yDoc 2.2 User's Guide

Contents

1. Introduction

Welcome to the *yDoc User's Guide*.
This guide explains how to use *yDoc*, a javadoc extension (basically a doclet/taglet bundle) that provides

- functionality to auto-generate, customize, and include UML diagrams in the API documentation of your Java products
- a filter interface which allows for custom suppression of class, field, or method documentation
- an easy to use mechanism for defining simple custom tags via XML

2. System Requirements

yDoc 2.2 requires JDK 1.5.0 installed on your system.
To view UML class diagrams in SVG or SVGZ format, you either need a browser with native SVG support or a SVG plug-in. For Microsoft Internet Explorer, you can download one such plug-in from the Adobe website.
If you want to run yDoc under Unix/Linux operating systems, you need to have an X server installed and running, since yDoc makes use of the java awt and/or swing packages (for UML generation only).

3. Installing yDoc

Unzip the yDoc archive (ydoc-2.2-jdk1.5.zip) into a directory of your choice. It will create a lib/, a doc/, and a resources/ subdirectory.
The lib directory contains the java classes you need to run the ydoc expansion as jar libraries.
The doc directory contains the yDoc User's Guide in HTML and PDF format, the DocFilter API Documentation in HTML format, and several usage samples.
The resources directory contains various property and configuration files which you can use to customize the behaviour of the yDoc expansion. See the property/configuration files for more details.

4. Running yDoc

Basically you run javadoc. The only difference is, that you tell javadoc to use the facilities provided by yDoc as a plug-in.
Read on for detailed information on how to do that. You can skip this part, if you are already familiar with using custom doclets for javadoc.

Using custom doclets

We recommend running javadoc either using a build tool such as ANT (version 1.5.2 or better) or directly from commandline. Before running javadoc from commandline, put your commandline options into a file called "options" and run javadoc by invoking
    javadoc @options @packages
where "packages" is the filename of a file containing the java packages you want to be documented.

See yDoc Quick Start for simple examples on how to use yDoc.

For detailed documentation on the javadoc options, see the javadoc tool homepage at http://java.sun.com/j2se/javadoc/index.html.

Using the yDoc doclet

To use the yDoc expansion the following options are especially important:

- **-sourcepath** *sourcepathlist*
  The *sourcepathlist* must contain the path to the Java sources, for which you want to generate the API documentation.
- **-docletpath** *docletpathlist*
  This option tells javadoc where to look for the yDoc expansion.
  The *docletpathlist* must contain the path to the library ydoc.jar
      <ydoc_install_dir>/lib/ydoc.jar
  and the resources directory
      <ydoc_install_dir>/resources
  Important:
  If you want to use the yDoc UML generation, *docletpathlist* must also contain the path to the library class2svg.jar
      <ydoc_install_dir>/lib/class2svg.jar,
  to your *compiled*, *unobfuscated* Java class files (*.class), for which you want to generate the API documentation, and to all libraries needed to compile your Java source files.
  You do not need the class2svg.jar library to use yDoc's other features.
- **-doclet ydoc.doclets.YStandard**
  The -doclet ydoc.doclets.YStandard option finally tells javadoc to actually use the YStandard doclet, which is the core class of the ydoc expansion.

See yDoc Features for more information on (custom) commandline options and on

how to use the specific capabilites of yDoc.

A sample options file on a Win32 operating system could look like this:

```
-d <destination directory>
-sourcepath <source directory>
-breakiterator
-generic
-umlautogen
-author
-docletpath <YID>/lib/ydoc.jar;<YID>/lib/class2svg.jar;<YID>/resources;<some path>/myapp.jar
-doclet ydoc.doclets.YStandard
-filterpath <YID>/lib/ydoc.jar
-filter ydoc.filters.ExcludeFilter
-tagletpath <YID>/lib/ydoc.jar
-tag param
-tag return
-tag see
-ytag y.uml
```

where **`<YID>`** denotes the **`<ydoc_install_dir>`**.
On Unix/Linux operating systems, you will have to use " **:** " as a path separator
instead of " **;** ".

yDoc Quick Start

This section demonstrates how to use yDoc to generate a Javadoc page of a sample
class that will automatically include an UML diagram depicting that class.

- **yDoc from commandline**

    Look in **`<YID>/doc/examples`** for sample options files and sample Java
    sources to test yDoc.
    All you need to do is invoking javadoc in **`<ydoc_install_dir>`** with either
        javadoc @doc/examples/options.sample.linux
    or
        javadoc @doc/examples/options.sample.win32
    depending on your operating system.

- **yDoc in ANT**

    Using ANT 1.5.2 or better, you can use ANT's javadoc task to run yDoc.

    Look in **`<YID>/doc/examples`** for a sample ANT build file and sample Java
    sources to test yDoc.
    All you need to do is invoking ant in **`<ydoc_install_dir>`** with
        ant -buildfile doc/examples/build-sample.xml test-ydoc

The generated API pages can now be found in
`<ydoc_install_dir>/doc/api/examples`. Have a look at these pages with an
SVG enabled browser.

Note that the generated UML diagrams are in PNG format. If you want to generate
the uml diagrams in a different format (SVG, SVGZ, GIF, JPG) simply change the
value of **uml_file_format** in the yDoc configuration file
`<ydoc_install_dir>/resources/ydoc.properties` accordingly.

The next tutorial step would be to look in **`<ydoc_install_dir>/doc/examples`**
for the sample option/build files and sample Java sources which have been used in
this example. Once you understand the options and tags, you are ready to use yDoc

in your own project.

5. yDoc Features

Generating UML class diagrams

yDoc will generate UML diagrams, if one or more of the following commandline options are used:

- **-umlgen**
- **-umltypegen**
- **-umlpackagegen**
- **-umloverviewgen**
- **-umlautogen**

All UML diagrams feature hyperlinks for the displayed packages, types, and type members, which allow direct access to the corresponding documentation.

See Custom command line options for additional details.

Important:
yDoc uses the Java Reflection API to generate the class diagrams, therefore you need to specify the path to your *compiled, unobfuscated* Java class files (*.class) and to all libraries needed to compile your Java source files in the -docletpath option. Your class files may be located in a jar file.

General Layout of UML class diagrams
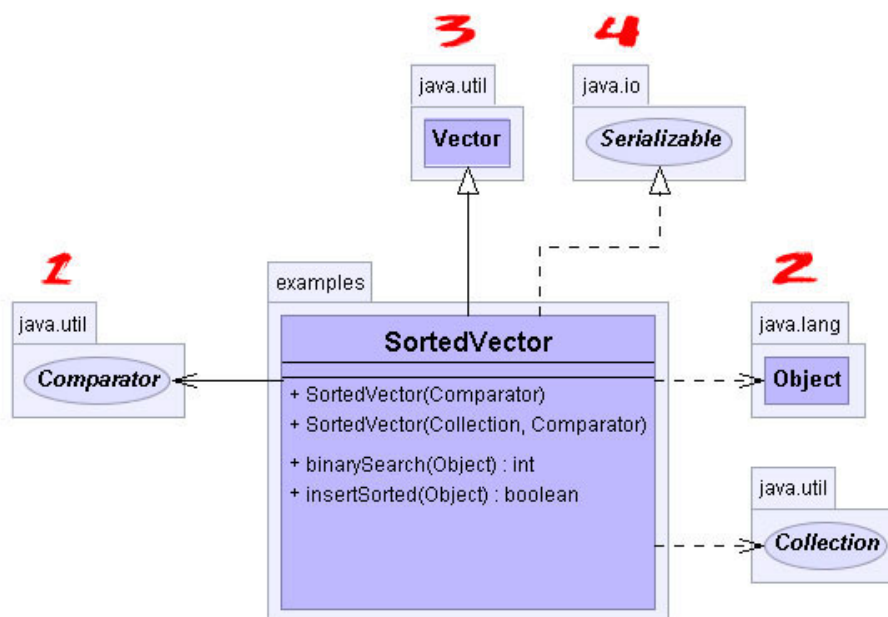
𝟏 **Associations**
structural relationships between a whole and its parts, i.e. *has a* or *instantiates*
Every declared field constitutes an association.

𝟐 **Dependencies**
semantic relationships in which a change to one thing may effect the semantics of the other thing
There are several heuristics as to what constitues a dependency. See UML style.

𝟑 **Generalizations**
specialization/generalization relationships, i.e. *is a* or *subclass/superclass*
For interfaces there may be more than one generalization relationship.

𝟒 **Realizations**
semantic relationships between classifiers, i.e. *interface/implementing class*
For interfaces there are no realization relationships.

Customizing UML class diagrams

Customization is handled via the Java property file `resources/ydoc.properties`. The following settings can be modified:

UML Style
The property **uml_style** determines which *style* to use for UML generation.
A style defines the way a class diagrams looks and is specified in a Java property file following the naming convention <stylename>.style. Style files belong into the `resources` directory.

The yDoc distribution comes with three predefined style files:

- default.style
- original.style
- theBlues.style

You can customize colors (main, border, text), fonts, shapes, and lines by either modifying an existing style file or creating a new one.

The yDoc 2.2 distribution includes StyleEd, an easy-to-use GUI-based style editor, that greatly simplifies customization. The editor is completely written in Java and will run on any Java 1.4.x or Java 5.0 platform. All files needed to run StyleEd are contained in the executable JAR file `lib/styleed.jar`, i.e. double-clicking the file or invoking
java -jar <YID>/lib/styleed.jar
will start StyleEd.

Some of the more exotic settings available for customization include:

- **uml_display_associations**
  Determines whether the associations list for a given class or interface should be displayed.
- **uml_display_dependencies**
  Determines the heuristic approach as to what constitutes a dependency for a class or interface.
  Possible choices are: none, all constructor and method parameters, all parameters and all method return types, and compile-time dependencies in general.
- **uml_display_package_dependencies**
  Determines whether to display transitive dependencies ("shortcuts") in overview diagrams.
  yDoc uses transitive reduction to remove transitive dependencies from the overview diagram graph.
- **uml_display_packages**
  Determines whether the package frames for a given class, interface, and list (association, dependencies, generalization, realization) should be displayed.
- **uml_display_parameters**
  Determines whether constructor and method parameters are listed.
- **uml_exclude_pattern_***
  A comma-separated list of full-qualified classname patterns to exclude from the appropriate relation list of the UML diagram. '?' and '*' can be used as wildcards.
- ***_compact_***
  These properties always refer to elements in the association, dependencies, generalization, and realization list.

For detailed information concerning customization possibilities and examples, please see the style files that come with the distribution.

If yDoc cannot find a specified style file or a specific property in a specified style file, it will use internal defaults which correspond to the values defined in default.style.

SVG display mode and related settings
If you are using SVG as file format for class diagrams, there are several display modes (or policies) available.

The property **svg_display_mode** allows you to set a specific mode.
Among others, the following modes are available:

- FIXED_SIZE
  the diagram will be displayed in a fixed size canvas
- ACTUAL_SIZE
  the diagram will be displayed in a canvas sized to the diagram's actual size
- FIT_TO_SIZE
  the diagram will be scaled to fit into a canvas with fixed width and fixed height
- SHRINK_TO_SIZE
  the diagram will be scaled to fit into a canvas with fixed width and fixed height, unless it already fits

The properties **svg_display_width** and **svg_display_height** determine the reference size for scaling.

The property **svg_display_reserve_minimum** finally specifies whether yDoc should always reserve a canvas at least the size of svg_display_width x svg_display_height even if the class diagram is actually smaller.

yDoc does not support display modes for GIF, JPG, and PNG images. GIF, JPG, and PNG images are always displayed by actual image size.

The property **svg_workaround** activates an experimental workaround for bugs concerning embedded SVG and the Adobe SVG plug-in v3.0 in browsers of the Gecko family. Using this feature turns off hyperlinks for type members.
See also the warning in the next section.

UML file format

The property **uml_file_format** in `resources/ydoc.properties` determines the file format for generated class diagrams.
The following formats are available:

- SVG (Scalable Vector Graphics)
- SVGZ (compressed SVG)
- GIF
- JPG
- PNG

SVG(Z) is the default and primary format, but you will need a svg plug-in for your browser to be able to display the diagrams. For Microsoft Internet Explorer, you can download one such plug-in from the Adobe website.

yDoc uses Batik to generate SVG files.
See the Acknowledgments at the end of this file.

We strongly recommend increasing Javadoc's heap size to at least 256 MB when using GIF, JPG, or PNG as file format in real-life projects. This can be achieved using the -J-Xmx option.
GIF, JPG, and PNG are meant to be alternatives for people who cannot get a svg plug-in, e.g. people working on Unix/Linux platforms.

WARNING:
Although the Adobe SVG plug-in has experimental support for browsers and platforms other than MSIE/Windows as well, installing the Adobe SVG plug-in v3.0 causes browsers of the Gecko family (Mozilla, Netscape 6+7, etc.) to crash on all platforms when displaying HTML with embedded SVG.

There is an inofficial workaround that embeds the SVG file indirectly by wrapping it in a inner frame with <IFRAME> tags.
There are several disadvantages to this procedure though:

- Does not work on Windows XP platforms (e.g. Mozilla 1.2.1 still crashes).
- Prevents access to the browser DOM from JavaScript in SVG for Gecko browsers.
- Only prevents the browser crash, but does not ensure that the SVG image is displayed correctly.
  For some WWW servers, it has been observed, that the SVG source code is displayed instead of the SVG image. It remains unclear, what the reason for this behaviour might be.
  For locally stored HTML the embedded SVG is always displayed correctly.

Using filters

yDoc provides one custom filter, which lets you exclude classes/interfaces, fields, and/or methods from documentation, if their documentation contains an **@y.exclude** tag.
To use this filter, you need to specify the following two commandline options:
  -filterpath **<ydoc_install_dir>/lib/ydoc.jar**
  -filter ydoc.filters.ExcludeFilter

To create and use your own filters, all you have to do is implementing the ydoc.filters.DocFilter interface and register the filter similar to the above example. The mechanism to register filters works similar to the one used for doclets.

DocFilter API

Documentation for the ydoc.filters.DocFilter interface, which comprises the DocFilter API.

Using the XML driven taglet factory

By specifying the -generic option, you can tell yDoc to register simple taglets, which are more powerful than the ones created by the standard -tag option and are defined in the **resources/taglet_definitions.xml** and **resources/taglet_templates.xml** files.
By adding more definitions to those files, you can use/register more simple taglets.

The basic idea is to have template definitions that define taglet behaviour and taglet definitions that define scope, name, and which template to use.

For examples on how to define taglets, see the two mentioned xml files.

Taglet Definitions
The following XML elements are used to define taglets:

- `<taglet>`
  Each of these elements results in the registration of one particular taglet.
  The value of the `name` attribute specifies the javadoc tag for the taglet. The value of the attribute `allowMultipleTags` specifies if more than one appearance of the javadoc tag per doc element is allowed. If not, all but the first tag will be ignored.
- `<usage>`
  Required element that specifies the taglet scope as per the taglet API.
- `<headline>`
  Required root element for `<singular>` and `<plural>`.
- `<singular>`

Required element that specifies the headline for the tag comment if only one
javadoc tag or no `<plural>` element is present.

- `<plural>`
  Optional element that specifies the headline for the tag comment if multiple
  javadoc tags are allowed and present.

In general, it is a good idea to use at least one '.' character in the name of custom
tags to avoid potential conflicts/overrides.

Template Definitions
The following XML elements are used to define templates:

- `<template>`
  Each of these elements results in the creation of one particular template.
  The value of the `name` attribute has to be unique among all templates. It is used
  to reference the template in the taglet definition.
- `<headline>`
  Required element that specifies the HTML code for the headline of the tag
  comment.
  You may specify one parameter sign, i.e. #0.
  You may use a single parameter multiple times, e.g. `<headline>`
  `<![CDATA[bla #0 bla#0bla]]></headline>`
  The element should contain unparsed character data, i.e. `<![CDATA[....]]>`
- `<content>`
  Required element that specifies the HTML formatting for the tag comment.
  The value of the `separator` attribute specifies if and how to break down the
  comment into parameters.
  Possible values are:
  - any single character   breaks the comment at each occurance
                                   of the specified character
  - the token "first-whitespace" breaks the comment at the first
                                   occurance of a whitespace
  - the token "whitespace"   breaks the comment at each occurance
                                   of a whitespace
  - the token "none" (default)   results in one token only, namely the
                                   whole comment
- `<content-item>`
  Required element that specifies the HTML code to wrap the tag comment in. If
  multiple javadoc tag are present for a particular doc element, then one content
  item is created for each tag comment.
  You may specify up to ten parameter signs, i.e. #x, where -1 < x < 10.
  You may use a single parameter multiple times.
  The element should contain unparsed character data.
- `<content-sep>`
  Optional element that defaults to "".
  Its value will be inserted between content items.
  The element should contain unparsed character data.
- `<content-start>`
  Optional element that defaults to "".
  Its value will be inserted directly after the headline, before the first content item.
  The element should contain unparsed character data.
- `<content-end>`

Optional element that defaults to "".
Its value will be inserted directly after the the last content item.
The element should contain unparsed character data.

In general, it is a good idea to use the `<DT>` tag for headlines and the `<DD>` tag for content, since all output generated by javadoc taglets appears in definition lists.

Custom command line options

yDoc provides some custom commandline options:

- **-generic**
  The taglet definitions in **`resources/taglet_definitions.xml`** and **`resources/taglet_templates.xml`** are used to create and register simple taglets.
- **-umlgen**
  UML diagrams will be created and embedded for all documented files with an **@y.uml** tag.
  @y.uml may be used in type, package, and overview documentation.
- **-umltypegen**
  UML diagrams will be created and embedded for all documented classes and interfaces, not only for those with an **@y.uml** tag.
- **-umlpackagegen**
  UML diagrams will be created and embedded for all documented packages, not only for those with an **@y.uml** tag.
- **-umloverviewgen**
  An UML overview diagram will be created and embedded, even if there is no **@y.uml** tag in overview.html.
- **-umlautogen**
  Same as using -umltypegen, -umlpackagegen, and -umloverviewgen in combination
- **-umlfileformat** *formatname*
  Overrides the uml_file_format property in **`resources/ydoc.properties`**
  See the section about UML file formats for a list of supported formats.
- **-umlstyle** *stylename*
  Overrides the uml_style property in **`resources/ydoc.properties`**.
  See the section about customizing UML diagrams for details on styles.
- **-filter** *class*
  Specifies the class file for the filter to be applied. Use the fully-qualified name for *class*. Use the -filterpath option to specify the path to the filter.
  This option works similar to the standard -doclet option.
- **-filterpath** *filterpathlist*
  Specifies the search paths for finding filter class files (*.class). The *filterpathlist* can contain multiple paths separated by the system-dependant path-separator.
  This option works similar to the standard -docletpath option.
- **-resourcepath** *resourcepathlist*
  Specifies the search paths for finding resource files (i.e. style files, taglet definition files, ydoc property file, ydoc license file). The *resourcepathlist* can contain multiple paths separated by the system-dependant path-separator.

- **-license** *file*
  Specifies the path to the license file.

6. Limitations

- The yDoc Evaluation version will only document ten classes.
  If one or more of those are excluded from documentation via the @y.exclude
  tag, they still count against that limit.
- In UML class diagrams generated by the yDoc Evaluation version, the
  associations list and the dependencies list will only display the ten above
  mentioned classes.

7. Acknowledgments

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).
yDoc uses Batik to generate SVG files. Batik is distributed under the Apache Software License, Version 1.1:

```
========================================================================
                The Apache Software License, Version 1.1
========================================================================

Copyright (C) 2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modifica-
tion, are permitted provided that the following conditions are met:

1. Redistributions of  source code must  retain the above copyright  notice,
   this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must
   include  the following  acknowledgment:  "This product includes  software
   developed  by the  Apache Software Foundation  (http://www.apache.org/)."
   Alternately, this  acknowledgment may  appear in the software itself,  if
   and wherever such third-party acknowledgments normally appear.

4. The names "Batik" and  "Apache Software Foundation"  must not be  used to
   endorse  or promote  products derived  from this  software without  prior
   written permission. For written permission, please contact
   apache@apache.org.

5. Products  derived from this software may not  be called "Apache", nor may
   "Apache" appear  in their name,  without prior written permission  of the
   Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS  FOR A PARTICULAR  PURPOSE ARE  DISCLAIMED.  IN NO  EVENT SHALL  THE
APACHE SOFTWARE  FOUNDATION  OR ITS CONTRIBUTORS  BE LIABLE FOR  ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL,  EXEMPLARY, OR CONSEQUENTIAL  DAMAGES (INCLU-
DING, BUT NOT LIMITED TO, PROCUREMENT  OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR  PROFITS; OR BUSINESS  INTERRUPTION)  HOWEVER CAUSED AND ON
ANY  THEORY OF LIABILITY,  WHETHER  IN CONTRACT,  STRICT LIABILITY,  OR TORT
(INCLUDING  NEGLIGENCE OR  OTHERWISE) ARISING IN  ANY WAY OUT OF THE  USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software  consists of voluntary contributions made  by many individuals
on  behalf  of the Apache Software  Foundation. For more  information on the
Apache Software Foundation, please see http://www.apache.org/.
```