IBM ILOG JViews Diagrammer V8.6

Introducing JViews Diagrammer

Table of contents

About JViews Diagrammer

Tells you about diagrams in general, the features of the JViews Diagrammer product, and some typical uses of displays created with JViews Diagrammer.

In this section

Overview

JViews Diagrammer provides tools, components and services to build user-friendly interfaces made up of diagrams, dashboards, as well as map displays. The typical process for building a display with JViews Diagrammer consists in using application design tools, then refining and extending the result with the JViews Diagrammer SDK. The displays can then be deployed in Swing applications, applets and on the Web.

Types of displays covered by JViews Diagrammer

Describes the three types of displays JViews Diagrammer allows you to create, depending on the type of data to represent.

In this section

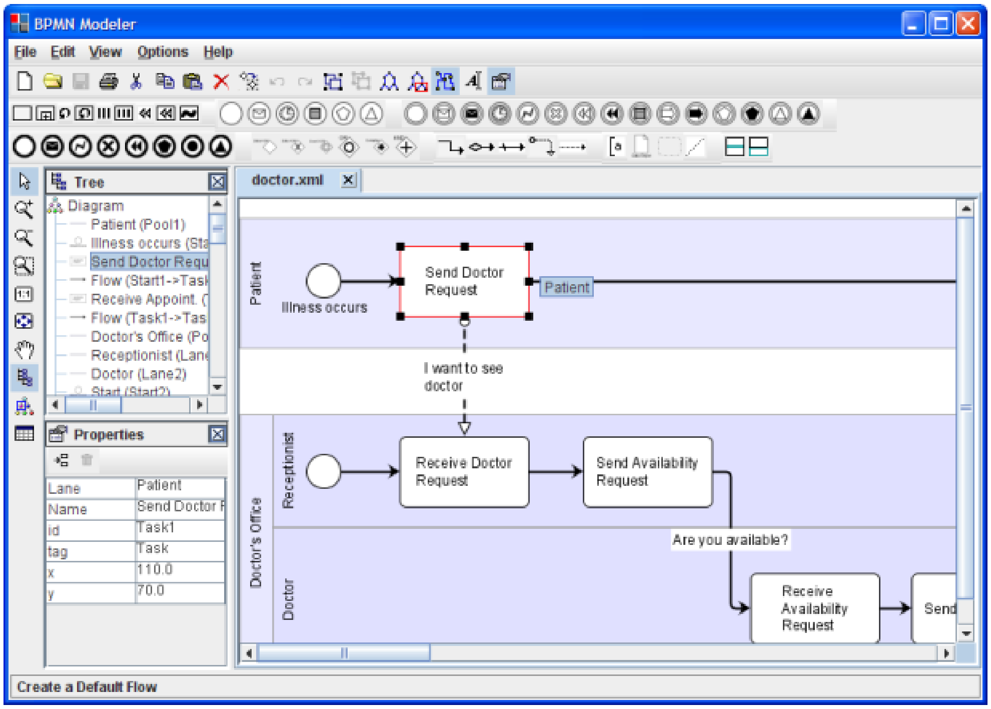
Diagrams

Diagrams are used to show the relationships between entities in a system. The entities are called *nodes* and the relationships are called *links*. This type of display helps model and manage a business system's entities connected either physically (like in a communications network) or logically (like in a process flowchart) .

JViews Diagrammer can be used to build computerized displays of static as well as dynamic diagrams. A diagram can be static in the sense that there are no changes in its appearance while it is displayed: it is a snapshot of a given system. Typical examples include flowcharts and organization charts.

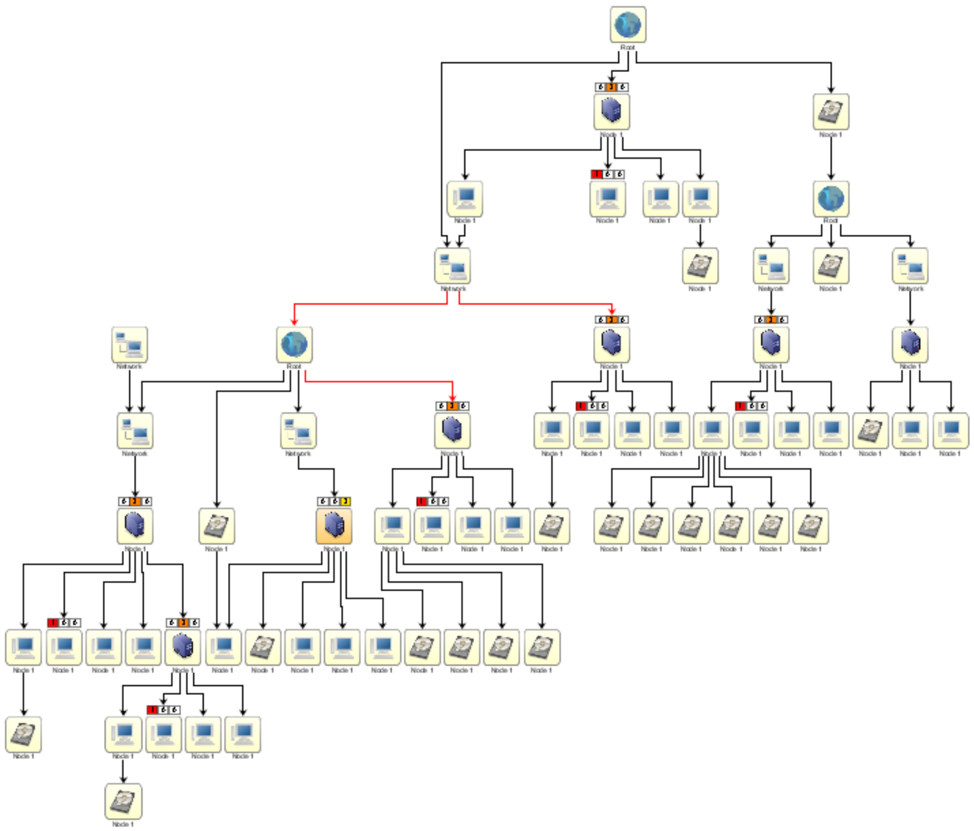
A dynamic diagram can react to user actions or external data feeds, or both. It remains in contact with business data during the display phase and is expected to change over time in response to business-related changes. Typical examples include process flow diagrams and network monitoring diagrams.

As diagrams grow larger and more complex, JViews Diagrammer allows you to make them more readable through its built-in graph layout algorithms, organizational techniques such as subgraphs and swimlanes, helpful zooming and scrolling behavior, and even customized interactions such as editing, drill-down, and more.



Lane	Patient
Name	Send Doctor R
id	Task1
tag	Task
x	110.0
y	70.0

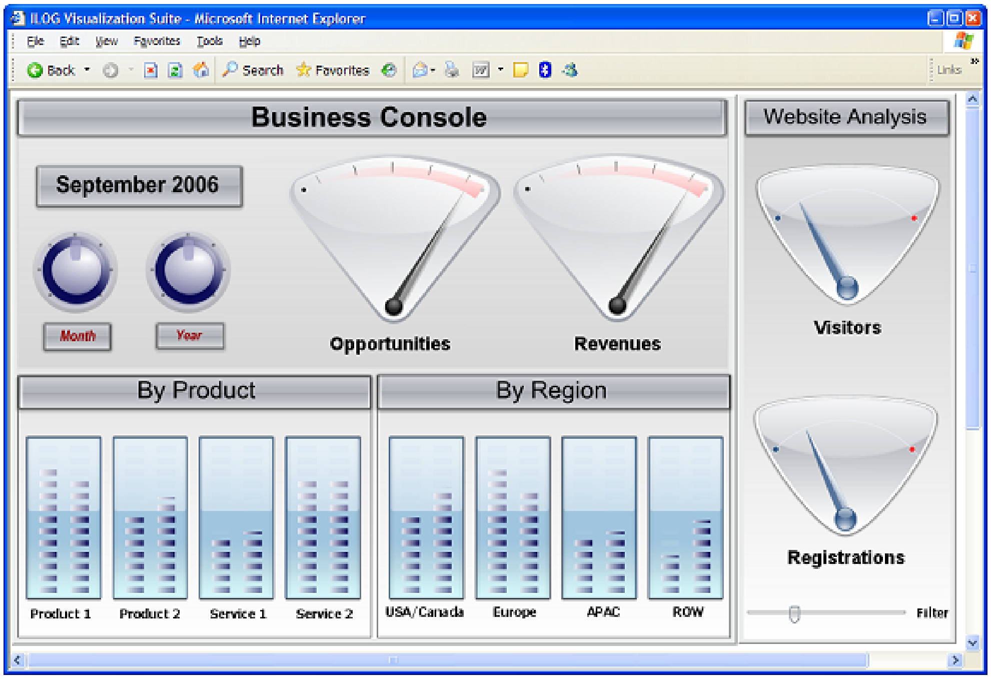
A workflow diagram



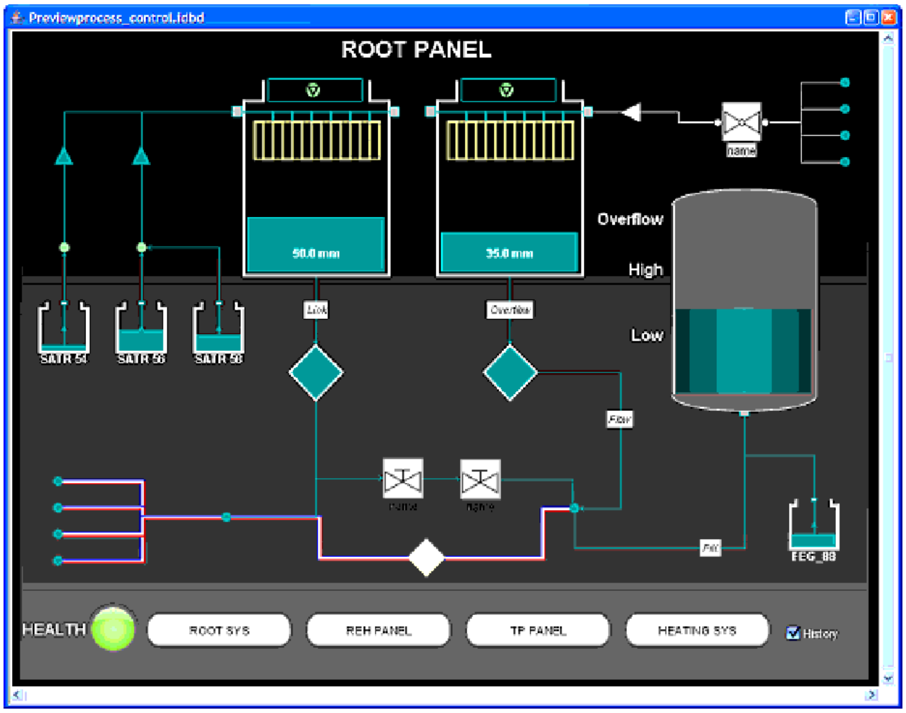
A topology diagram

Dashboards

Dashboards are used for monitoring business or industrial systems. They help operational people oversee the KPIs (Key Performance Indicators) of business processes. In the industrial field, dashboards are used for schematics, process control or SCADA (Supervisory Control And Data Acquisition) applications to show the current status of physical equipment. The data to be monitored is graphically represented by gauges, dials, sliders, meters, and so on, that are manually placed by the user on the dashboard. A connection between the data repository and these graphic objects is required for the dashboard to visually and realistically translate measurements or the status of physical equipment.



A business activity monitoring dashboard

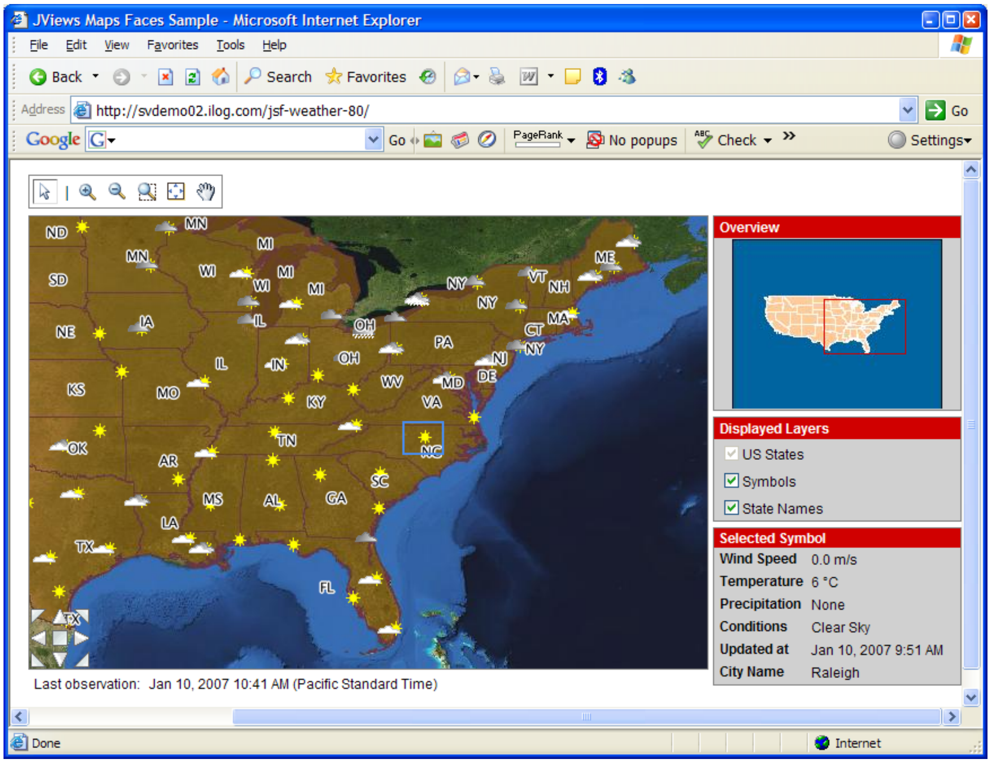


An industrial monitoring dashboard

Map-based displays

Map-based displays are used to represent and manage georeferenced assets in applications such as logistics, defense, traffic monitoring, or network systems. Objects are placed on top of a map according to their latitude and longitude coordinates; in a dynamic application, the geographic positions of the objects are updated as the values of the underlying data change.

Map-based displays are fully supported if the extra product IBM® ILOG® JViews Maps is purchased and installed.


JViews Maps Faces Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Go
Google Go PageRank No popups Check Settings
Overview
Done
Internet

http://svdemo02.ilog.com/jsf-weather-80/

Displayed Layers

- ✓ US States
- ✓ Symbols
- ✓ State Names

Selected Symbol

Wind Speed	0.0 m/s
Temperature	6 °C
Precipitation	None
Conditions	Clear Sky
Updated at	Jan 10, 2007 9:51 AM
City Name	Raleigh

Last observation: Jan 10, 2007 10:41 AM (Pacific Standard Time)

A weather map in a Web application

Types of deployment platforms

It has become more and more common to deliver the same application both as a rich desktop client for power users and over the Web for remote users. JViews Diagrammer displays can be deployed on various platforms:

- Rich client
- JSF/Ajax/Portals
- Eclipse™

Traditional Java™ applets and applications are best suited for highly interactive tasks like workflow modeling. For workflow monitoring or administration, a thin client approach may work best. JViews Diagrammer supports DHTML clients as well as the traditional Java clients.

On the Web side, there are dedicated JSP™ /JSF (JavaServer™ Faces) components that can live on a Web server and generate the interface for the browser. By mixing images and JavaScript™ /DHTML code, these components deliver content for either traditional Web pages or portals that implement the JSR 168 standard. They are also able to deal with asynchronous requests to manage the Ajax (Asynchronous JavaScript and XML) behavior and minimize page refreshing. In addition to traditional visualization and monitoring functions, Ajax behavior provides powerful in-place editing capabilities, such as adding objects to a diagram, connecting entities interactively, showing contextual popup menus, and editing an object's attributes, with immediate synchronization with the underlying data models on the server.

Key aspects of a JViews Diagrammer user interface

Developing a graphical user interface with JViews Diagrammer requires to be aware of the following key aspects:

1. Symbols
2. Model-driven diagrams
3. Dashboards and Monitoring Panels

Symbols

Symbols are the starting point of a display (diagram or dashboard). A symbol is a self-contained graphic object that represents a physical or conceptual element in the underlying application. For example, symbols can represent trucks, factories, network elements, dials or gauges. They have built in behavior which means that they react dynamically to data changes or user interaction.

A symbol and its behavior are not defined by code but in a CSS (Cascading Style Sheet) file that contains a description of the JavaBeans™ to use, their settings and logic. At run time, a generic engine instantiates the right class and configures the created instances as specified in the CSS file. For better performance, or if such symbols can be frozen for a given application, it is also possible to generate Java™ source code corresponding to these symbols.

Model-driven diagrams

Applications that must automatically create a diagram from a data source (like workflows, business processes, entity relationships, or network topologies) use a model-driven approach. The design tool used to create this type of display is the Designer for JViews Diagrammer. The Designer creates a display by binding the data source elements to symbols created with the Symbol Editor. For example, you can specify that a given type of data instance be represented by a given symbol, and that a particular value of a field be bound to some symbol parameters to alter the aspect of the symbol. The specification of how the visual aspect of the symbol is controlled by the data model is performed at a high level as the symbols already contain their particular visual logic. Other aspects of the display, such as automatic graph layout options, links, background images, subgraphs, zooming can also be tuned through the Designer tool. The mapping between the data model and the visual entities (symbols) is described in a CSS format. The output is a project file combining the CSS part and the XML part (data source) and can be loaded into the application at run time. The project's individual elements can be accessed through dedicated Java classes for further customization of the look or behavior of the display.

Dashboards and monitoring panels

To build displays for monitoring or supervision purposes (like industrial schematics, business dashboards, and other generic human-machine interfaces), the approach is quite the opposite from the model-driven approach, in the sense that you start by manually placing symbols created with the Symbol Editor on top of a static background. This is done with another design tool called the Dashboard Editor. Symbol parameters can be associated with application data at design time, but this mapping will be resolved only at run time when connecting the actual data source. The resulting dashboard or schematic is an XML file associated with palettes of symbols that can be loaded into an application window and fed with real-time data.

The JViews Diagrammer tool chain

JViews Diagrammer contains several design tools to automate the production of applications without coding. The design tools are point-and-click editors that allow user interface developers to quickly prototype the look and feel of the display without having to spend time with Java™ code. These tools address the different aspects involved with producing appropriate content for graphical diagrams, dashboards and map displays. Besides the need to reduce the coding part of an application, the design tools help address the different roles in the development chain. For example, a graphic designer can provide attractive content for the user interface without necessarily being involved in technical development, or an application administrator can enrich the application without modifying the core of the system.

The design tools provided with JViews Diagrammer are the following:

♦ Symbol Editor

  A point-and-click interface that allows you to create and edit symbols. It also enables you to add dynamic behavior to these symbols with rules defining how the elements of the symbol will react to data changes. For example, rules can define the alarm conditions that will cause a part of a factory symbol to blink, or how far a needle on a rotary gauge will rotate. Symbols are organized in palettes that are reusable in other design tools.

♦ Designer

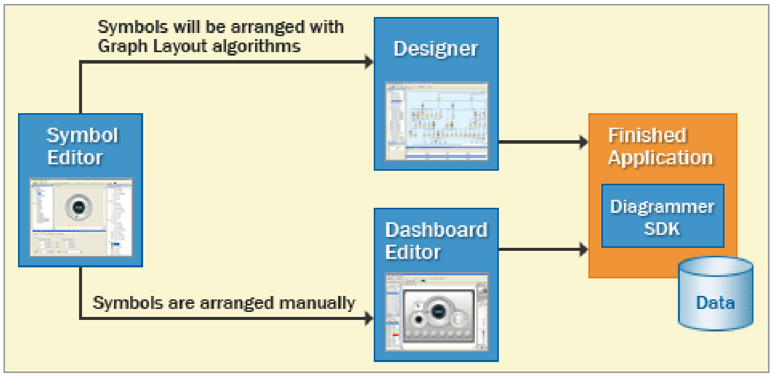
  A point-and-click editor for easily specifying most aspects of a diagram. It is well-suited for defining the look and feel of applications that must automatically create a diagram—such as a business process, a network typology, a workflow, a dataflow, or entity-relational diagrams—from a data source.

♦ Dashboard Editor

  A point-and-click interface for the easy creation of industrial panels, business dashboards and other generic human-machine interfaces. It allows users to manually place symbols, created with the Symbol Editor, on top of a static background. It then connects them to the underlying data.

♦ Symbol Compiler

  The Symbol Compiler is a tool that allows you to translate the symbol definition created with the Symbol Editor into Java classes. The symbols that you create in the Symbol Editor use a CSS file to store the information necessary to their graphical representation. A compiled symbol provides improvements in terms of performance, as it is created in Java code instead of CSS, and flexibility, as its features can be extended through the JViews Framework API.


Symbols will be arranged with Graph Layout algorithms
Designer
Symbol Editor
Finished Application
Diagrammer SDK
Dashboard Editor
Data
Symbols are arranged manually

The JViews Diagrammer tool chain

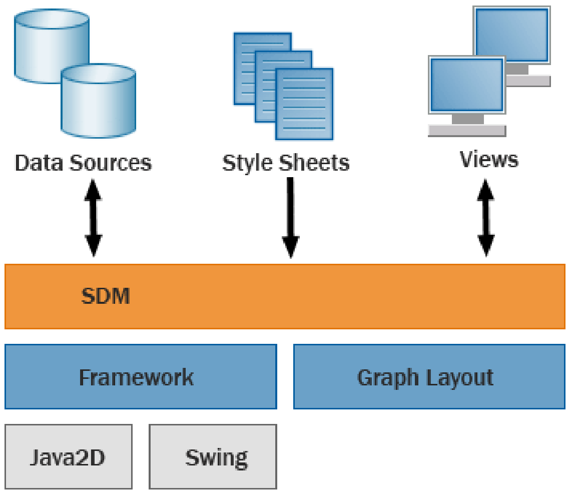
Basic concepts

Presents the concepts you need to know to evaluate JViews Diagrammer and start to appreciate its features and capabilities. The JViews Diagrammer documentation makes use of these concepts and you will need to understand them to implement your requirements.

In this section

General architecture

JViews Diagrammer consists of several layers, each providing a specific set of services for developers during the development cycle.



Architecture of JViews Diagrammer

JViews Diagrammer implements a Swing-like model-view architecture, which provides a clear separation between data and its presentation. The JViews Diagrammer data model is part of the Styling and Data Mapping (SDM) engine, and it forms the connection between your application's data and the views on display. The two principal responsibilities of the SDM module are notification and styling. When it receives new data from its back-end data source, the SDM module automatically notifies all views registered for an update. Likewise, when the user interacts with views, the model may change.

SDM relies on the services of two underlying layers, also included in JViews Diagrammer—the Framework SDK and the Graph Layout SDK.

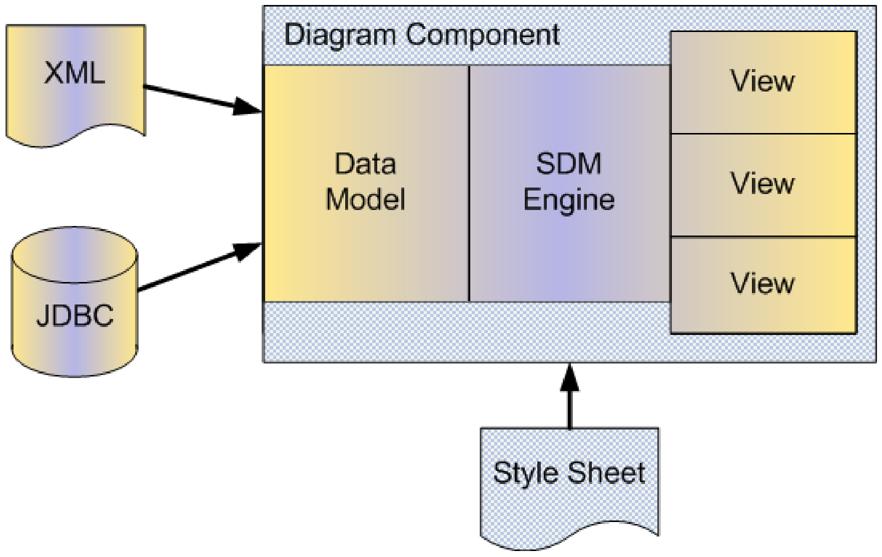
The Framework SDK sits atop Java 2D™ and Swing, and provides a comprehensive structured 2D graphics API that includes graphic objects, interactors, views, transformations, graphs and subgraphs, editing commands, printing, and thin client support

The Graph Layout SDK provides a set of sophisticated algorithms to automatically rearrange the graph elements for optimal readability. Graph layout brings order and clarity to complex diagrams, moving the nodes and routing to create a more usable picture with domain-specific aesthetic conventions.

Views

Ultimately, diagrams are displayed on the user screen within views that can occupy an entire window or part of one.

Each view has its own zoom level and displays a part of the diagram. The user can zoom and pan within a view to focus on regions of interest. At any moment, multiple views can be used with a single diagram as if the user had several cameras to give different points of view, see the following figure.



A diagram component with multiple views

The overview is a special use of the multiple-view concept: it is a view that displays the entire diagram in miniature, overlaid with a navigation rectangle that represents the visible part of the diagram in the main view. By moving and resizing this rectangle, the user can easily zoom and pan the main view.

The diagram's graphical objects can be grouped in layers that control their display priority. You may, for example, decide that the nodes and links are in a higher layer than the background map: they are always displayed on top of the map. You may also decide that labels are always displayed on top of everything by grouping them in the highest layer. Layers can also be set visible or invisible—for each view—so that you can temporarily hide an entire group of symbols or a background map.

Finally, JViews Diagrammer offers three alternative views of a diagram's data model: a table view, a tree view, and a property sheet.

In the table view, the properties of the nodes and links are displayed as a Swing JTable that can be edited in an application. In the tree view, the data model is displayed as a Swing JTree, which is useful for selecting objects—the Data Model panel in the Designer uses this view. For both of these, see the following figure.

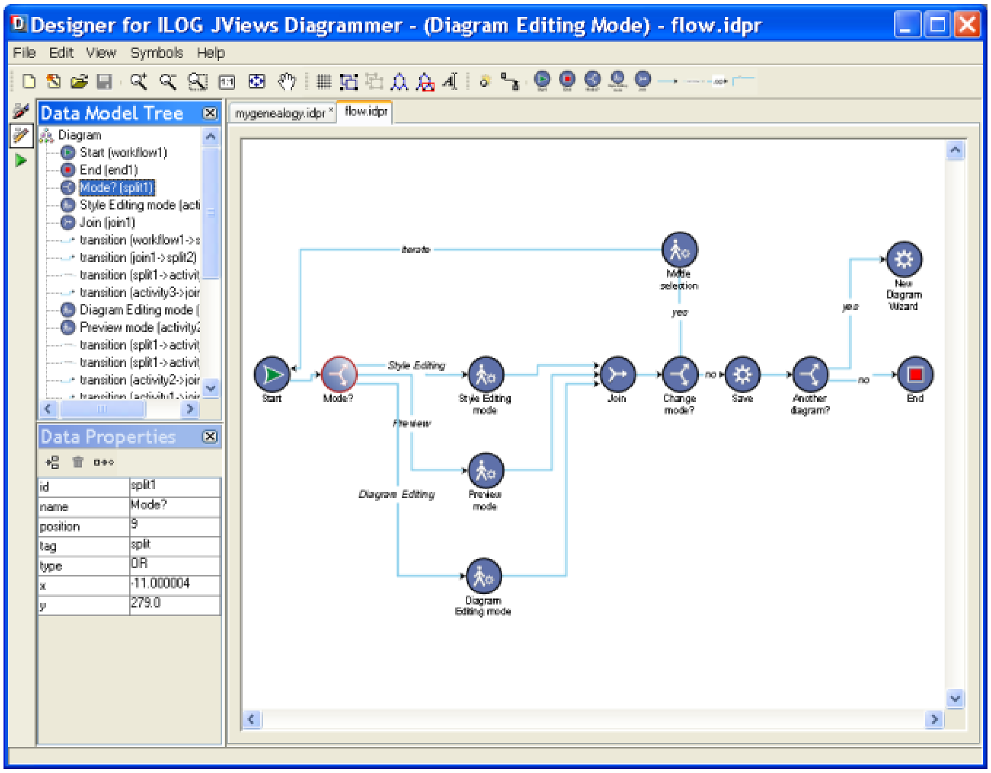


Diagram
- Start (workflow1)
- End (end1)
- Mode? (split1)
- Style Editing mode (acti
- Join (join1)
- transition (workflow1->s
- transition (join1->split2)
- transition (split1->activi
- transition (activity3->joir
- Diagram Editing mode (
- Preview mode (activity2
- transition (split1->activi
- transition (split1->activi
- transition (activity2->joir

id	split1
name	Mode?
position	9
tag	split
type	OR
x	-11.000004
y	279.0

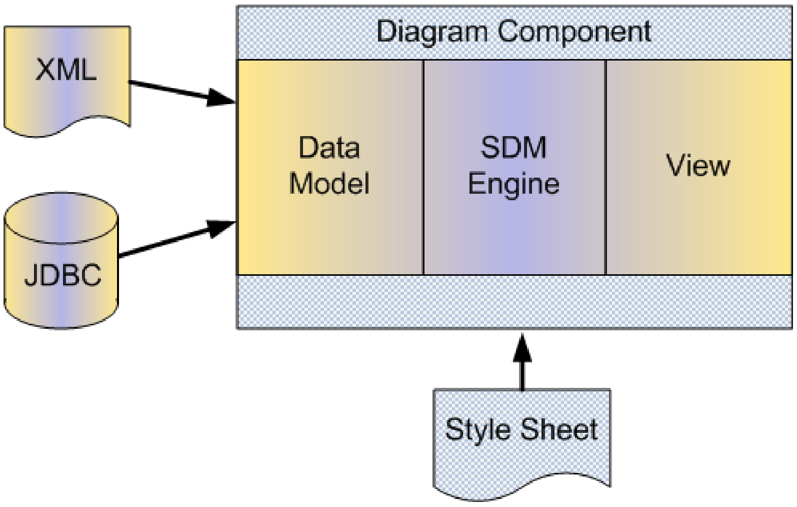
Tree view (top left) and table view (lower left) in the Designer

In a property sheet, a simple list of properties is displayed—the optional Styling Properties panel in the Designer displays this view.

Data

JViews Diagrammer offers you the opportunity to stay away from the low-level graphics API, and to worry only about your business data and the way you want to display it. The section on *Creating diagramming applications* explains how this works in more detail; at this point it is sufficient to explore the basic concept of a data model.

JViews Diagrammer is based on a model-view architecture that cleanly separates the data, the display, and the interaction facets of the component, see figure *The model-view architecture of JViews Diagrammer*. It follows the Swing architecture where the developer takes care of populating the data model and the component takes care of displaying the data and enabling interactions like selection and editing.



The model-view architecture of JViews Diagrammer

In JViews Diagrammer, the data model is an interface that manages nodes and links. Nodes have a set of generic properties like x and y coordinates, and user-defined properties can be added to store application-dependent information. Similarly, links have generic properties like `to` and `from` for the source and destination nodes, and can have further, user-defined properties. Based on this data model, JViews Diagrammer knows how to display the diagram, and how to manage end-user interactions.

The data model needs to be populated with application data. To do this, you have the choice between using a prebuilt implementation such as the in-memory data model, or connecting the diagram directly to your data by implementing the data model interface. The latter solution avoids data duplication and enables finer synchronization between the diagram and the data.

JViews Diagrammer provides data sources to populate your diagram from XML files, JDBC connections, or flat files in formats like CSV (comma-separated values).

Styling

Besides displaying nodes and links, your application will need to convey qualitative information on your business data. Although diagrams such as business processes, electrical networks, WAN networks, UML diagrams, and supply-chain maps are all based on nodes and links, they do not look like similar: each case requires notations and symbols that are application-specific.

To define the notation for a particular application, JViews Diagrammer makes use of a powerful model-based styling mechanism that relies on style rules. Each rule defines conditions on the data model that trigger graphical changes in the display. For example, you can define a rule that draws a green rectangle when the `status` property of a node equals `fine`; and another rule that turns the rectangle to red when the `status` property equals `alarm`.

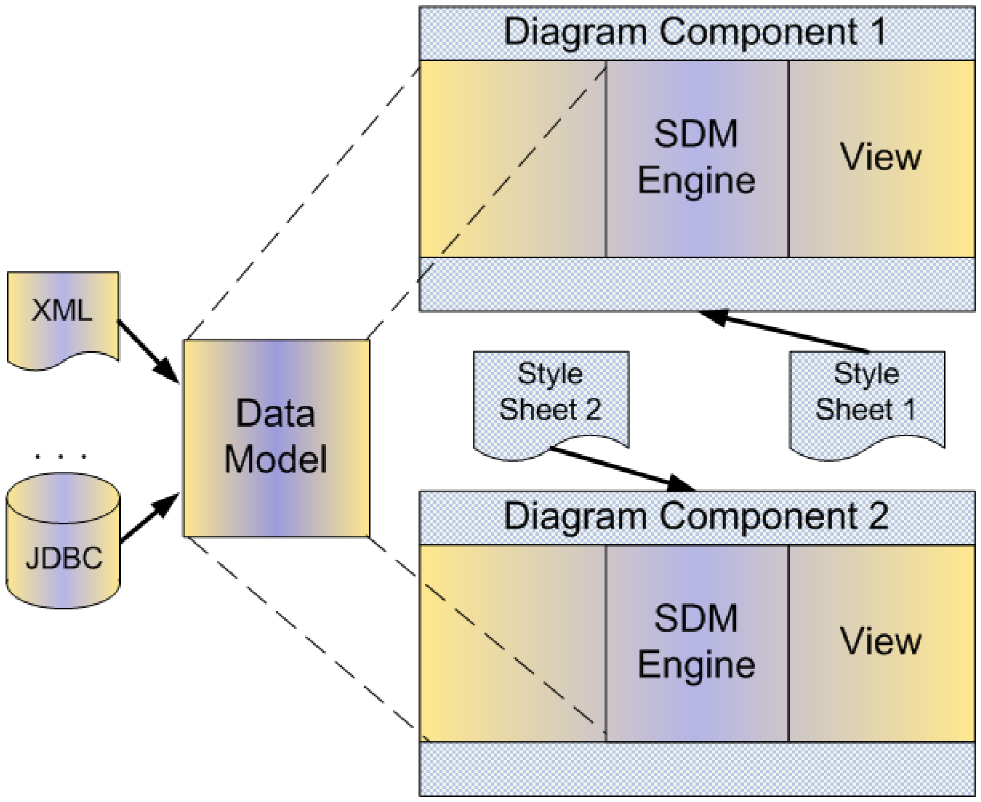
When you define a notation, you create the basic symbols to be used for default situations. In an application that deals with business processes, these symbols represent activities, participants, messages, and so on. In a telecommunications application, you need symbols for the many types of network elements. Each basic symbol is defined by at least one style rule.

Once the basic symbols are created, you then need to modify or annotate them to reflect specific properties that you want to display. You may want to change the background color to `red` when the `status` of a node equals `alarm`, or you may want to add a work-in-progress icon when an activity is currently being performed. For each situation, you define a rule that may complement or override the graphics effects defined by more generic rules.

The styling mechanism is also used to declare and customize options for the diagram as a whole, like the use of layout algorithms and their parameters, or the use of a background map and its source file and projection.

The set of style rules is stored in a style sheet, and you can dynamically load a new style sheet while keeping the same data and underlying data model. This facility is useful when you need to adapt the display to a particular situation or user profile. For example, the technical properties of a business process can be hidden to the business analyst, and shown only to the software engineer in charge of implementing the process.

The style sheet syntax conforms to the CSS syntax—a Web standard—but you need not bother with the details of this syntax at this point. The separation of style sheets and the data model from the diagram component means that you can build several components based on the same data, with the same or different styling, see the following figure.



A family of diagram components for a data model

JViews Diagrammer eases the process of defining a notation with a development tool called the Designer. Using the Designer, you define the conditions for a rule using natural language and you define the styling effects of the rule through an intuitive panel that lets you change graphical properties. While you are modifying rules or adding new ones, you can select a rule in a tree view and see how the notation is changing within a preview window.

JViews Diagrammer comes equipped with Business Process Management Notation (BPMN) a standard notation for business processes that is specified by the BPMI organization (see *http://www.bpmi.org*). This notation provides a comprehensive set of symbols and sophisticated swimlanes.

Symbology

JViews Diagrammer introduces the concept of *symbol* used to populate the graphical interfaces.

A symbol is an abstraction of composite graphic that can be used through style sheets or directly through Java™ code. Symbols are organized within palettes, and they have a dedicated tool to edit them (the Symbol Editor). By default, such symbols are based on a set of CSS directives which can be interpreted at run time. As a consequence, palettes of symbols can be edited and modified at any time to improve symbols or to add new symbols to the application. The other way to use symbols is to generate the Java source code corresponding to each symbol. In this case, symbols are tightly integrated with the application and cannot be modified from the outside.

A composite graphic object is an instance of `IlvSDMCompositeNode`. This class is in the SDM package but it is a subclass of `IlvCompositeGraphic` which is a JViews Framework class.

User interactions

When you have populated your data model, and defined your notation, you still need to provide your end user with a means of interacting with the diagram.

Modeling applications require interactions for creating nodes of many sorts, for creating links with various shapes, for setting user-defined properties, for selecting nodes and links, for moving nodes around, for editing subgraphs, and so forth.

JViews Diagrammer offers a wide range of interactors that implement the most common editing actions. When an interactor is installed on a view, it handles the user events, transmits the changes to the data model, and refreshes the view.

Monitoring applications often require a subset of the interactions proposed in modeling applications. The most common ones are zoom, pan, and selection. In monitoring applications, the selection is often specific to the application: when the user selects a node or a link, JViews Diagrammer performs a specific action like opening a dialog box or selecting all the alarms generated by the selected object. To implement application-specific interactions, you will have to derive new interactors from existing ones using the documented API provided with the SDK.

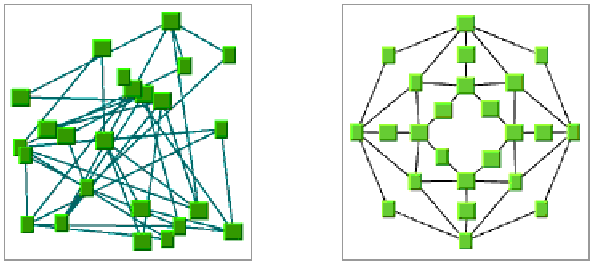
Graph layout

Note: Graph layout features are only available if you have purchased a full JViews Diagrammer license.

In some situations, diagrams can be drawn manually and still kept clear to read. The user typically places the nodes and links and repositions the nodes manually if the diagram becomes cluttered.

In many cases, the diagram is too complex to be handled manually, or there is no human being involved: the diagram is built dynamically with application data that does not contain any geometric information. In such cases, automatic layout algorithms are required to position the nodes and to route the links.

The Graph Layout package is a set of layout algorithms provided with JViews Diagrammer to position nodes and labels, and to route links, automatically. The goal of a layout algorithm is to propose solutions in which nodes do not overlap, links do not cross nodes or links, and labels do not overlap other objects. Ideal solutions do not always exist, but an algorithm will tend to an optimal one, see the following figure.


Applying a graph layout: before and after

The most effective way of displaying a graph depends on its type and sometimes on industry standards. JViews Diagrammer offers various algorithms with many parameters to adapt to the many situations. For example, the hierarchical layout is mainly used for displaying flows and processes; circular and bus layouts for LAN networks; radial tree layout for semantic networks and website maps; and tree layout for organization charts, decision trees, and directories.

In JViews Diagrammer, several graph layouts can be applied to a single diagram: a subgraph may require a different layout to its parent; or one layout can be applied to only a subset of the nodes. Constraints can also be set, for example, to make nodes with a certain property stay above the others.

When it comes to routing links, JViews Diagrammer offers the following: strategies to limit the length of links, algorithms to bundle links or connect nicely to nodes—especially when several links arrive at a single node.

The label layout (labeling) algorithm makes sure that labels do not overlap—which would make them hard to read—by shifting them slightly away from their base position and by rotating them if necessary. This algorithm is used for labels on nodes and links, and for labels in IBM® ILOG® JViews Maps.

Refreshing the display in real time

If your application data changes continually with real-time updates, the diagram display must stay synchronized and will therefore need to be refreshed often.

JViews Diagrammer uses refresh techniques that reduce the amount of graphical primitives to draw, while eliminating the annoying flickering effect that you can observe when a screen is erased and redrawn. It uses techniques like double buffering or triple buffering.

More generally, the 2D vectors that draw the basic shapes for nodes, links, and vector maps to be displayed in a view are stored in a spatial data structure called the grapher. The grapher ensures very fast redisplays and user interactions even when the diagram and its background reach several hundreds of thousands of vectors.

Creating diagramming applications

Introduces the architecture of JViews Diagrammer to give you an overview of how diagramming is done in JViews Diagrammer and the internal and external components that contribute.

In this section

Managing the views

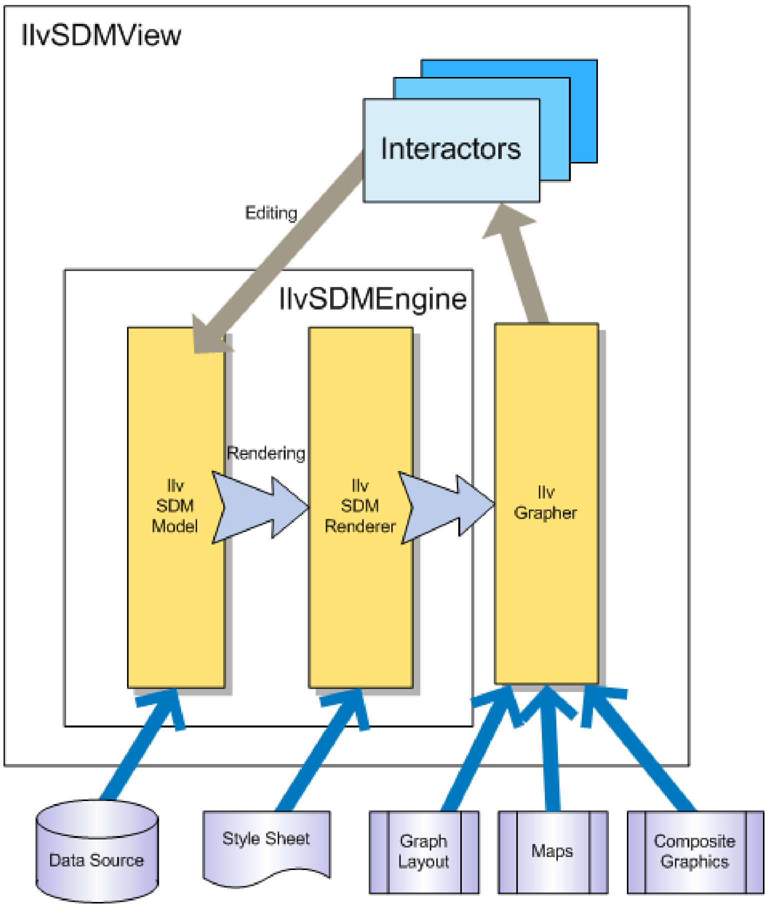
Describes the components of JViews Diagrammer that play a part in the management of the views.

In this section

The SDM engine

The Styling and Data Mapping (SDM) engine is one of the most important pieces of JViews Diagrammer as it controls the data-to-graphics mapping. There are four key elements in the data-to-graphics mapping process:

- A data model that interfaces to the data to display or edit. This data model is completely independent of the GUI, and refers only to the business objects of your application.
- Renderers that style the diagram as a whole and the graphic objects in it. Renderers apply the styles specified in the style sheets.
- A grapher in which the graphic objects representing the data model are created as nodes and links. It provides the infrastructure that is minimally necessary to draw a diagram.
- Interactors that permit user actions on graphic objects. Common requirements are for zoom, pan, select, and object creation functions.



The SDM engine and the data-to-graphics mapping

As shown in the figure above, the mapping between the data model and the graphical representation is bidirectional:

♦ Data model to graphics: the rendering process is controlled by the style sheet, which lets you tell the SDM engine how you want each particular kind of data object to be displayed in the grapher. The rendering process is performed by specialized renderers.

- When the data model is loaded, the SDM engine explores it and creates graphic objects representing the nodes and links defined by the data model in the grapher.
- When the state of an object in the data model changes, the SDM engine updates the graphic object representing the modified data object.

♦ Graphics to data model: the editing process relies on built-in editing facilities that act directly on the underlying data model. The actions in an editing application are implemented by interactors. For example:

- When the user moves a graphic object (for example, in an editor), the SDM engine updates the geometric properties of the object in the data model.
- When the user expands or collapses a node (for example, in a navigation application), the SDM engine updates the expand/collapse status of the object in the data model.

The data model interface

The SDM data model is the interface that tells the SDM engine how to get the data to be displayed. The SDM data model is an abstract description of a set of nodes and links between nodes. Nodes and links have a user-defined type (also called the “tag”), and a set of named properties.

JViews Diagrammer provides prebuilt data models, and you can implement the data model interface to connect your data.

The renderers

A renderer is a Java™ class that helps to manage the graphical representation of your business data. JViews Diagrammer supplies many predefined renderers. Several renderers are usually active at the same time.

The StyleSheet renderer is responsible for creating and customizing graphical objects and is always active. The GraphLayout renderer applies a layout algorithm and is often active (only available if you have purchased a full JViews Diagrammer license).

Predefined renderers are supplied for displaying info balloons, a legend, subgraphs, and swimlanes. There are also predefined renderers to assign a random color to nodes, to assign a specific interactor to nodes, and many more. The SDK lets you define new renderers for your application needs.

The grapher

The grapher is a Java™ class responsible for storing and managing the graphics objects that are displayed by JViews Diagrammer. The grapher not only manages nodes and links; it also manages any graphics objects used to display a background and any decorations around the diagram like a legend.

The grapher can contain hundreds of thousands of graphics, and yet refresh them instantly. Its spatial data structure is optimized to retrieve the graphic objects located in an area rapidly.

The grapher belongs to IBM® ILOG® JViews Framework, and a comprehensive API is available to finely tune its behavior.

The graphical objects contained in the grapher are drawn in the views connected to the grapher. The user of your application can zoom and pan in the views, and can interact with the objects in the views using the mouse.

The interactors

An interactor is a Java™ class that manages the behavior of an object in response to an event, for example, a mouse click. JViews Diagrammer supplies various predefined interactors.

View interactors affect the display in general. The view interactors are typically used to manage the zoom and pan facilities, and the selection and creation of nodes and links. It is important to remember that when a node is added or modified with an interactor on a view, the corresponding action is performed in the data model.

An object interactor is local to a graphic object. It handles the action to perform when an event is received by a particular object, for example, double-click to open a list of related alarms.

Populating a diagram

Populating a diagram involves the following items:

- ♦ Data sources
- ♦ Style sheets
- ♦ Project

The data sources

The role of the data source is to load the data to display in the diagram, and possibly write back the data if it has been modified.

There are the following predefined types of data source: flat files (in the Designer only), XML, and JDBC.

When you cannot use flat files, XML, or JDBC, you can always connect the data model to your data by implementing the data model interface in Java™ .

The style sheets

A style sheet controls the mapping of data to a graphic representation through style rules conforming to the CSS2 syntax. It defines the way the objects of your data model will be translated to graphic objects.

You can write these rules using the CSS2 syntax, or you can use the Designer. The Designer helps you to define style rules in a visual environment: with the Designer, you do not need to know CSS and you see instantly what your styling looks like.

The project

The project is an association of a style sheet and a data source. It groups the inputs for a diagram. A project is saved as an XML file with the extension .idpr (JViews Diagrammer Project File).

A project is typically generated by Designer, the editing tool available for loading data from a data source and creating a style sheet.

By loading a project in an application, you make a diagram available for display.

Graph layout

Note: Graph layout features are only available if you have purchased a full JViews Diagrammer license.

The SDM engine and renderers create the nodes and links in the grapher, but the positions of these objects may not be ideal: some nodes may overlap and there may be too many link crossings, so that the resulting diagram can be difficult to apprehend.

When a graph layout renderer is declared, a graph layout algorithm positions the nodes and labels that are stored in the grapher and routes the links.

There are several algorithms: some just position nodes, others just route links, and some do both.

You need to choose from the various algorithms available, and your choice will depend on the topology of your diagram, and possibly on your industry. The many parameters will help you to tune the algorithm to fit your specific needs. You can use the Designer to help find the best solution.

Backgrounds and maps

If your application requires a geographic map as a background, you can install a map renderer that uses the IBM® ILOG® JViews Maps facilities to read map formats—vector or raster—and to display nodes according to their latitude and longitude.

You can instead add a vector or a bitmap background that is not a map. You can use Composer, a graphics editor provided with JViews Diagrammer, to draw your vector background. Composer saves to the JViews proprietary IVL format or to SVG format. Other authoring tools that create bitmaps or SVG images can also be used.

Using Designer projects

The default style sheet provided with JViews Diagrammer displays a classical diagram with basic nodes and links that do not embody application-specific properties.

To provide your end user with meaningful displays, you need to assign symbols using the Symbol Editor tool accessible through the Designer.

The easiest way to define the diagram look and feel is to use the Designer. You can also write your style sheet manually without Designer, and set up the diagram parameters with the SDK, but you should try Designer first before deciding to take a different route.

How the Designer helps the developer

As a Java™ developer, you can decide to build diagram components with the Java API of JViews Diagrammer and the CSS2 styling language. But starting with a point-and-click editor like the Designer will make the early phases of development easier and faster.

The Designer is a development tool that is designed for defining the look and feel of your diagram.

With the Designer:

- ♦ You do not need to know the CSS syntax (although you do need to know the principles behind CSS)
- ♦ You can change the graphics properties through simple dialogs
- ♦ You get instant feedback on the modifications you make on style rules
- ♦ You can modify the data model
- ♦ You generate a project that can be used right away in your application

The Designer offers facilities for filling the data model, viewing the diagram, viewing and creating style rules, configuring the layout, defining a background, and saving a project.

Filling the data model

To use the Designer, all you need is a data model.

The Designer provides a wizard that helps you to fill the data model from XML files, a JDBC connection, CSV files, or by entering the data manually. It also provides samples of models to help you get started.

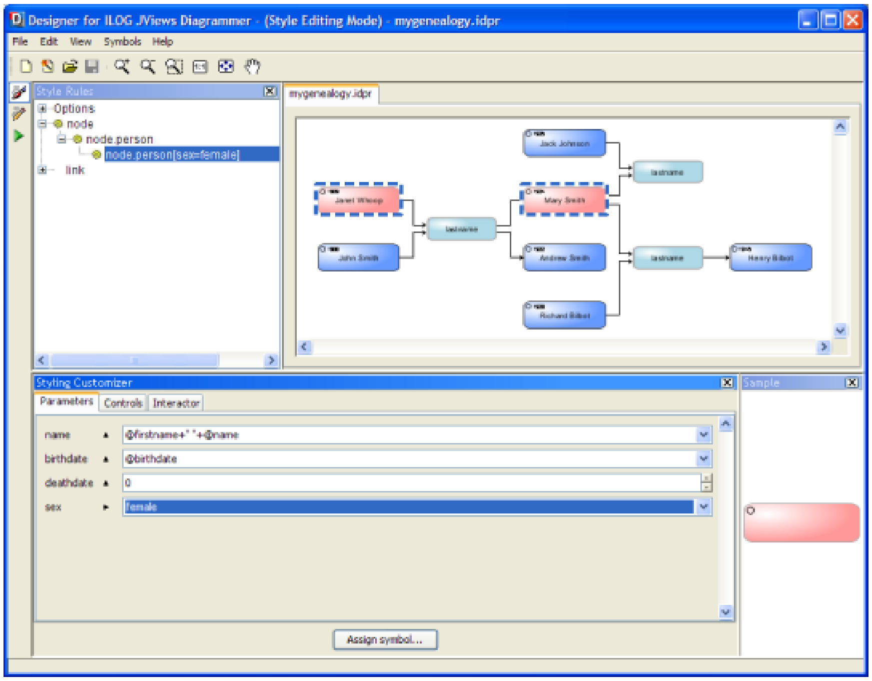
Viewing the data model as a diagram

Once the data model is filled, you can keep the default style sheet or select another style sheet to display the diagram. At this stage, you can preview your diagram in the Designer.

Viewing style rules

Once you have chosen a style sheet, you can start to create style rules to refine the graphical representation of your data model.

The Designer presents the style rules in a tree, from which you can manage and edit rules, see the following figure.



Options
node
node.person
node.person[sex=female]
link

name	@firstname+" "+@name
birthdate	@birthdate
deathdate	0
sex	female

Style rules in a tree view (top left) in the Designer

Each rule displays its conditions in the tree so that you clearly see which objects in the data model are matched by the rule. If the conditions become too long for easy reading, you can define custom rule names instead.

The graphical properties set by the rule are displayed in a panel called the Styling Customizer.

Creating style rules

When you create a new rule, a natural-language editor helps you to specify the selector part by proposing conditions based on your data model.

Once the conditions are defined, you use the Styling Customizer panel to specify the graphic effects to display when the conditions are satisfied for a node or a link.

Configuring the layout

Within the Designer, you can access the graph layout facilities in a wizard that helps you to choose layout algorithms, and then work in a Styling Customizer panel that lets you tune the parameters of the selected algorithms. The result is instantly visible in the preview window.

Note: Graph layout features are only available if you have purchased a full JViews Diagrammer license.

Defining a background

You can add a background to the diagram—bitmap or raster.

If you need a geographic map and you want the nodes to be positioned according to their longitude and latitude, you will need to use IBM® ILOG® JViews Maps with JViews Diagrammer. The Designer lets you integrate maps from IBM® ILOG® JViews Maps into your project.

Saving the project

When you have set up the look and feel of your diagram, you can save your project to be loaded in your application.

Controlling the diagram in an application

JViews Diagrammer comes with a set of classes designed to ease the development of Swing GUIs. These classes allow end users to control one or more diagrams in an application.

Actions

More than sixty predefined Swing actions call the main methods of JViews Diagrammer. They include actions like select, zoom, pan, undo, redo, save, paste, print, print preview, and many more.

Toolbars and menus

JViews Diagrammer provides Swing JToolbars and Swing JMenus that contain buttons linked to JViews Diagrammer actions. A few predefined toolbars and menus such as the Edit toolbar (`EditBar` class) and the Edit menu (`EditMenu` class) are provided to accelerate your development.

Overview

The overview displays the entire diagram within a small area. It has a movable and resizable rectangle which indicates the area of the diagram that is visible. The end user can move and resize the rectangle to pan the main view or to change its zoom level.

Tree and table

JViews Diagrammer proposes an alternative way to view and select nodes and links through a Swing JTree. This view complements the diagram view.

The data model and the data properties of the model objects can be viewed and modified with a Swing JTable.

Property sheets

When double-clicked, a graphic object can display its data properties in a property sheet provided by JViews Diagrammer. Through this property sheet, the end user can edit the data properties of the graphic object. When a style rule is selected, it can display the styling properties in a property sheet as well as in the tabbed panes of the Styling Customizer.

Application

If you need to prototype or to develop fast, you can use the prebuilt JViews Diagrammer application, which already contains toolbars and menus, palettes, property sheet, overview, and more.

Advanced configuration

At some point you need to develop one of the following types of application: a Java™ Swing application, an applet, or a servlet. In this application, you create an `IlvDiagrammer` object, you load the project generated by the Designer, and you add the Swing or DHTML toolbars and menus to control the diagram with buttons like zoom, pan, edit, and so on.

For prototyping, you can decide to start with the prebuilt JViews Diagrammer application that only requires a loaded project to run.

Customizing the application

You may well have precise requirements for your application that cannot be satisfied just with the style sheets and the prebuilt behavior of JViews Diagrammer. This is the case if your application requires specialized interactions or if you had to implement the data model interface to connect to your application data.

With the JViews Diagrammer SDK, you have a comprehensive API to use or extend the Java classes involved in the creation of your diagram. You can basically access all the entities that play a role in and around the diagram: the SDM engine, the data sources, the renderers, the grapher, the views, the interactors, the composite graphics, and the graphic objects themselves.

Some of these classes belong to IBM® ILOG® JViews Framework, which provides low-level graphics services. With IBM® ILOG® JViews Framework, you have in-depth control of the underlying graphical system that helps you to go beyond the limits of the JViews Diagrammer diagram component.

The IBM® ILOG® JViews Framework path

It could happen that you are not satisfied with using a JViews Diagrammer diagram component to build a diagram. You may prefer to manage the drawing process, the synchronization between the graphics and your data, the refresh modes, and so on, for yourself.

If this is your choice, if you do not like high-level components that hide the graphics complexity or do not want the styling mechanism provided by JViews Diagrammer, the JViews Diagrammer product remains a great choice for you: you can find everything you want in IBM® ILOG® JViews Framework and the Graph Layout package, which are part of JViews Diagrammer. (Graph layout features are available only if you have purchased a full JViews Diagrammer license.)

However, you can still use the powerful symbols you may have designed using the Symbol Editor. By default, symbols are based on CSS and are interpreted at run time. But you can also generate the Java source code corresponding to some or all of your symbols (see Using the Symbol Compiler). In this case, generated symbols are JavaBeans™ and subclasses of `IlvCompositeGraphic`. The JViews Framework allows you to instantiate and fully manage such objects in your applications. You will have more control over the way objects are created, deleted, and animated.

With the JViews Framework approach, you can manage the grapher class yourself, you can create your own graphic objects and links that you place in the grapher, and you can apply graph layout algorithms from JViews Diagrammer to lay out the diagram.

This alternative approach was taken by thousands of IBM® ILOG® JViews developers before JViews Diagrammer was put on the market. However, it is likely that you will have to code many services that are prebuilt for you in JViews Diagrammer. Your results can be similar, and you will surely be able to manage the diagram very precisely, but your project will be more costly.

Managing dashboards

Describes the two types of processes provided to manage dashboards.

In this section

Overview
Gives a short introduction on dashboards.

Direct data feeding
Describes one of the two ways to manage a dashboard.

Dashboard introspection
Describes the second way to manage a dashboard.

Overview

A dashboard is a graphical component composed of IBM® ILOG® JViews symbols arranged on an editable background. It is used to display business or system critical information graphically.

Dashboards are created with the Dashboard Editor. They are made of background graphics and active symbols stored using an XML description (usually with the `.idbd` extension).

JViews Diagrammer contains a set of utility components to manage dashboards at a high level. The following typical services are provided:

- Loading dashboards
- Listing the components of a dashboard
- Listing the parameters for each symbol used in the dashboard
- Getting information about each parameter, such as type, default value, and so on

For consistency, a dashboard (`IlvDashboardDiagram`) is derived from the `IlvDiagrammer` class. As a consequence, all the functions provided by SDM are available. When a dashboard is loaded from a dashboard file, a model is automatically created and accessible from within the application. Additional information specified on the dashboard’s symbols (such as parameters and other properties) is available through the `IlvDashboardSymbol` class.

The typical processes for managing a dashboard are twofold:

- *Direct data feeding*
- *Dashboard introspection*

Direct data feeding

In this process, the dashboard is loaded from its description file. Utility methods allow you to retrieve symbols of the loaded dashboard, and then to set particular values for the properties. When properties have been mapped to some specific parameters in the Dashboard Editor, the model already contains the corresponding properties to be directly set on the symbols. The following is an example of the code to write when you manage a dashboard through direct data feeding.

Managing a dashboard through direct data feeding

```
IlvDashboardDiagram _dashboard;
IlvDashboardContext context = new IlvDashboardContext();
// create the dashboard
_dashboard = new IlvDashboardDiagram(context);
String path = "data/dashboard.idbd";
// ......
url = new URL("file:./" + path);
// ......
// Load the dashboard
_dashboard.readDashboard(url);
// Retrieve a given symbol
Object symbol = _dashboard.getObject("my_symbol");
// Set a symbol property
_dashboard.setObjectProperty(symbol, "my_value", new Integer(123));
```

The application **<installdir>/jviews-diagrammer86/samples/dashboard/bam/index.html** is provided as part of the JViews Diagrammer demonstration software to illustrate how to create dashboards that are directly managed by Java™ code.

Dashboard introspection

In this process, the dashboard is also loaded from its description file, but the application is able to dynamically discover the content of the dashboard and manage the association between the symbols and the custom data contained in the application. In this case, you have to take advantage of the iterators provided by the `IlvDashboardDiagram` class and the metadata description available at the level of each symbol.

This process allows you to create very generic applications based on content (that is, dashboards created with the Dashboard Editor) instead of code. The same runtime engine is able to connect data to symbols from their description, manage navigation, animation, and so on.

The application **<installdir>/jviews-diagrammer86/samples/dashboard/tunnel-monitoring/index.html** is provided as part of the JViews Diagrammer demonstration software to illustrate this generic approach with a simulator that is able to compute and deliver a set of active values, and dashboards that are dynamically connected to these active values, purely based on the mapping information described in dashboard files.

For greater flexibility and also to allow you to create more advanced dashboards, the Dashboard Editor is extendible. This allows you to create dashboards and symbols with the right level of information expected by your runtime application. Please refer to the Using the Dashboard Editor documentation for more information about how to customize the Dashboard Editor.

For more information, see the Using the Dashboard Editor user documentation, the BAM Dashboard and Tunnel Monitoring samples, and the `ilog.views.dashboard` package in the Java API documentation.

Integration and deployment

There are two interesting options for delivering rich visual applications that can be deployed to multiple targets:

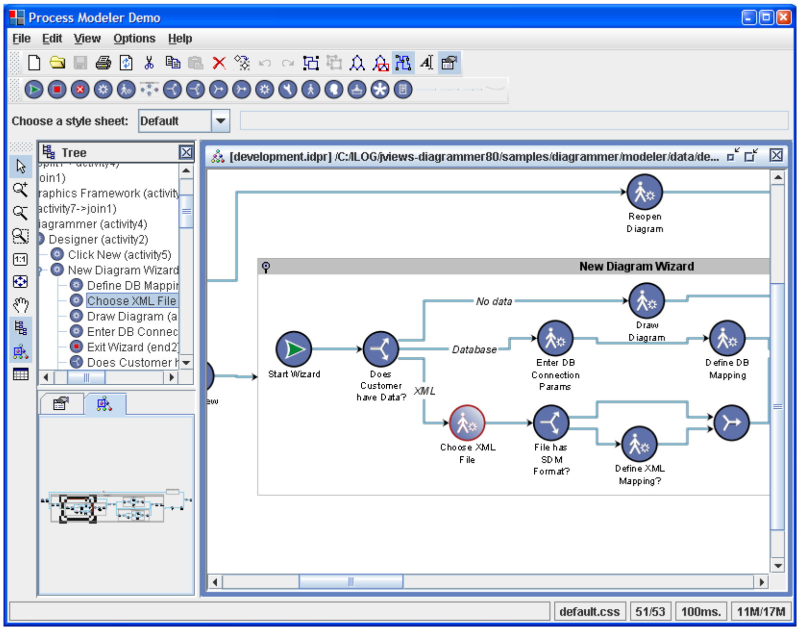
♦ The first option consists in minimizing the amount of code written. For example, because the interface is primarily based on descriptive reusable content (such as Symbols, CSS, XML, Dashboards), the task to finalize the application for a given platform is reduced.

♦ The second option consists in using a model-driven architecture to automate the creation of user interfaces directly from data models. This very systematic approach is particularly suitable for diagrams and data-centric applications.

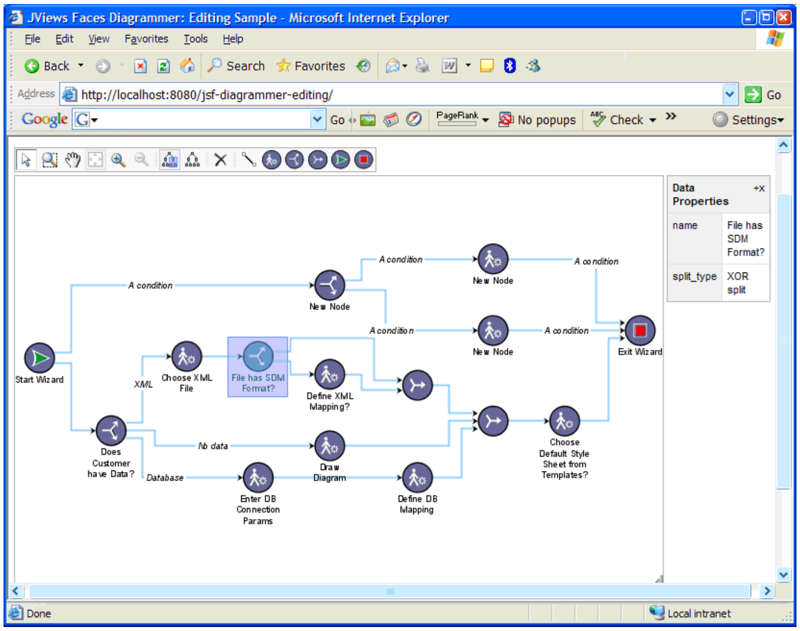
These techniques require a set of portable visual components that can deliver graphical content on multiple platform targets, and design tools that create reusable visual entities and the specifications for displays driven by underlying data. The importance of design tools goes beyond the need to reduce the coding part of an application, and offers an opportunity to provide different tools for the different roles in the development chain, and eventually create new ones. For example, a graphics designer can provide attractive content for a user interface without necessarily being involved in technical development. Also, an application administrator can enrich the application without modifying the core of the system.

JViews Diagrammer offers dedicated portable components for diagrams, dashboards, and generic Human Machine Interfaces (HMI). The design tools (Symbol Editor, Dashboard Editor and Designer) simplify the development process and minimize the amount of code to write. Development time is spent primarily on creating descriptive content, and the coding part is limited to integration and data management. At run time, different families of components, including Swing, Eclipse™ , and Web are used for integration. For the Web side, there are dedicated JavaServer™ Faces components that can reside on a Web server and generate an interface for a browser. By mixing images and JavaScript™ /DHTML code, the components deliver content for either traditional Web pages or portals that implement the JSR 168 standard. They are also able to deal with asynchronous requests that manage Ajax behavior and minimize page refreshes.

For more information, please refer to the Building Web applications user documentation.



A business process diagram deployed as a rich client


JViews Faces Diagrammer: Editing Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/jsf-diagrammer-editing/
Go
Data Properties
name File has SDM Format?
split_type XOR split
A condition
New Node
Start Wizard
XML
Choose XML File
File has SDM Format?
Define XML Mapping?
Exit Wizard
Does Customer have Data?
No data
Draw Diagram
Database
Enter DB Connection Params
Define DB Mapping
Choose Default Style Sheet from Templates?
Done
Local intranet

The same business process diagram deployed as a Web/Ajax interface

By generalizing the usual software component approach and systematically adding design tools and descriptive content, JViews Diagrammer addresses a large number of visual requirements and deployment to multiple platforms. With more precise roles in the development chain, applications can be enriched by creating new content that is dynamically loaded without modifying the code base. Another interesting aspect is the ability to reuse graphical content, such as visual symbols or look and feel definitions, from one application to another independently from code. In the end, developers have more freedom to build and deliver interactive applications that end users will enjoy using.

Index