOpts cOpts = new CompilerOpts();

private ColorManager colorManager;
private XMLEditorOutlinePage outline = null;

private String xggName = "";
private boolean dirtyCompile = true;

public XGGEditor() {
  super();
  sp.SPUtil.CONSOLE = false;
  cOpts.setMode1_5(true);
  colorManager = new ColorManager();
  setSourceViewerConfiguration(new
XMLConfiguration(colorManager));
  setDocumentProvider(new
XMLDocumentProvider());

  try {
    factorySAX = SAXParserFactory.newInstance();
    sax = factorySAX.newSAXParser();
  } catch (FactoryConfigurationError e) {} catch
(Exception e) {}
}

public boolean isDirtyCompile() {
  return dirtyCompile;
}
```

```
public void setComments(boolean b) {
  cOpts.setComments(b);
}

public void setStructIndent(boolean b) {
  cOpts.setStructIndent(b);
}

public void setPrototype(boolean b) {
  cOpts.setPrototype(b);
}

public boolean doXGGParse() {
  boolean result = true;
  InputStream isXML = null;
  IFile file = (IFile)
getEditorInput().getAdapter(IFile.class);
  try {
    isXML = file.getContents();

    file.deleteMarkers(IMarker.PROBLEM, true,
IResource.DEPTH_INFINITE);

    dirtyCompile = true;
    xggMain = new XGGEclipseFormat(cOpts,
file.getFullPath().makeAbsolute()
      .toFile(), file.getProject());

    XGGEclipseParser xparser = new
XGGEclipseParser(xggMain, file.getName(),
      cOpts.isVerbose(), file);

    sax.setProperty("http://xml.org/sax/properties/lex
ical-handler", xparser);
    sax.parse(isXML, xparser);

    String str = file.getName();
    str = str.substring(0, str.indexOf('.'));

    xggMain.setName(str + "GUI");
    xggMain.setRootName(file.getName());
    if (outline != null)
      outline.setXGGTree(xggMain);

    dirtyCompile = false;
  } catch (CoreException e) {
    reportError(file, IMarker.SEVERITY_ERROR, 0,
"Core Error: "
      + e.getMessage());
    result = true;
  } catch (IOException e) {
    reportError(file, IMarker.SEVERITY_ERROR, 0, "IO
Error: " + e.getMessage());
    result = true;
  } catch (SAXException e) {
    if (xggMain != null) {
      LinkedList<CompilerMessage> ll =
xggMain.getMessages();

      CompilerMessage cm = null;
      try {
        cm = ll.removeLast();
      } catch (NoSuchElementException e2) {}

      if (cm != null) {
        reportError(file, IMarker.SEVERITY_ERROR,
cm.getRow(), cm.getMessage());
      } else
        reportError(file, IMarker.SEVERITY_ERROR, 0,
"SAX Error: "
          + e.getMessage());
    } else
      reportError(file, IMarker.SEVERITY_ERROR, 0,
"SAX Error: "
        + e.getMessage());
    result = true;
  } finally {
    if (isXML != null) {
      try {
        isXML.close();
      } catch (IOException e) {}
      isXML = null;
    }
  }

  outputWarnings(file);

  return result;
}

private void outputWarnings(IFile file) {
  if (xggMain != null) {
    LinkedList<CompilerMessage> ll =
xggMain.getMessages();

    CompilerMessage cm;
    try {
      while ((cm = ll.removeFirst()) != null) {
        reportError(file, IMarker.SEVERITY_WARNING,
cm.getRow(), cm
          .getMessage());
      }
    } catch (NoSuchElementException e) {}
  }
}

public void doXGGCompile(boolean doOutput) {
  if (isDirty()) {
    doSave(null);
    dirtyCompile = true;
  }

  if (dirtyCompile) {
    if (!doXGGParse())
      return;
  }

  dirtyCompile = true;
  PrintWriter pwFile = null;
  BufferedReader EvtRead = null;
  EventFileMaker evm = null;
  IFile mainFile = (IFile)
getEditorInput().getAdapter(IFile.class);
  try {
    cOpts.setMode1_5(true);
    String filestr = xggMain.writeFile();

    IPath basePath =
mainFile.getFullPath().removeFileExtension();
    xggName = basePath.lastSegment();
    basePath = basePath.removeLastSegments(1);

    IPath root =
Platform.getLocation().append(basePath);
    IPath npath = root.append(xggName + "GUI");
    npath = npath.addFileExtension("java");

    File fOutFile = npath.toFile();

    pwFile = new PrintWriter(fOutFile);
    pwFile.println("//Created by XGG Tool");
    pwFile.println("//Roy Gian S.P Balderrama");
    pwFile.println();
    pwFile.print(filestr);
    pwFile.close();
```

```
    npath = basePath.append(xggName + "GUI");
    npath = npath.addFileExtension("java");

    IFile nfile =
FileBuffers.getWorkspaceFileAtLocation(npath);
    if (nfile != null)
      nfile.refreshLocal(1, null);
    else {
      IFolder nfold = mainFile.getProject().getFolder(
        basePath.removeFirstSegments(1));
      nfold.refreshLocal(2, null);
    }

    if (!doOutput)
      return;

    root = Platform.getLocation().append(basePath);
    npath = root.append(xggName);
    npath = npath.addFileExtension("java");

    File fEvtFile = npath.toFile();
    IFile file = (IFile)
getEditorInput().getAdapter(IFile.class);

    try {
      EvtRead = new BufferedReader(new
FileReader(fEvtFile));
    } catch (FileNotFoundException e) {
      EvtRead = null;
    }

    String sEvtFile = file.getName().substring(0,
file.getName().indexOf('.'));

    String rawEvtFile =
root.append(xggName).toString();

    evm = new EventFileMaker(sEvtFile, rawEvtFile,
xggMain, cOpts, EvtRead);

    filestr = evm.writeFile();
    if (EvtRead != null) {
      EvtRead.close();
      EvtRead = null;
    }

    if (evm.isDirty()) {
      pwFile = new PrintWriter(fEvtFile);
      pwFile.print(filestr);
      pwFile.close();

      nfile =
FileBuffers.getWorkspaceFileAtLocation(npath);
      if (nfile != null)
        nfile.refreshLocal(1, null);
      else {
        IFolder nfold = mainFile.getProject().getFolder(
          basePath.removeFirstSegments(1));
        nfold.refreshLocal(2, null);
      }
    }
  } catch (CoreException e) {
    reportError(mainFile, IMarker.SEVERITY_ERROR,
0, "Core Error: "
      + e.getMessage());
  } catch (FileNotFoundException e) {
    reportError(mainFile, IMarker.SEVERITY_ERROR,
0, "FileNotFound Error: "
      + e.getMessage());
  } catch (XGGCompileException e) {
    CompilerMessage cm = e.getCMessage();
    if (cm != null)
      reportError(mainFile, IMarker.SEVERITY_ERROR,
cm.getRow(), cm
          .getMessage());
    else
      reportError(mainFile, IMarker.SEVERITY_ERROR,
0, "Compiler Error: "
          + e.getMessage());
  } catch (RecognitionException e) {
    String errStr = null;
    if (evm != null)
      errStr = evm.getErrorMessage();

    if (errStr != null)
      reportError(mainFile, IMarker.SEVERITY_ERROR,
0,
          "Error in updating event file: " + errStr);
    else
      reportError(mainFile, IMarker.SEVERITY_ERROR,
0,
          "Error in updating event file!");
  } catch (IOException e) {
    reportError(mainFile, IMarker.SEVERITY_ERROR,
0, "IO Error: "
        + e.getMessage());
  }

  outputWarnings(mainFile);
}

public void reportError(IResource resource, int
severity, int line, String msg) {
  try {
    IMarker m =
resource.createMarker(IMarker.PROBLEM);
    if (line > 0)
      m.setAttribute(IMarker.LINE_NUMBER, line);
    m.setAttribute(IMarker.MESSAGE, msg);
    m.setAttribute(IMarker.PRIORITY,
IMarker.PRIORITY_HIGH);
    m.setAttribute(IMarker.SEVERITY, severity);
  } catch (CoreException e) {}
}

public void doSave(IProgressMonitor
progressMonitor) {
  super.doSave(progressMonitor);
  doXGGParse();
}

protected IContentOutlinePage getOutlinePage() {
  if ((outline == null) || (outline.isDisposed())) {
    outline = new XMLEditorOutlinePage();
    doXGGParse();
  }

  return outline;
}

@SuppressWarnings("unchecked")
public Object getAdapter(Class required) {
  if (IContentOutlinePage.class.equals(required))
    return getOutlinePage();
  return super.getAdapter(required);
}

public void dispose() {
  colorManager.dispose();
  super.dispose();
}
}
```

****
FILE: sp/xggplugin/editors/XGGMultiPageEditor.java

```
package sp.xggplugin.editors;
```

```
import org.eclipse.core.resources.*;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.jface.dialogs.ErrorDialog;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.*;
import org.eclipse.ui.part.FileEditorInput;
import org.eclipse.ui.part.MultiPageEditorPart;
import org.eclipse.ui.ide.IDE;

import org.eclipse.jdt.internal.ui.javaeditor.*;
import org.eclipse.ui.views.contentoutline.*;
import org.eclipse.core.resources.*;
import org.eclipse.core.runtime.*;
import java.io.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;
import org.eclipse.ui.part.*;

@SuppressWarnings("restriction")
class JavaUnitEditor extends CompilationUnitEditor
{}

public class XGGMultiPageEditor extends
MultiPageEditorPart implements
  IResourceChangeListener {
private String xggName = "";

private XGGEditor xggEditor = null;
private IEditorPart guiEditor = null;
private FileEditorInput guiInput = null;
private XGGOutlinePage outline = null;

private IPath basePath = null;
private IPath guiPath = null;
private IFile mainFile = null;

public XGGMultiPageEditor() {
  super();
  ResourcesPlugin.getWorkspace().addResourceCha
ngeListener(this);
}

public IEditorPart getEditorAt(int index) {
  return getEditor(index);
}

public int getCurrentPage() {
  return getActivePage();
}

public IPage getActiveIPage() {
  return getOutlinePageFor(getActiveEditor());
}

private void createPage0() {
  try {
    mainFile = (IFile)
getEditorInput().getAdapter(IFile.class);

    IPath curpath =
mainFile.getFullPath().removeFileExtension();
    xggName = curpath.lastSegment();
    basePath = curpath.removeLastSegments(1);
    basePath = basePath.removeFirstSegments(1);

    xggEditor = new XGGEditor();
    int index = addPage(xggEditor, getEditorInput());
    setPageText(index, "XML File");
    setPartName(xggEditor.getTitle());
  } catch (PartInitException e) {
    ErrorDialog.openError(getSite().getShell(),
      "Error creating nested text editor", null,
e.getStatus());
  }
}

private void initGUIEditor() throws PartInitException
{
  guiPath = basePath.append(xggName + "GUI");
  guiPath = guiPath.addFileExtension("java");

  IFile newfile =
mainFile.getProject().getFile(guiPath);

  guiInput = new FileEditorInput(newfile);

  if (newfile.getLocation() == null) {
    guiEditor = null;
    return;
  }

  File f = newfile.getLocation().toFile();
  if (f.exists())
    guiEditor = new JavaUnitEditor();
  else
    guiEditor = null;
}

private void createPage1() {
  int index = 1;
  if (getPageCount() > index)
    removePage(index);
  try {
    initGUIEditor();
    if (guiEditor != null)
      addPage(index, guiEditor, guiInput);
    else {
      Label lbl = new Label(this.getContainer(),
SWT.HORIZONTAL);
      lbl.setText("Unable to open " + guiPath);
      addPage(index, lbl);
    }
    setPageText(index, "UI Source Code");
  } catch (PartInitException e) {
    ErrorDialog.openError(getSite().getShell(),
      "Error creating nested text editor", null,
e.getStatus());
  }
}

protected void createPages() {
  createPage0();
  createPage1();
}

public void doSave(IProgressMonitor monitor) {
  getEditor(0).doSave(monitor);
  getEditor(1).doSave(monitor);
}

public void doXGGCompile(boolean doOutput) {
  if (xggEditor != null)
    xggEditor.doXGGCompile(doOutput);
}

public void setComments(boolean b) {
  if (xggEditor != null)
    xggEditor.setComments(b);
}

public void setStructIndent(boolean b) {
  if (xggEditor != null)
    xggEditor.setStructIndent(b);
}

public void setPrototype(boolean b) {
  if (xggEditor != null)
    xggEditor.setPrototype(b);
```

```
}

/**
 * Saves the multi-page editor's document as
another file. Also updates the text
 * for page 0's tab, and updates this multi-page
editor's input to correspond to
 * the nested editor's.
 */
public void doSaveAs() {
  IEditorPart editor = getEditor(0);
  editor.doSaveAs();
  setPageText(0, editor.getTitle());
  setPartName(xggEditor.getTitle());
  setInput(editor.getEditorInput());
  createPage1();
  xggEditor.doXGGParse();
}

public void gotoMarker(IMarker marker) {
  setActivePage(0);
  IDE.gotoMarker(getEditor(0), marker);
}

/**
 * The <code>MultiPageEditorExample</code>
implementation of this method
 * checks that the input is an instance of
<code>IFileEditorInput</code>.
 */
public void init(IEditorSite site, IEditorInput
editorInput)
  throws PartInitException {
  if (!(editorInput instanceof IFileEditorInput))
    throw new PartInitException("Invalid Input: Must
be IFileEditorInput");
  super.init(site, editorInput);
}

public boolean isSaveAsAllowed() {
  return true;
}

protected void pageChange(int newPageIndex) {
  super.pageChange(newPageIndex);

  IEditorPart editor = getEditor(newPageIndex);

  if ((editor != null) && (outline != null) && (!
outline.isDisposed()))
    outline.showOutline(getActiveEditor());

  if (newPageIndex == 0) {
    if ((xggEditor.isDirtyCompile()) ||
(xggEditor.isDirty()))
      xggEditor.doXGGParse();
  } else if (newPageIndex == 1) {
    if (guiEditor == null) {
      try {
        initGUIEditor();
        if (guiEditor != null) {
          createPage1();
          setActiveEditor(getEditor(1));
        }
      } catch (PartInitException e) {}
    } else {
      guiPath = basePath.append(xggName + "GUI");
      guiPath = guiPath.addFileExtension("java");

      try {
        IFile newfile =
mainFile.getProject().getFile(guiPath);
        newfile.refreshLocal(1, null);
      } catch (CoreException e) {}
    }
  }
}

/**
 * Closes all project files on project close.
 */
public void resourceChanged(final
IResourceChangeEvent event) {
  if (event.getType() ==
IResourceChangeEvent.PRE_CLOSE) {
    Display.getDefault().asyncExec(new Runnable() {
      public void run() {
        IWorkbenchPage[] pages =
getSite().getWorkbenchWindow().getPages();
        for (int i = 0; i < pages.length; i++) {
          if (((FileEditorInput)
xggEditor.getEditorInput()).getFile()
              .getProject().equals(event.getResource())) {
            IEditorPart editorPart =
pages[i].findEditor(xggEditor
                .getEditorInput());
            pages[i].closeEditor(editorPart, true);
          }
        }
      }
    });
  }
}

protected IPage getOutlinePageFor(Object object) {
  if (object == null)
    return null;

  IPage outlinePage = null;

  if (object instanceof IAdaptable)
    outlinePage = (IPage) ((IAdaptable) object)
      .getAdapter(IContentOutlinePage.class);

  if (outlinePage == null)
    outlinePage = (IPage)
Platform.getAdapterManager().getAdapter(object,
      IContentOutlinePage.class);

  return outlinePage;
}

private void addEditor(IEditorPart editor) {
  IPage outlinePage =
this.getOutlinePageFor(editor);

  if (outlinePage != null)
    outline.addOutlinePage(editor, outlinePage);
}

protected IContentOutlinePage getOutlinePage() {
  if ((outline == null) || (outline.isDisposed())) {
    outline = new XGGOutlinePage(this);

    addEditor(getEditor(0));
    addEditor(getEditor(1));
  }

  return outline;
}

@SuppressWarnings("unchecked")
public Object getAdapter(Class required) {
  if (IContentOutlinePage.class.equals(required))
    return getOutlinePage();
  return super.getAdapter(required);
}
```

cl

```
/**
 * The <code>MultiPageEditorPart</code>
implementation of this
 * <code>IWorkbenchPart</code> method
disposes all nested editors. Subclasses
 * may extend.
 */
public void dispose() {
  ResourcesPlugin.getWorkspace().removeResource
ChangeListener(this);
  super.dispose();
}
}

**********
FILE: sp/xggplugin/editors/XGGOutlinePage.java

package sp.xggplugin.editors;

import java.util.*;

import org.eclipse.ui.*;

import org.eclipse.ui.views.contentoutline.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.jface.viewers.*;
import org.eclipse.ui.part.*;
import org.eclipse.swt.*;
import org.eclipse.jdt.internal.ui.javaeditor.*;

@SuppressWarnings("restriction")
public class XGGOutlinePage extends Page
implements IContentOutlinePage,
  ISelectionChangedListener {
XGGMultiPageEditor mpe;
PageBook pagebook = null;
IContentOutlinePage mainPage = null;
ContentOutline content = null;
IViewPart outlinePart = null;
Object selectedPage = null;
IPage currentPage = null;
ISelection currentSelection = null;
IActionBars myActionBars;
SubActionBars currentActionBars;
SubActionBars defcurrentActionBars;

Map<Object, SubActionBars> subActionBars = new
HashMap<Object, SubActionBars>();
ArrayList<ISelectionChangedListener> listeners =
new ArrayList<ISelectionChangedListener>();
Map<Object, Object> outlineMap = new
HashMap<Object, Object>();

public XGGOutlinePage(XGGMultiPageEditor anMpe)
{
  mpe = anMpe;
}

public void createControl(Composite parent) {
  pagebook = new PageBook(parent, SWT.NONE);
}

public void dispose() {
  if (pagebook != null && !pagebook.isDisposed())
    pagebook.dispose();
  listeners.clear();

  currentPage = null;

  currentSelection = null;

  Collection<Object> pages = outlineMap.values();
  for (Iterator<Object> i = pages.iterator();
i.hasNext();) {
    Object page = i.next();
    if (page instanceof IPage)
      ((IPage) page).dispose();
  }
  outlineMap.clear();

  subActionBars.clear();
}

public Control getControl() {
  showOutline(mpe.getEditorAt(mpe.getCurrentPage
()));
  return pagebook;
}

public void setActionBars(IActionBars anActionBars)
{
  myActionBars = anActionBars;

  if (selectedPage != null)
    this.showOutline(selectedPage);
}

public void setFocus() {
  if (currentPage != null)
    currentPage.setFocus();
}

public void
addSelectionChangedListener(ISelectionChangedLis
tener listener) {
  listeners.add(listener);
}

public void
removeSelectionChangedListener(ISelectionChange
dListener listener) {
  listeners.remove(listener);
}

public ISelection getSelection() {
  return currentSelection;
}

public void setSelection(ISelection selection) {
  currentSelection = selection;

  if (listeners.isEmpty())
    return;

  SelectionChangedEvent event = new
SelectionChangedEvent(this, selection);
  for (Iterator<ISelectionChangedListener> i =
listeners.iterator(); i
    .hasNext();) {
    ISelectionChangedListener listener = i.next();
    listener.selectionChanged(event);
  }
}

public void addOutlinePage(IEditorPart key, IPage
outlinePage) {
  if (key == null)
    return;

  if (outlinePage == null)
    outlineMap.remove(key);

  outlineMap.put(key, outlinePage);
}

public void showOutline(Object anObject) {
  selectedPage = anObject;
```

```
  if ((isDisposed()) || (myActionBars == null))
    return;

  Object object = outlineMap.get(selectedPage);
  if (object == null || !(object instanceof IPage)) {
    object = new DefaultOutlinePage();
    outlineMap.put(selectedPage, object);
  }

  IPage outlinePage = (IPage) object;

  if (selectedPage != null && selectedPage
instanceof ISelectionProvider)
    ((ISelectionProvider)
selectedPage).removeSelectionChangedListener(thi
s);

  if (currentActionBars != null) {
    currentActionBars.deactivate();
    currentActionBars.updateActionBars();
  }

  if (outlinePage instanceof ISelectionProvider)
    ((ISelectionProvider)
outlinePage).addSelectionChangedListener(this);

  Control control = outlinePage.getControl();
  if (control == null || control.isDisposed()) {
    if (outlinePage instanceof IPageBookViewPage) {
      try {
        ((IPageBookViewPage)
outlinePage).init(getSite());
      } catch (PartInitException e) {
        e.printStackTrace();
      }
    }

    outlinePage.createControl(pagebook);
    control = outlinePage.getControl();
  }

  SubActionBars subActBars = (SubActionBars)
subActionBars.get(selectedPage);
  if (subActBars == null) {
    if (outlinePage instanceof JavaOutlinePage) {
      subActBars = (SubActionBars)
getSite().getActionBars();
      defcurrentActionBars = (SubActionBars)
getSite().getActionBars();
    } else
      subActBars = new
SubActionBars(myActionBars);

    outlinePage.setActionBars(subActBars);
    subActionBars.put(selectedPage, subActBars);
  }
  currentActionBars = subActBars;

  pagebook.showPage(control);
  currentPage = outlinePage;

  if (currentActionBars != null) {
    currentActionBars.activate();
    currentActionBars.updateActionBars();
  }
}

public void
selectionChanged(SelectionChangedEvent event) {
  this.setSelection(event.getSelection());
}

public boolean isDisposed() {
  return pagebook == null || pagebook.isDisposed();
}

private class DefaultOutlinePage extends
ContentOutlinePage {
private Label control;

public void createControl(Composite parent) {
  control = new Label(parent, SWT.NONE);
  control.setText("An outline is not available.");
}

public Control getControl() {
  return this.control;
}

public void setFocus() {
  this.getControl().setFocus();
}
}
}

**********
FILE: sp/xggplugin/editors/XMLConfiguration.java

package sp.xggplugin.editors;

import org.eclipse.jface.text.IDocument;
import
org.eclipse.jface.text.ITextDoubleClickStrategy;
import org.eclipse.jface.text.TextAttribute;
import
org.eclipse.jface.text.presentation.IPresentationRec
onciler;
import
org.eclipse.jface.text.presentation.PresentationReco
nciler;
import
org.eclipse.jface.text.rules.DefaultDamagerRepairer
;
import org.eclipse.jface.text.rules.Token;
import org.eclipse.jface.text.source.ISourceViewer;
import
org.eclipse.jface.text.source.SourceViewerConfigura
tion;

public class XMLConfiguration extends
SourceViewerConfiguration {
private XMLDoubleClickStrategy
doubleClickStrategy;
private XMLTagScanner tagScanner;
private XMLScanner scanner;
private ColorManager colorManager;

public XMLConfiguration(ColorManager
colorManager) {
  this.colorManager = colorManager;
}

public String[]
getConfiguredContentTypes(ISourceViewer
sourceViewer) {
  return new String[]
{ IDocument.DEFAULT_CONTENT_TYPE,
      XMLPartitionScanner.XML_COMMENT,
XMLPartitionScanner.XML_TAG };
}

public ITextDoubleClickStrategy
getDoubleClickStrategy(
  ISourceViewer sourceViewer, String contentType)
{
  if (doubleClickStrategy == null)
    doubleClickStrategy = new
XMLDoubleClickStrategy();
```

```
  return doubleClickStrategy;
}

protected XMLScanner getXMLScanner() {
  if (scanner == null) {
    scanner = new XMLScanner(colorManager);
    scanner.setDefaultReturnToken(new Token(new
TextAttribute(colorManager
      .getColor(IXMLColorConstants.DEFAULT))));
  }
  return scanner;
}

protected XMLTagScanner getXMLTagScanner() {
  if (tagScanner == null) {
    tagScanner = new
XMLTagScanner(colorManager);
    tagScanner.setDefaultReturnToken(new
Token(new TextAttribute(colorManager
      .getColor(IXMLColorConstants.TAG))));
  }
  return tagScanner;
}

public IPresentationReconciler
getPresentationReconciler(
  ISourceViewer sourceViewer) {
  PresentationReconciler reconciler = new
PresentationReconciler();

  DefaultDamagerRepairer dr = new
DefaultDamagerRepairer(getXMLTagScanner());
  reconciler.setDamager(dr,
XMLPartitionScanner.XML_TAG);
  reconciler.setRepairer(dr,
XMLPartitionScanner.XML_TAG);

  dr = new
DefaultDamagerRepairer(getXMLScanner());
  reconciler.setDamager(dr,
IDocument.DEFAULT_CONTENT_TYPE);
  reconciler.setRepairer(dr,
IDocument.DEFAULT_CONTENT_TYPE);

  NonRuleBasedDamagerRepairer ndr = new
NonRuleBasedDamagerRepairer(
    new
TextAttribute(colorManager.getColor(IXMLColorCons
tants.XML_COMMENT)));
  reconciler.setDamager(ndr,
XMLPartitionScanner.XML_COMMENT);
  reconciler.setRepairer(ndr,
XMLPartitionScanner.XML_COMMENT);

  return reconciler;
}
}

**********
FILE:
sp/xggplugin/editors/XMLDocumentProvider.java

package sp.xggplugin.editors;

import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.IDocumentPartitioner;
import org.eclipse.jface.text.rules.FastPartitioner;
import
org.eclipse.ui.editors.text.FileDocumentProvider;

public class XMLDocumentProvider extends
FileDocumentProvider {

protected IDocument createDocument(Object
element) throws CoreException {
  IDocument document =
super.createDocument(element);
  if (document != null) {
    IDocumentPartitioner partitioner = new
FastPartitioner(
      new XMLPartitionScanner(), new String[]
{ XMLPartitionScanner.XML_TAG,
          XMLPartitionScanner.XML_COMMENT });
    partitioner.connect(document);
    document.setDocumentPartitioner(partitioner);
  }
  return document;
}
}

**********
FILE:
sp/xggplugin/editors/XMLDoubleClickStrategy.java

package sp.xggplugin.editors;

import org.eclipse.jface.text.*;

public class XMLDoubleClickStrategy implements
ITextDoubleClickStrategy {
protected ITextViewer fText;

public void doubleClicked(ITextViewer part) {
  int pos = part.getSelectedRange().x;

  if (pos < 0)
    return;

  fText = part;

  if (!selectComment(pos)) {
    selectWord(pos);
  }
}

protected boolean selectComment(int caretPos) {
  IDocument doc = fText.getDocument();
  int startPos, endPos;

  try {
    int pos = caretPos;
    char c = ' ';

    while (pos >= 0) {
      c = doc.getChar(pos);
      if (c == '\\') {
        pos -= 2;
        continue;
      }
      if (c == Character.LINE_SEPARATOR || c == '\"')
        break;
      --pos;
    }

    if (c != '\"')
      return false;

    startPos = pos;

    pos = caretPos;
    int length = doc.getLength();
    c = ' ';

    while (pos < length) {
      c = doc.getChar(pos);
      if (c == Character.LINE_SEPARATOR || c == '\"')
        break;
```

```
      ++pos;
    }
    if (c != '\"')
      return false;

    endPos = pos;

    int offset = startPos + 1;
    int len = endPos - offset;
    fText.setSelectedRange(offset, len);
    return true;
  } catch (BadLocationException x) {}

  return false;
}

protected boolean selectWord(int caretPos) {

  IDocument doc = fText.getDocument();
  int startPos, endPos;

  try {

    int pos = caretPos;
    char c;

    while (pos >= 0) {
      c = doc.getChar(pos);
      if (!Character.isJavaIdentifierPart(c))
        break;
      --pos;
    }

    startPos = pos;

    pos = caretPos;
    int length = doc.getLength();

    while (pos < length) {
      c = doc.getChar(pos);
      if (!Character.isJavaIdentifierPart(c))
        break;
      ++pos;
    }

    endPos = pos;
    selectRange(startPos, endPos);
    return true;

  } catch (BadLocationException x) {}

  return false;
}

private void selectRange(int startPos, int stopPos) {
  int offset = startPos + 1;
  int length = stopPos - offset;
  fText.setSelectedRange(offset, length);
}
}

**********
FILE: sp/xggplugin/editors/XMLEditor.java

package sp.xggplugin.editors;

import org.eclipse.ui.editors.text.TextEditor;

public class XMLEditor extends TextEditor  {
private ColorManager colorManager;

public XMLEditor() {
  super();
  colorManager = new ColorManager();
  setSourceViewerConfiguration(new
XMLConfiguration(colorManager));
  setDocumentProvider(new
XMLDocumentProvider());
}
}

**********
FILE:
sp/xggplugin/editors/XMLEditorOutlinePage.java

package sp.xggplugin.editors;

import org.eclipse.swt.widgets.Composite;
import
org.eclipse.ui.views.contentoutline.ContentOutlineP
age;

import sp.format.*;
import org.eclipse.swt.graphics.*; //import
org.eclipse.core.runtime.*;
import org.eclipse.jface.viewers.*;
import org.eclipse.jface.resource.*;
import java.util.*;

public class XMLEditorOutlinePage extends
ContentOutlinePage {
XGGFormat xggMain = null;

public void createControl(Composite parent) {

  super.createControl(parent);

  TreeViewer viewer = getTreeViewer();
  viewer.setContentProvider(new
XGGContentProvider());
  viewer.setLabelProvider(new XGGLabelProvider());
  viewer.addSelectionChangedListener(this);
  viewer.setInput(xggMain);
  viewer.setAutoExpandLevel(2);
}

public void setXGGTree(XGGFormat aFormat) {
  xggMain = aFormat;
  TreeViewer viewer = getTreeViewer();
  if (viewer != null)
    viewer.setInput(xggMain);
}

public boolean isDisposed() {
  TreeViewer viewer = getTreeViewer();

  return (viewer == null);
}

class XGGContentProvider implements
ITreeContentProvider {
public Object[] getChildren(Object parentElement) {
  if (parentElement instanceof TreeType)
    return ((TreeType)
parentElement).getTreeChildren();
  return new Object[0];
}

public Object getParent(Object element) {
  if (element instanceof TreeType)
    return ((TreeType) element).getTreeParent();
  return null;
}

public boolean hasChildren(Object element) {
  return getChildren(element).length > 0;
}
```

```
public Object[] getElements(Object inputElement) {
  return getChildren(inputElement);
}

class XGGLabelProvider extends LabelProvider {
HashMap<ImageDescriptor, Image> imageCache =
new HashMap<ImageDescriptor, Image>();

public Image getImage(Object element) {
  ImageDescriptor descriptor = null;
  if (element instanceof String)
    descriptor =
PluginUtils.getImageDescriptor("public_co.gif");
  else if (element instanceof TreeType)
    descriptor =
PluginUtils.getImageDescriptor(((TreeType)
element)
      .getIconName());
  else
    return null;

  Image image = imageCache.get(descriptor);
  if ((image == null) && (descriptor != null)) {
    image = descriptor.createImage();
    imageCache.put(descriptor, image);
  }
  return image;
}

public void dispose() {
  for (Iterator<Image> i =
imageCache.values().iterator(); i.hasNext();)
    (i.next()).dispose();
  imageCache.clear();
}
}
}

**********
FILE: sp/xggplugin/editors/XMLPartitionScanner.java

package sp.xggplugin.editors;

import org.eclipse.jface.text.rules.*;

public class XMLPartitionScanner extends
RuleBasedPartitionScanner {
public final static String XML_COMMENT =
"__xml_comment";
public final static String XML_TAG = "__xml_tag";

public XMLPartitionScanner() {

  IToken xmlComment = new
Token(XML_COMMENT);
  IToken tag = new Token(XML_TAG);

  IPredicateRule[] rules = new IPredicateRule[2];

  rules[0] = new MultiLineRule("<!--", "-->",
xmlComment);
  rules[1] = new TagRule(tag);

  setPredicateRules(rules);
}
}

**********
FILE: sp/xggplugin/editors/XMLScanner.java

package sp.xggplugin.editors;

import org.eclipse.jface.text.rules.*;
import org.eclipse.jface.text.*;

public class XMLScanner extends RuleBasedScanner
{

public XMLScanner(ColorManager manager) {
  IToken procInstr = new Token(new
TextAttribute(manager
    .getColor(IXMLColorConstants.PROC_INSTR)));

  IRule[] rules = new IRule[2];
  // Add rule for processing instructions
  rules[0] = new SingleLineRule("<?", "?>",
procInstr);
  // Add generic whitespace rule.
  rules[1] = new WhitespaceRule(new
XMLWhitespaceDetector());

  setRules(rules);
}
}

**********
FILE: sp/xggplugin/editors/XMLTagScanner.java

package sp.xggplugin.editors;

import org.eclipse.jface.text.*;
import org.eclipse.jface.text.rules.*;

public class XMLTagScanner extends
RuleBasedScanner {

public XMLTagScanner(ColorManager manager) {
  IToken string = new Token(new
TextAttribute(manager
    .getColor(IXMLColorConstants.STRING)));

  IRule[] rules = new IRule[3];

  // Add rule for double quotes
  rules[0] = new SingleLineRule("\"", "\"", string, '\\');
  // Add a rule for single quotes
  rules[1] = new SingleLineRule("'", "'", string, '\\');
  // Add generic whitespace rule.
  rules[2] = new WhitespaceRule(new
XMLWhitespaceDetector());

  setRules(rules);
}
}

**********
FILE:
sp/xggplugin/editors/XMLWhiteSpaceDetector.java

package sp.xggplugin.editors;

import
org.eclipse.jface.text.rules.IWhitespaceDetector;

public class XMLWhitespaceDetector implements
IWhitespaceDetector {

public boolean isWhitespace(char c) {
  return (c == ' ' || c == '\t' || c == '\n' || c == '\r');
}
}
```

XI. ACKNOWLEDGEMENTS

I would like to thank the following people who have helped me finish this SP

- My older sister who helped me with a lot of things.
- My family who gave full support to me.
- To my dear teachers who have imparted their knowledge to me.
- Ma'am Magboo from guiding me throughout the whole study.
- Dr. Magboo from correcting my documents.
- Mr. Tuano from celebrating his birthday with the panel which distracted them from my proposal.
- Mr. Co for his expertise in Java.
- Mr.Chua for finishing ML early to give us more time.
- Mr. Patino for the XML tips and for the numerous sit-ins.
- The person in Azeus Systems who assigned me in the WPF group which gave me the idea to do this SP.
- Jenzen "Wapo" De Lara for the laptop and PSP.
- Donnel Patio for giving me invaluable help during the day of my defense.
- April Samson for having the defense at the same day as me, giving me enough time to catch up to it.
- Ginel Bacquero for the laptop.
- JC Pasco from help with the documents on the proposal.
- Whoever owns the Flash Drive I am using right now.
- Mark "Macho" Isla and Angelica Ricafrente for being the model students. Mark also saved me from taking the Doraemon role in the variety show.
- JM and Aaron for being the most abusive betatesters in dota AI+.
- Noah Mangahas for serious non-abusive AI+ betatesing.
- Ice frog for making my AI map "sponsored".
- Karen Ballesteros for lending money to get my grad pics
- Mike Tayag for the boastful moments.
- Therese Marcelo from the scans of paper in PI 100.
- Farrah Aaliwin and Shermaigne Yao for taking care of the CS197 reports.
- Rac Merjudio for the yearbook writeup.
- Jackie Isip from reminding me to start the MP in CS191 ML and help from STS movie reports .

- Ivhory Abag for being “friends forever” with Jenzen.
- Dave “The Master” Dela Cruz for sponsoring my yearbook.
- Ruth Gorospe for being the leader for the STS reports.
- To anyone whom I forgot to mention.