The yWays Library Manual

The yWays Library Manual

Preface

The main difference between the visualization of biochemical pathways and the visualization of other data arising in biochemistry is that the visualization should rather exhibit a topological structure than numerical data. The yWays library provides a rich set of tools for the visualization of biochemical pathways. The main features of yWays are:

- Basic datastructures to handle biochemical pathways.
- A Java Bean for graphically displaying and editing biochemical pathways.
- Automatic layout algorithms.
- Import/export filters for different file formats.
- The implementation is in pure JAVA 2.

This manual contains a description of all data structures, algorithms and GUI Elements provided by the yWays library.

This manual is structured as follows: First the architecture of yWays is discussed with its main components: the core and the subsystems. In the following chapters these components are described in detail.

Chapter 1. License Terms and Availability

Any use of yWays requires a license which is distributed by:

YWorks GmbH

<contact@yworks.de>

For the current price list, license terms, available packages, supported platforms, and other information please visit: http://www.yworks.de/yWays

Chapter 2. Getting Started

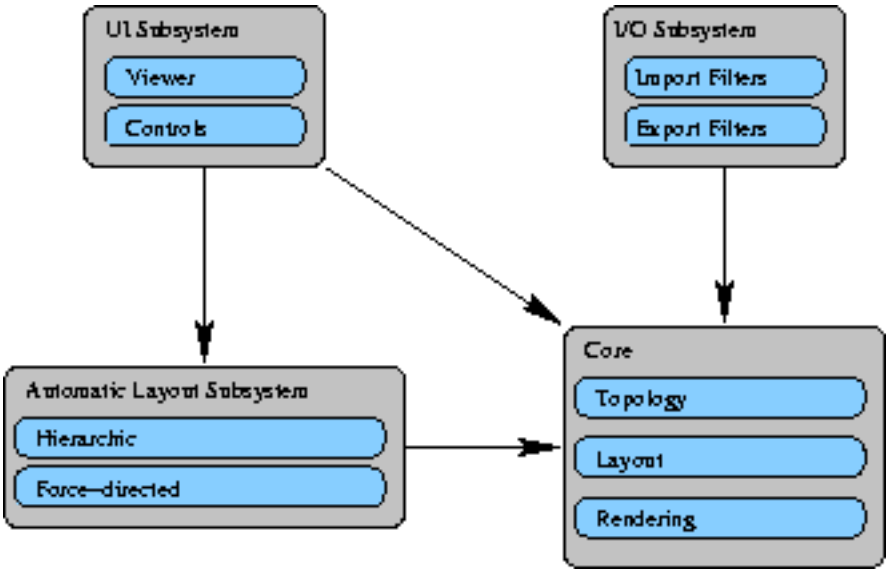
System Requirements

To use the yWays library you must have a JAVA 2 run-time environment installed on your computer. JAVA 2 runtime environments are currently available for Linux, Windows 95, Windows 98, Windows NT 4.0, Solaris, HP-UX und a variety of other machines. Please check the Java Homepage of Sun [http://java.sun.com] for infomations about the availability of Java 2 for your system.

To use the XML import/export capabilities of yWays you need an XML Parser and the JAXP API Version 1.1.

Chapter 3. Architecture of yWays

The yWays software is divided into several parts, which we call *subsystems*, which are based on the *core*. These subsystems are the *UI Subsystem*, the *Automatic Layout Subsystem* , and the *I/O subsystem*.



The Core

The core contains classes and interfaces to describe diagrams of biochemical networks. All subsystems are based on the core.

The chapter " The Core" covers the base subsystem in detail.

The UI Subsystem

The UI subsystem provides a Java Bean to display and edit biochemical networks.

The chapter "The UI Subsystem" covers the automatic layout subsystem in detail.

The Automatic Layout Subsystem

The automatic layout subsystem contains algorithms to calculate the layout of a diagram of a biochemical network.

The chapter " The Automatic Layout Subsystem" covers the automatic layout subsystem in detail.

The I/O Subsystem

The I/O Subsystem handles the serialization of biochemical networks.

The chapter "The I/O Subsystem" covers the I/O subsystem in detail.

Chapter 4. The Core

In this chapter contains a description of the core. The core contains the data structures that are used in yWays to represent biochemical pathways and their visualizations. All classes and interfaces of the core are part of the package `yWays.base` or `yWays.layout`.

To represent a diagram of a biochemical network, yWays follows a layererd approach which defines three different layers of information:

- The *topology layer* defines the structure of the biochemical network. The structure is mainly defined by a set of compounds and and a set of reactions. At this level reactions are defined as a transformation of one set of compounds into another set of compounds.
- The *layout layer* defines a geometric description of the elements of the topology layer. This geometric description contains the coordinates of the geomtetric objects representing the elements of the topology layer.
- The *rendering layer* provides information which defines the rendering of the diagram elements. The type of data stored in the rendering layer depends on the rendering device and the rendering mechanism used.

Every layer is defined by a set of interfaces. For every interface a default implementation is provided, which has the same name as the interface followed by `Impl`.

The Topology Layer

The topology layer defines the structure of the biochemical network. The topological layer of a diagram is defined by an instance of the interface `ReactionNet`.

There are two main types of elements at this layer: *compounds* and *reactions*. Compounds are represented by the the interface `Compound` and are the basic entity. Reactions are represented by the interface `ReactionNet` and are interpreted as a transformation of compounds. Mathematically speaking a compound is a relation over the set of compounds. A reaction partitions the compounds participating in it in three main groups:

- Substrats
- Products
- Catalysts

There is the following chemical interpretation for the above definitions: A reaction corresponds to a chemical reaction in which the substrats are transformed into the products. The transformation is done with the help of some catalysts. If the reaction is reversible, the products can also be transformed into the substrats with the same catalysts.

The groups Substrats and Products are further refined to emphasize the importance of the corresponding compound in a certain reaction. The group substrats can be divided into *main-substrats* and *co-substrats* , the group products into *main-products* and *co-products* .

The Layout Layer

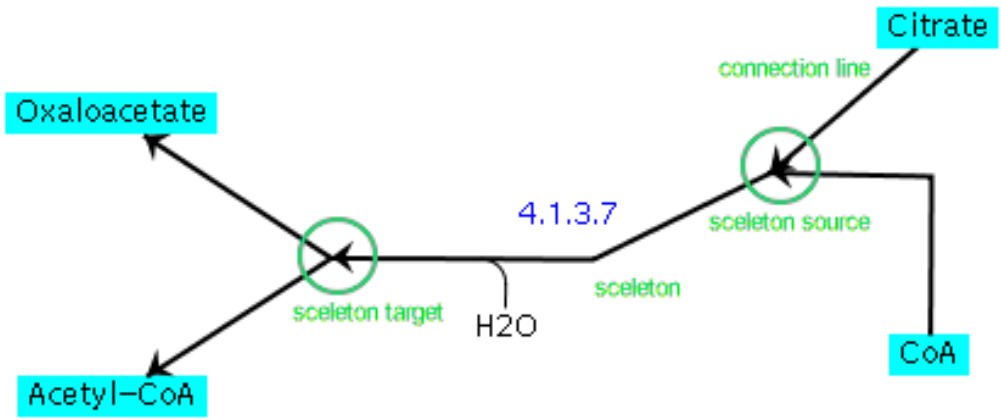
The layout layer defines a geometric description of the elements of the topology layer. The layout layer of a diagram is defined by an instance of the interface `ReactionNetLayout`. This inter-

faces defines a mapping for each element of a `ReactionNet` to an appropiate layout object which contains the geometric description of its graphical representation.

The interface `CompoundLayout` defines the geometric description of the graphical representation of a compound. In yWays the graphical representation of a compound is a shape. Usually the graphical representation is a reactangel containing a text label with the name of the compound. The geometric description of this shape is defined by the coordinates of the bounding box of the shape.

The interface `ReactionLayout` defines the geometric description of the graphical representation of a reaction. The graphical representation of a reaction is much more commplex than the graphical representation of a compound. It is defined as a set of curves with special properties, we refer to this set as *hyperedge*. The hyperedge contains one special curve which is called *skeleton*. One endpoint of the skeleton is called *skeleton source*, the other one *skeleton target*. For each main substrat of the reaction there is a curve connecting the shape of the compound to the skeleton source. We call this curve *connection line*. Similarly each main product is connected with a curve to the skeleton target. Additionaly, for each co-compound of the reaction there is a curve connecting the shape of the co-compound to a point on the skeleton. Note that this points needs not to be the skeleton source, resp. the skeleton target, as it is the case for main compounds.

As an example for the graphical representation of compounds and reactions, the following figure shows a reaction with the main substrats Citrate, CoA and the main products Oxaloacetate, Acetyl-Coa. There is a cosubstrat H2O and a catalyst denoted by the EC number "4.1.3.7".



Rendering Layer

The rendering layer provides informartion which defines the rendering of the diagram elements. The rendering layer of a diagram is defined by an instance of the interface `ReactionNetRenderingInfo`. This interfaces defines a mapping for each element of a `ReactionNet` to an appropiate rendering object which contains the rendering attributes for the element.

The interface `CompoundRenderingInfo` defines the acces to the rendering attributes of a compound.

The interface `ReactionRenderingInfo` defines the access to the rendering attributes of a reaction.

The type of data stored as rendering attribute depends on the rendering engine. Each rendering engine can define its own rendering attributes and store them in the rendering layer.

Chapter 5. The UI Subsystem

This chapter contains a description of the UI subsystem. The UI subsystem provides UI components for displaying and editing biochemical networks. All UI components are Java Swing Components and can therefore easily be integrated in any Swing application. The subsystem is located in the package `yWays.viewer`.

The design of the UI subsystem follows the Model-View-Control design pattern: The model is given by an instance of `ReactionNet`, which is defined the core. The view part is the Java Bean `PathwayView`, which is located in the package `yWays.viewer`. The control part is played by instances of `ViewMode` of yFiles.

The PathwayView Bean

The PathwayView Bean is a Java Swing Component which displays a biochemical network. It has the following properties:

- Zoom level
- View point
- Catalyst visibility
- Co-compound visibility
- Automatic layout algorithm
- Selection state of compounds and reactions

The properties zoom level and view point define the part of the network, which is currently visible. The properties catalyst and co-compound visibility define, if catalysts and/or co-compounds are currently displayed by the component. The property automatic layout algorithm defines an automatic layout algorithm which is responsible for the placament of the diagram elements. Furthermore, some compounds and reactions may be selected, which is denoted by the selection state property.

Chapter 6. The Automatic Layout Subsystem

This chapter contains a description of the automatic layout subsystem. The automatic layout subsystem provides algorithms to calculate the placement of the diagram elements. The automatic layout subsystem consists of the package `yWays.autolayout` and its subpackages.

Automatic layout algorithms are definded by the interface `LayoutAlgorithm`. The class `AbstractLayoutAlgorithm` implements this interface and defines methods for retrieving and storing options of a layout algorithm via `OptionHandlers`. All automatic layout algorithms provided by yWays are derived from this class.

Currently yWays provides two automatic layout algorithms:

- Force-Directed
- Hierarchic

The following sections contain a description of these algorithms.

Force-Directed Layout Algorithm

The force directed layout algorithm simulates a physical model, which consists of weights and forces, and tries to minimize the total energie in the model. The resulting layout usually exhibts well symmetries and clusters in the network. Often a force-directed layout is used to get a first impression from a data set and to identify clusters. It has the disadvantage of producing sometimes overlaps of compounds and more crossings than desired. The force-directed layout algorithm is located in the package `yWays.autolayout.forcedirected`.

Options

The only option currently provided by the force-directed layout algorithm is `PreferredEdgeLength`. The algorithm tries to assign each connection line and the skeleton line this length.

Hierarchic Layout Algorithm

The hierarchic layout algorithm tries to arrange the compounds in a fashion that the irreversible reactions point in one direction. This type of visuallization is especially usefull for metabolic pathways, in which the reactions occur only in one direction. The algorithm partitions the compounds into layers and than arranges the compounds in the layers, such that the number of crossings between different reactions is low. The hierarchic layout algorithm usually produces few crossings and no overlaps. However, if the data is not directed, the reasulting layout may be not satisfactory and a force-directed layout may be more appropiate. The hierarchic layout algorithm is located in the package `yWays.autolayout.hierarchic`.

Options

The hierarchic layout algorithm provides currently the following options: minimal node distance, minimal layer distance and drawing style. These options are discussed in the following sections.

Minimal Layer Distance

This option defines the minimal distance between two consecutive layers. Increasing this value leads to placements which need more space, but choosing this value two small sometimes results in a crowded layout. Usually a value between 20 and 50 is best.

Minimal Node Distance

This options defines the minimal distance between two compounds in the same layer. Increasing this value leads to placements which need more space, but choosing this value two small sometimes results in a crowded layout. Usually a value between 20 and 50 is best.

Drawing Style

The drawing style denotes how the reactions are drawn.

In the *curved* drawing style, the reactions are drawn, such that the diagram uses as few space as possible. This results normaly in a considerable amount of bends.

In the *linear segment* drawing style, each line, i.e. the skeleton and the connection lines of an reaction, has at most 2 bends. This resulting placements have few bends, but are usualy not very compact.

Chapter 7. The I/O Subsystem

This chapter contains a description of the IO subsystem. The IO subsystem handles the serialization of biochemical networks. The subsystem is located in the package `yWays.io`.

Import/Export Handlers

The interface `ImportHandler` defines an import filter for reaction nets. An import filter reads data from an `InputStream` and returns an instance of `ReactionNetDiagram`.

The interface `ExportHandler` defines an export filter for reaction nets. An export filter takes an instance of `ReactionNetDiagram` and serializes it to an `OutputStream`.

Both, `ImportHandler` and `ExportHandler`, are derived from the interface `FileHandler` which defines a file extension and a short description for a handler. The interface `IOHandler` defines a handler which is import and export handler at the same time.

yWays Default Handlers

There is currently one import/export handler and two export handler provided by yWays.

The class `XMLIOHandler` imports and exports data in an XML format. The format is discussed in detail in Appendix " The yWays XML Format" The `XMLIOHandler` is based on JAXP 1.1, make sure that the JAXP 1.1 libraries are included in the classpath when this handler is used.

The class `GIFExportHandler` provides an export handler which creates a GIF image from a reaction net.

The class `GMLExportHandler` provides an export handler which creates an GML file from a reaction net. The Graph Modelling Language (GML) is a format supported by several graph drawing tools, i.e. LEDA Graphwin and yFiles.

Chapter 8. Practical Hints

This chapter contains some hints for using yWays and helps you to avoid some pittfals.

Defining a network with the help of `ReactionNetImpl`

With the help of class `ReactionNetImpl` you can create your own instance of `ReactionNet`.

First you have to create instances of `CompoundImpl` to represent the compounds in your network. Add all instances of `CompoundImpl` which are main-product or main-substrat in one ore more reaction to the `ReactionNet` instance with the method `addCompund()`.

Warning

Each compound that you add in this way to the network will be returned by the `compounds()` method and will therefore define a node in the diagram. Make sure that you add each instance of `CompoundImpl` only once and to create only one instance of `CompoundImpl` for each main-compound in the network.

To create a reaction, first store the products and substrats of the reaction in a Collection. Create an instance of `ReactionImpl` with these collections as argument. With the methods `addCoProducts()` and `addCoSubstrat()` co-compounds can be added.

Warning

Each instance of `Compound` that you add in this way to the network will be returned by the `cocompounds()` method and will therefore define a node in the diagram. For this reason `ReactionNetImpl` supports for the moment only co-compounds which are connected to exactly one reaction. So make sure that you create always a new instance of `CompoundImpl` before you add it this way to a reaction.

To define a catalyst for a recation, simply create an instance of `CompoundImpl` and add it via the `addCatalyst()` method to the reaction. Finally you just have to add the reaction to the network with the `addReaction()` metthod.

Appendix A. The yWays XML Format

In this appendix gives a short overview over the yWays XML format. Files in the yWays format are recommended to have the extension `.xml`. The DTD of the yWays XML format has the following URI:

http://pathways.informatik.uni-tuebingen.de/dtds/ReactionNetDiagram.dtd.

A file in the yWays XML format must begin with the following header:

```
<?xml version="1.0"?>
<!DOCTYPE PathwayDiagram SYSTEM
     "http://pathways.informatik.uni-tuebingen.de/dtds/PathwayDiagram.dtd">
```

The yWays XML format follows the layered approach of the yWays library. Currrently the topological layer and the layout layer are supported. The global structure of an yWays XML file is as follows:

```
<ReactioNetDiagram id="id" style="KEGG">
  <!-- Topology Layer -->
  <ReactionNet>
          ...
  </ReactionNet>
  <!-- Layout Layer -->
  <ReactionNetLayout>
          ...
  </ReactionNetLayout>
<ReactionNetDiagram>
```

Topology Layer

The topology layer consists of two lists, one containing the compounds of the reaction net, and one containing the reactions.

```
<ReactionNet>
  <CompoundList>
          ...
  </CompoundList>
  <ReactionList>
        ...
  </ReactionList>
</ReactionNet>
```

Each compound in the compound list must have a unique id and a name. The id of the compound must begin with a letter. The following fragment shows an example:

```
<Compound id="C4">
  <Name>Benzylalcohol</Name>
</Compound>
```

Each reaction in the reaction list may have a unique id and lists of substrates, compounds, co-substrates, co-compounds and catalysts. Each entry in the lists refers to the definition of the compound by its id. The id of the reaction must begin with a letter. The following fragment shows an example:

```
<Reaction id="R233">
```

```
  <Substrat ref="C274"/>
  <Substrat ref="C276"/>
  <Product ref="C278"/>
  <Product ref="C279"/>
  <Catalyst ref="C280"/>
</Reaction id="R233">
```

Layout Layer

The layout layer is subject to change and is therefore not documented for the moment. For further information refer to the yWays XML format DTD.