



JClass[®] ServerReport 6.2

Programmer's Guide

**© 2013 Quest Software, Inc.
ALL RIGHTS RESERVED.**

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Quest Software, Inc.

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST'S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters

LEGAL Dept

5 Polaris Way

Aliso Viejo, CA 92656

www.quest.com

email: legal@quest.com

Refer to our Web site for regional and international office information.

Trademarks

Quest, Quest Software, the Quest Software logo, AccessManager, ActiveRoles, Aelita, Akonix, Benchmark Factory, Big Brother, BridgeAccess, BridgeAutoEscalate, BridgeSearch, BridgeTrak, BusinessInsight, ChangeAuditor, CI Discovery, Defender, DeployDirector, Desktop Authority, Directory Analyzer, Directory Troubleshooter, DS Analyzer, DS Expert, Foglight, GPOAdmin, Help Desk Authority, Imceda, IntelliProfile, InTrust, Invirtus, iToken, JClass, JProbe, LeccoTech, LiteSpeed, LiveReorg, LogADmin, MessageStats, Monosphere, NBSpool, NetBase, NetControl, Npulse, NetPro, PassGo, PerformaSure, Point, Click, Done!, Quest vToolkit, Quest vWorkSpace, ReportADmin, RestoreADmin, ScriptLogic, SelfServiceADmin, SharePlex, Sitraka, SmartAlarm, Spotlight, SQL Navigator, SQL Watch, SQLab, Stat, StealthCollect, Storage Horizon, Tag and Follow, Toad, T.O.A.D., Toad World, vAutomator, vConverter, vEcoShell, VESI, vFoglight, vPackager, vRanger, vSpotlight, vStream, vToad, Vintela, Virtual DBA, VizionCore, Vizioncore vAutomation Suite, Vizioncore vEssentials, Vizioncore vWorkflow, WebDefender, Webthority, Xaffire, and XRT are trademarks and registered trademarks of Quest Software, Inc in the United States of America and other countries. For a complete list of Quest Software's trademarks, see <http://www.quest.com/legal/trademark-information.aspx>. Other trademarks and registered trademarks used in this guide are property of their respective owners. Other trademarks and registered trademarks are property of their respective owners.

Third Party Contributions

JClass ServerViews contains some third party components (listed below). Copies of their licenses may be found on our website at www.quest.com/legal/third-party-licenses.aspx.

Component	License or Acknowledgement
Apache Tomcat	Apache Foundation License version 2.0
GifEncoder	Copyright 1996 by Jef Poskanzer (www.acme.com).
JDOM	Copyright 2002–2002 Brett McLaughlin & Jason Hunter, all rights reserved.

Programmer's Guide
January 10, 2013
Version 6.2

Table of Contents

Preface	17
Assumptions	17
Examples and Demos	17
API Documentation (Javadoc)	18
Licensing	18
About Quest Software, Inc.	18

Part I: Using JClass ServerReport

1 Learning JClass ServerReport Basics	23
1.1 Key Concepts	23
The Document	23
The Page Templates and Frames	23
The Information Content and Flow	24
1.2 Your First ServerReport-based Application	24
The JCDocument Object	25
The JCPage and JCFrame Objects	26
The JCFlow Object	26
1.3 Units of Measurement	27
Setting a Default Unit of Measurement	27
Converting Units of Measurement	27
Defining Points	28
2 Defining Text-Based Content	29
2.1 The JTextStyle Object	29
Predefined JTextStyle Objects	29
Defining your own JTextStyle Objects	31
2.2 Defining Font Attributes	32
Italic and Bold	32
Underlining	33
Subscripts and Superscripts	33
Background Color	34
2.3 Defining Paragraph Attributes	34
Alignment	34

	Indents	35
	Paragraph Spacing	36
2.4	Defining Tabs	37
	Adding Tabs to a Style	37
	Tab Alignment	37
	Tab Position	38
	Tab Fill	38
2.5	Defining Lists	39
	Understanding Lists and List Items	39
	Creating an Ordered List	40
	Creating an Unordered List	42
	Nesting Lists	44
2.6	Using Macros	44
	Using Numbering Macros	45
	Using Page Total Macros	46
	Creating Custom Macros	47
2.7	Customizing Fonts and Font Mappings	49
	Adding Your Own Fonts	49
	Creating Your Own Font Mappings	52
	Setting TrueType Font Properties	54
2.8	Internationalization	55
	Euro Symbol	56

3 Defining Image-Based Content 59

3.1	Adding Raster-Based Image Content	59
	Image Resolution	59
	Importing Static Image Files	59
	Importing Swing Icons	60
3.2	Adding Vector-Based Image Content	61
	Setting Line Properties	61
	Setting Fill Properties	62
	Drawing Shapes	62
3.3	Importing JClass ServerChart and Other Components	67
3.4	Adding Media Clips	68
	Methods and Parameters	68
	Example	69
3.5	Handling Items that Overflow the Frame	70
	Items Embedded in the Flow	70
	Floating Items	73

	Items Embedded in a Frame	74
3.6	Adding Image-Based Content in XML	77
	Adding Base64-Encoded Images in XML	77

4 Defining Linked Content.99

4.1	Named Locations for Internal Links	79
4.2	Text and Images Links	81
	Using beginHyperlink to Format Hyperlink Text	81
	<hyperlink> XML tag	82
4.3	Bookmarks	82
	BookmarkNode constructor	83
	Example	83
4.4	Table of Contents and Other Reference Lists	85
	Creating Entries with JCContentsListEntry	85
	Creating a Contents List with JCContentsList	88
	Printing the Contents List	89
	Adding a Contents List to the Bookmark Tab	90

5 Defining Table-Based Content91

5.1	Table Structure	91
5.2	JCFlowTable versus JCPageTable	92
	Attribute Overview	93
5.3	JCFlowTable Overview	94
	Using JCFlowTable	94
	Creating a Table with JCFlowTable	95
5.4	JCPageTable Overview	97
	Using JCPageTable	98
	Creating a Table Using JCPageTable	98
	Table Styles	102
5.5	Customizing Tables	107
	Alternating Row or Column colors	108
	Adding Borders	108
	Adding Header Borders	111
	Applying Background Colors	112
	Adjusting the Size of a Table	112
5.6	Customizing Cells	113
	Setting the Vertical Alignment	113
	Defining Cell Margins	115
	Customizing Cell Borders	116

	Spanning Cells	121
5.7	Table Wrapping	122
5.8	Converting Tables	123
	Converting JClass LiveTables	124
	Converting Swing JTables	125
	Converting JDBC Databases	126
5.9	Using Tables With XML	127
	Tags Supported in Cells	127
	Setting Borders on Tables	128
	Spanning Cells and Rows in XML	129

6 Adding Content in Other Ways 131

6.1	Adding Content Using Subframes	131
	Defining a Subframe and its Content	131
	Embedding a Subframe	132
	Floating a Subframe	133
6.2	Adding Content Using XML Fragments	134
	Defining and Using an XML Fragment	135
	List of Supported XML Elements for XML Fragments	138

7 Defining Page Templates 139

7.1	Page Template Formats	139
	Simple Page Template Tags	140
	Full Page Template Tags	144
7.2	XML-Based Page Templates	147
7.3	The JCPage Object	148
7.4	The JCFrame Object	148
7.5	The Interactions of JCFlow and JCFrame Objects	149
7.6	A Sample Template	150
7.7	Page Template Techniques	153
7.8	Rotating Frame Contents	155
7.9	Adding a Watermark	157
	Creating the Watermark Text	158
	Configuring the Text Properties of the Watermark	158
	Specifying the Location of the Watermark	159
	Rotating, Scaling, or Translating the Watermark	159
7.10	Applying Page Templates	160
	Loading External XML Files	161
	Loading XML Strings	161

8	Flowing Content into Your Document	163
8.1	Flow Frames Versus Static Frames	163
8.2	The JCFlow Object	164
	Constructors	165
	Understanding How JCFlow Prints Content	165
8.3	Body Flow	166
8.4	Front Matter Flow	167
	Understanding How Flow Affects Contents Lists	167
	Programming the Front Matter Flow	168
8.5	Flows Example	168
9	Rendering Your Document	175
9.1	JCDocument Object	175
	Printing to a File	175
	OutputPolicy and FlushPolicy	175
	PDF Compression	176
	Printing Large Documents	176
	Printing PDF documents from the command line (UNIX)	176
	The JCPDFPrinter Object	177
	Defining RTF Behavior for Unsupported Features	177
	RTF Output Compatibility	178
9.2	Listening for JClass ServerReport Events	178
10	Setting Document-level PDF Features	181
10.1	Metadata	181
10.2	Security	183
	The RC4 Encryption Provider	183
	Setting Encryption and Password Security Options	183
	The JCPDFSecurity Object	184
10.3	Accessibility and Structured PDF Documents	185
	Adding Tags	186
	Tables in Structured PDF Documents	187
	Creating a Structured PDF Document	188
11	Adding Formulas to JClass ServerReport	189
11.1	Introduction	189
11.2	util.formulae's Hierarchy	190
11.3	Expressions and Results	191

11.4	Math Values	191
	MathScalar	192
	MathVector	192
	MathMatrix	193
11.5	Operations	194
	The Defined Mathematical Operations	194
	Reducing Operations to Values	196
11.6	Expression Lists	197
11.7	Exceptions	198
11.8	Using Formulas in JClass ServerReport	198
	Performing a Mathematical Operation on a Range of Cells	198

Part II: Using JClass ServerReport with XML

12	JClass ServerReport XML Tutorial	203
12.1	Introduction	203
	Running the Tutorial	204
12.2	Lexicon	205
12.3	Overview of the XML changes	205
12.4	Starting the XML Document	206
12.5	Creating a Page Template	207
	Page Template and Flow	207
	XML Document Shell	209
	Advanced Page Template	209
12.6	Creating Sections	210
	Adding a New Section	210
	Flowing Text into Sections	210
	Flowing Text Into the Header	211
	Flowing Text Into the Footer	211
12.7	Adding and Formatting Text	211
	Defining different text styles	211
	Applying text styles	212
	Applying XML style tags	213
12.8	Adding Hyperlinks	213
	Creating external links	214
	Creating internal links	214
12.9	Using Macros	215
	Setting page numbers with macros	215

12.10	Defining Draw Styles and Adding Tables	215
	Defining Draw Styles	215
	Creating a Table	216
12.11	Triggering an External Java Class	218
	Using external Java code	218
	ExternalTutorialHandler source code	219
12.12	Adding Metadata and Lists	221
	Adding Metadata	221
	Adding Lists	221
12.13	Adding Front Matter	224
	Defining the Page Templates for the Front Matter	224
	Starting and Ending the Front Matter Flow	225
	Adding a Title Page	225
	Adding a Table of Contents	226
	Adding a Foreword	228
12.14	The Completed XML Document Code	229
13	XML and JClass ServerReport	237
13.1	Specifying Other XML Files	238
	Applying a page-template From XML	238
	Applying text-styles From XML	239
	Applying draw-styles From XML	240
	Applying symbol-styles From XML	241
	Applying table-styles From XML	241
	Applying Scope to XML Styles	242
13.2	Using the external-java-code Tag	242
	ExternalCodeHandler Interface	243
13.3	Using JClass ServerReport to Convert XML to PDF or RTF	243
	Overview of XML Parsing	243
	Requirements of Your XML Parser	244
	Creating a Document with XML	244
	Loading Document XML Files	245
	Loading Document XML Strings	246
	Configuring an XML Input Source To Create a PDF or RTF	246
13.4	Using JClass ServerReport to Convert from XML to HTML	246
	Extensible Style Sheets Language (XSL) Background	247
14	JClass ServerReport DTD Tags	251
14.1	Structured Versus Linear Usage	251

	Structured	251
	Linear	251
14.2	Document Tags	252
	alt-flow	253
	bold	253
	bookmark	254
	bookmark-tree	254
	cell	255
	chart-data	257
	column-info	258
	contents-list	259
	contents-list-entry	260
	current-text-style	262
	default-text-style	262
	doc-frame	263
	document	263
	document-properties	264
	embed-chart	265
	embed-image	267
	embed-media-clip	269
	external-java-code	271
	float-chart	272
	float-image	274
	float-media-clip	276
	flow	278
	flow-table	278
	footer-frame	281
	front-matter	282
	header-frame	283
	header-table	284
	horizontal-rule	285
	hyperlink	286
	italic	287
	list	287
	list-item	289
	macro	289
	mark-location	290
	meta-data	291
	new-column	291

	new-line	292
	new-page	292
	new-paragraph	292
	new-section	293
	output-contents-list	293
	overflow-rules	294
	page-table	296
	paragraph	300
	poster	300
	row	301
	section	302
	space	302
	tab	302
	unsupported-operation	303
	use-text-style	304
14.3	Page Template	304
14.4	Text Styles	304
	text-styles	305
	text-style	305
	tab-stop	307
14.5	Draw Styles	308
	draw-styles	308
	draw-style	308
14.6	Symbol Styles	309
	symbol-styles	310
	symbol-style	310
14.7	Table Styles	311
	table-styles	311
	table-style	312
	alternate	314

Part III: Using JClass ServerReport Designer

15	JClass ServerReport Designer	317
15.1	Background Information	317
15.2	Launching JClass ServerReport Designer	317
15.3	The Tool Bar	318
15.4	The File Menu	319

Designing a new page template	319
Designing a new document	320
Designing new document styles	320
Designing a new text style	320
Designing a new draw style	320
Designing a new table style	320
Opening a ServerReport XML file	320
Saving a document or document styles	321
Specifying the file encoding type	321
Specifying the page template type	321
Setting your preferences	322
15.5 The Help Menu	324

16 Using JClass ServerReport Designer 325

16.1 Basic Steps for Creating a ServerReport Document	325
16.2 Common Procedures	326
Creating a page, frame, text style, tab, draw style, or table style	326
Cloning a page, frame, text style, tab, draw style, or table style	326
Deleting a page, frame, text style, tab, draw style, or table style	326
Naming a page, frame, text style, draw style, or table style	327
Changing the unit of measurement	328
Changing the size of a page or frame	328
Changing the location of the page or frame	328
Changing the color	328
16.3 Preview	331
Previewing a document	331
Previewing all	332
Previewing the current page template or a current style	332
Exporting the PDF	333

17 Page Templates 335

17.1 One Page Tab	335
Setting headers and footers in Simple page templates	338
Changing a page's orientation	338
Changing a page's flow	339
Changing the orientation of a frame's contents	339
Changing the number of columns in a frame	339
Changing a frame's margins	339
Changing a frame's border	340

	Changing a frame's flow	340
17.2	All Pages Tab	341
	Changing a page's flow	342
	Changing the visibility of the flow lines	342
18	Text Styles	343
18.1	The Sample Area	344
18.2	Designing Text Styles	344
	Changing line and paragraph spacing	344
	Changing paragraph indentation	345
	Changing font properties	345
	Setting tabs	346
19	Draw Styles	349
19.1	The Sample Area	350
19.2	Designing Draw Styles	350
	Defining the line type	350
	Defining the fill rule	351
	Defining the line width	351
	Changing line spacing	351
	Changing the dash length	351
20	Table Styles	353
20.1	The Sample Area	354
20.2	Designing Table Styles	354
	Showing the first row's top border	354
	Defining borders	355
	Selecting the header and text style names	355
	Alternating colors for rows or columns	360

Part IV: Reference Appendices

A	Render Objects	363
A.1	Render Object Categories	363
A.2	Subclasses of the Render object	364
B	RTF Limitations and Unsupported Features	367
B.1	Unsupported Features	367

B.2	Limitations	368
C	XML Background Information.	369
C.1	XML Primer	369
C.2	DTD Primer	370
	Glossary	371
	Index	387

Preface

Assumptions ■ *Examples and Demos* ■ *API Documentation (Javadoc)*
Licensing ■ *About Quest Software, Inc.*

Welcome to JClass ServerReport. JClass ServerReport offers Java developers a set of robust methods and procedures for adding paginated, formatted, flowed-text, and image output to Java applications.

Assumptions

This manual assumes that you have some experience with the Java programming language. You should have a basic understanding of object-oriented programming and Java programming concepts such as classes, methods, and packages before proceeding with this manual.

Examples and Demos

JClass ServerReport ships with examples and demos. The examples show you how to create and customize a single JClass ServerReport component. Demos demonstrate how to use one or more JClass ServerReport components within the context of a larger application. Many of the JClass ServerReport examples and demos are referenced in this guide.

The quickest way to run the examples and demos is to use the Jakarta Tomcat application server that is installed with JClass ServerViews.

1. Start the server by selecting *tomcat-startup* from the *JCLASS_SERVER_HOME/bin/* directory.
Note: Microsoft Windows users can launch the Tomcat server from the JClass ServerViews program group off the Start menu.
2. In a web browser, go to: <http://localhost:8686/server-samples/>
3. Select the link for JClass ServerReport.

A list of examples and demos is displayed.

4. Click the name of an example or demo to run it.

Most examples and demos provide a direct link to their source code. The source code is installed with the compiled classes in the *JCLASS_SERVER_HOME/examples/sreport/* and *JCLASS_SERVER_HOME/demos/sreport/* directories.

API Documentation (Javadoc)

The Javadocs for the JClass ServerReport API are part of the *JClass ServerViews API Documentation*. The API documentation is installed automatically when you install JClass ServerViews. It is located in the *JCLASS_SERVER_HOME/docs/api/* directory.

The following packages are particularly relevant for JClass ServerReport:

- `com.klg.jclass.sreport.*`
- `com.klg.jclass.util.server`

On Microsoft Windows installations, you can find a link to the API documentation from the Start menu under the JClass ServerViews program group.

Licensing

You need a valid license to use JClass ServerReport. Information about licensing is outlined in the *JClass ServerViews Installation Guide*, which is automatically installed when you install JClass ServerReport. You can find a PDF version and an HTML version in the *JCLASS_SERVER_HOME/docs/getstarted/* directory.

About Quest Software, Inc.

Quest Software (now a part of Dell) simplifies and reduces the cost of managing IT for more than 100,000 customers worldwide. Our innovative solutions make solving the toughest IT management problems easier, enabling customers to save time and money across physical, virtual and cloud environments. For more information about Quest go to www.quest.com.

Contacting Quest Software

Email	info@quest.com
Mail	Quest Software, Inc. World Headquarters 5 Polaris Way Aliso Viejo, CA 92656 USA
Web site	www.quest.com

See our web site for regional and international office information.

Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a Quest product and have a valid maintenance contract. Quest Support provides unlimited 24x7 access to Support Portal at <http://support.quest.com>.

From our Support Portal, you can do the following:

- Retrieve thousands of solutions from our online Knowledgebase
- Download the latest releases and service packs
- Create, update and review Support cases

View the Global Support Guide for a detailed explanation of support programs, online services, contact information, policies and procedures. The guide is available at:
<http://support.quest.com>.

Quest Communities

Get the latest product information, find helpful resources, and join a discussion with the JClass Quest team and other community members. Join the JClass community at
<http://jclass.inside.quest.com/>.

Part I

Using JClass ServerReport

Learning JClass ServerReport Basics

*Key Concepts ■ Your First ServerReport-based Application
Units of Measurement*

JClass ServerReport uses a flow-markup approach for creating multipage documents to be generated in PDF or RTF format. The major components of JClass ServerReport are:

- The `JCDocument` object, which stores document-level attributes and the list of completed pages (`JCPage` objects), and against which the `JCFlow` object is created.
- The flow mechanism, which is responsible for allocating content to pages and creating new pages as necessary.

1.1 Key Concepts

JClass ServerReport output consists of several different components: the document, the page templates and frames, and the content and flow.

1.1.1 The Document

The document holds the pages and frames, into which content is flowed. A document can contain page templates, text styles, draw styles, and table styles, to enhance the output, which can be either PDF or RTF. In JClass ServerReport, the document is represented by the `JCDocument` object.

1.1.2 The Page Templates and Frames

Before you can flow content into a document, you need a page template that defines how the page is laid out. Page templates specify:

- the physical size of the page
- the location and size of the frames
- the order text is to progress through those frames
- the next page to generate when the existing page and/or section is full

Frames are placed in the page template to hold the text and images that are flowed into the document. For more information on page templates, see Chapter 7, [Defining Page Templates](#).

1.1.3 The Information Content and Flow

Content is added—or flowed—to a JClass ServerReport document using the `JCFLOW` object. For PDF documents, there are two flow processes: the body flow and the front matter flow. The body flow handles the core of your document, such as the chapters in a book. The front matter flow takes care of all the content that appears before page 1 of your document, such as a title page, table of contents, acknowledgements, lists of other content, and the preface. While the front matter flow occurs after the body flow, the front matter content is displayed before the body content in the final document. For more information, see Chapter 8, [Flowing Content into Your Document](#).

Terminology: The word “flow” is used in two different contexts in JClass ServerReport. Flow as a verb, as in “to flow content into a frame,” means to direct content into any frame. Flow used as a noun, as in “body flow,” refers specifically to a frame that has been designated as a flow-frame in order to hold the contents of the `JCFLOW` object.

The objects and templates define where and how the text gets rendered. The methods from the `JCFRAME` and `JCFLOW` classes control the flow of text from one frame to the next and from one page to the next.

When you want to control the flow of text throughout the document, use the `JCFLOW` class. `JCFLOW` manages layout from frame to frame and from page to page. When you want to control a frame layout or render text apart from the main flow of the document, use the `JCFRAME` class.

1.2 Your First ServerReport-based Application

At a minimum, you need to perform the following steps to format and flow a JClass ServerReport document.

Step

1. Select or create a page template.
 2. Instantiate the document.
 3. Instantiate the flow.
 4. Send the document to the output stream.
-

Note: You can also create documents in the JClass ServerReport Designer. For more information, see Chapter 15, [JClass ServerReport Designer](#).

The Application Code

If you compile and run the following program, it generates a one-page PDF document containing the text “Hello, World.”

Note: For information on setting up your environment, please refer to the [JClass ServerViews Installation Guide](#).

```
1 package examples.sreport
2
3 import java.io.StringReader;
4 import java.io.BufferedReader;
5
6 import com.klg.jclass.sreport.JCFlow;
7 import com.klg.jclass.sreport.JCDocument;
8
9 public class DocumentPrinter{
10
11     public static void main(String[] args) {
12
13         // Create a document in PDF format,
14         // setting the page template to be a simple 8.5 x 11 Letter page
15         JCDocument document = new JCDocument(system.out,
16             JCDocument.BLANK_8p5X11, JCDocument.PDF_PRINTER);
17
18         // Instantiate a flow object on the document
19         JCFlow flow = new JCFlow(document);
20
21         // Print some text to the document
22         flow.print("Hello, World.");
23
24         // Print the document to the PDF printer
25         document.print();
26     }
```

Note: This document was created as a PDF; to create an RTF document, line 15 should use `RTF_PRINTER` instead of `PDF_PRINTER`.

On line 15, we construct the document, specifying that the page template to use is the standard 8.5 x 11 that comes with JClass ServerReport. We then create the flow (line 18), render the text “Hello, World” to the flow (line 21), and send the document to `system.out` as a PDF document (line 24).

1.2.1 The JCDocument Object

The `JCDocument` object holds the `JCPage` and `JCFrame` objects into which text and images are flowed.

When you instantiate a new `JCDocument`, you can specify the output stream, along with the name of the standard template it is to use and the printer type, which determines the output that will be generated from the document (PDF or RTF). Alternatively, you can indicate the name of

the standard template to use by calling the `JCPageTemplate.loadTemplates()` method and passing a list of page templates.

JClass ServerReport prints using `JCDocument.print()`. The print command can be as simple as:

```
doc.print();
```

1.2.2 The JCPage and JCFrame Objects

Figure 1 displays a `JCPage` object, which can only exist in a `JCDocument` object. The `JCPage` object contains the `JCFrame` object (body) that holds the text and the `JCFlow` object that renders the text.

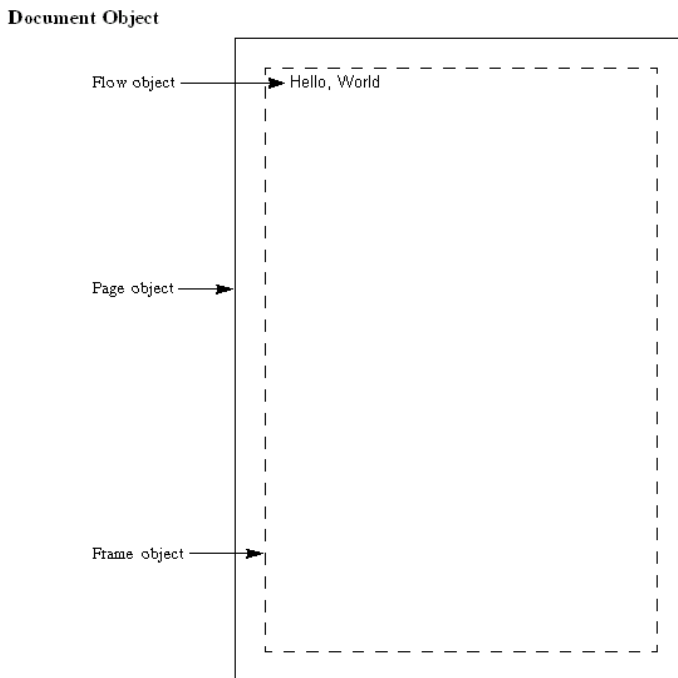


Figure 1 Example of how objects are used to build a page.

1.2.3 The JCFlow Object

When a `JCFlow` object is instantiated for a document, it generates the document's first page. This first page will normally have one or more flow frames, and the first of these will be initialized as the current frame. Once a current frame has been initialized, all flow content is passed to that frame until the frame becomes full.

Note: If the first page does not have a flow frame, successive pages will be generated until a flow frame is discovered.

You create the flow by instantiating a `JCFlow` object, passing in a `JCDocument` as a parameter. The passed-in `JCDocument` is the only one that may be associated with the `JCFlow` object.

The template page's `flow-frame`, `flow-page`, and `flow-section` attributes control the order by which the flow progresses through frames and pages. To control flow beyond the standard sequence specified by the templates, the program needs to call `JCFlow` methods.

Once you have completed the program's code, you must send the document to the output stream. In the example provided in [The Application Code](#), line 24 is responsible for sending the document to the printer: `document.print();`. For more information on sending documents to a printer, see Chapter 9, [Rendering Your Document](#).

Your successfully printed report must then be deployed on your J2EE application server. For information regarding the use of `JClass ServerReport` and different application servers, see the Application Server Guide in the `JCLASS_SERVER_HOME/docs/app_server` directory.

1.3 Units of Measurement

Many `JClass ServerReport` functions require you to measure distances on the output page. For example, to import images or draw shapes, you must pinpoint the location on the page where the image or drawn object is to be placed. To do so, you must possess an understanding of the methods `JClass ServerReport` provides for the precise measurement of linear distances.

The `JCUnit` class lets you define linear distances using three different units of measurement: centimeters, inches, and points. You can set default units and convert distances from one unit type to the next. You can use `JCUnit.Point` to precisely identify a location on a page and `JCUnit.Margins` to create a margin on the inside of a frame.

1.3.1 Setting a Default Unit of Measurement

You can set the default unit type to be centimeters, inches, or points. Once you set a default unit, all methods that use measurement units use the default, unless instructed otherwise. For example, to set centimeters as the default unit type, enter:

```
JCUnit.setDefaultUnit(JCUnit.CM);
```

1.3.2 Converting Units of Measurement

Your application may need to convert a distance from one unit type to another on the fly. `JCUnit` provides methods to convert a distance to each supported unit type. For example, suppose you have defined a distance in centimeters, as follows:

```
JCUnit.Measure measurement = new JCUnit.Measure(JCUnit.CM, 5);
```

To convert distance to a measurement in inches, enter:

```
double distanceInInches = JUnit.getAsInches(
    measurement.units, measurement.distance);
```

or more simply,

```
measurement.getAs(JUnit.INCHES);
```

1.3.3 Defining Points

Some JClass ServerReport functions require you to define a location on a page, for example, in order to draw a line or a polygon. `JUnit.Point` makes it possible for you to precisely define locations, using any of the available units of measurement.

```
JUnit.Point point = new JUnit.Point(JUnit.INCHES, 2.5, 2.5);
```

The preceding example pinpoints a location on the page at the *x*- and *y*-coordinates of 2.5 inches by 2.5 inches. To draw a line or a polygon, you would define other points on the page and use the corresponding `JCFrame` method to connect those points with lines.

Defining Text-Based Content

*The JTextStyle Object ■ Defining Font Attributes ■ Defining Paragraph Attributes
Defining Tabs ■ Defining Lists ■ Using Macros
Customizing Fonts and Font Mappings ■ Internationalization*

JClass ServerReport allows you to customize the appearance of text in your document. This can be done by defining text styles, by applying font attributes, and by setting paragraph attributes and indentation.

To apply styles to your text, use the `JCFlow.print()` or `JCFrame.print()` properties.

Note: When using JClass ServerReport to produce RTF output, text styles should not be changed in the middle of a paragraph if the styles have different paragraph spacing, line spacing, indents or tabs; this is a limitation of the RTF functionality. If JClass ServerReport encounters any such cases, the last text style used in the paragraph is the one that is used for the entire paragraph.

2.1 The JTextStyle Object

The `JTextStyle` class gives you control over the appearance of text in your document output. Many applications require several styles for different types of paragraphs, such as headings, addresses, indented block quotes, and so on. You can use any of the standard styles that come with JClass ServerReport, or you can create and modify your own styles.

Text style can also be associated with a scope object, which can then be used to prevent collisions between identically named styles in a multi-threaded environment (for example, a servlet). The value of the scope is the object that the style will be used within. Possible scope values include an instance of `JCDocument`, an instance of `ServletConfig`, or null (which specifies global scope, indicating that the style is visible to the entire Java VM).

2.1.1 Predefined JTextStyle Objects

Although you can easily create and modify your own styles, you may want to take advantage of the built-in standard styles.

You can apply any standard style using `JCFrame.print()`; for example:

```
frame.print(JCTextStyle.HEADING_BOLD, "North America");
```

The preceding example prints the text “North America” in the standard style `HEADING_BOLD`. Standard styles are constants and, by convention, always appear in uppercase letters. You cannot modify the standard styles themselves, but you can use them to create your own styles. For more information, refer to Section 2.1.2, [Defining your own JCTextStyle Objects](#).

The following table lists and describes the appearance of the standard styles available in JClass ServerReport, all of which have global scope. The standard fonts are Times Roman, Courier, and Helvetica.

Style	Appearance	Name Property
BOLD	Left-aligned, single-spaced, 10 pt. bold Times Roman.	Bold
BOLD_ITALIC	Left-aligned, single-spaced, 10 pt. bold, italic Times Roman.	Bold Italic
CODE	Left-aligned, single-spaced, 10 pt. plain Courier.	Code
CODE_INDENTED	Left-aligned, single-spaced, 10 pt. plain Courier with left, right, and paragraph indents of 0.25".	CodeIndented
DEFAULT_HEADER	Center-aligned, single-spaced, 14 pt. bold Times Roman.	default header
DEFAULT_TEXT	Left-aligned, single-spaced, 12 pt. plain Times Roman.	default text
HEADING	Left-aligned, single-spaced, 10 pt. plain Helvetica.	Heading
HEADING_BOLD	Left-aligned, single-spaced, 10 pt. bold Helvetica.	HeadingBold
HEADING1	Left-aligned, single-spaced, 18 pt. bold Helvetica.	H1
HEADING2	Left-aligned, single-spaced, 18 pt. plain Helvetica.	H2
HEADING3	Left-aligned, single-spaced, 16 pt. bold Helvetica.	H3
HEADING4	Left-aligned, single-spaced, 16 pt. plain Helvetica.	H4
HEADING5	Left-aligned, single-spaced, 14 pt. bold Helvetica.	H5
HEADING6	Left-aligned, single-spaced, 14 pt. plain Helvetica.	H6
HEADING7	Left-aligned, single-spaced, 12 pt. bold Helvetica.	H7
INDENTED	Left-aligned, single-spaced, 10 pt. plain Times Roman, with left, right, and paragraph indents of 0.25".	Indented

Style	Appearance	Name Property
ITALIC	Left-aligned, single-spaced, 10 pt. italic Times Roman.	Italic
NORMAL	Left-aligned, single-spaced, 10 pt. plain Times Roman.	Normal
PLAIN	Left-aligned, single-spaced, 10 pt. plain Times Roman.	Plain

2.1.2 Defining your own JTextStyle Objects

When you instantiate a `JCDocument` object, `JClass ServerReport` generates a default style (plain 12 pt TimesRoman) for any text you print. You can create and modify styles that control the appearance of text in your document, including font selection, indents, and line spacing.

A quick way to create a `JCTextStyle` object is to clone an existing one, saving you the trouble of specifying every attribute of the style.

```
JCTextStyle style = (JCTextStyle) JCTextStyle.NORMAL.clone();
style.setNameandScope("Body", document);
style.setLeftIndent(new JUnit.Measure(JCUnit.CM, 0.5));
style.setRightIndent(new JUnit.Measure(JCUnit.CM, 0.5));
style.setParagraphIndent(new JUnit.Measure(JCUnit.CM, 0.5));
flow.setCurrentTextStyle(style);
```

The preceding example creates a `JCTextStyle` by cloning the standard `NORMAL` style, uses `setNameandBody()` to name it `Body` and set its scope to document (meaning the document on which the style is being used), and gives it left, right, and paragraph (first line) indents of 0.5 centimeters. `JCFlow.setCurrentTextStyle()` is called to apply this style to the text in the current flow. All subsequent text will appear in this style until a new style is applied.

Using JTextStyle Objects within your Application

Once you have all of the text styles you will need for your application defined, you must apply them to the text in your code. For example:

```
flow.setCurrentTextStyle(normal);
normal.setFontStyle(Font.BOLD | Font.ITALIC);
flow.print("Hello, world!");
flow.newParagraph();
normal.setFontStyle(Font.PLAIN);
```

defines a text style named `normal`, and applies it to the text "Hello, world!"

Cleaning Up JTextStyle Objects

After a program is finished with a scope object that has been used to create text styles, and is no longer needed in a document, it should be cleaned up by calling `JTextStyle.cleanup(scope)`, where `scope` is the scope object associated with the style.

For example, calling `JTextStyle.cleanup(document)` cleans up all document-level text styles, and should be called after the document has been printed. Calling `JTextStyle.cleanup(ServletConfig)` cleans up all text styles created with the `ServletConfig` in the servlet's `init()` method.

2.2 Defining Font Attributes

Java maps fonts from their AWT names to their platform-specific equivalents. For example, “TimesRoman” maps to “Times Roman” in Windows. When you program a font change, you use the AWT name. JClass ServerReport then identifies a font that corresponds to the desired Java font. In this way, your code can run across platforms, finding and using the desired fonts.

In JClass ServerReport, you can specify which font a text style is to use, for example:

```
JTextStyle.setFontFamily("TimesRoman");
```

or by passing in an actual Java font object, for example:

```
JTextStyle.setFont(new java.awt.font("TimesRoman", Font.PLAIN, 12));
```

Using a font map, JClass ServerReport then identifies a font that corresponds to the selected Java font.

You can use any of the following fonts in your document: TimesRoman, Helvetica, and Courier. These fonts have slightly different names on different platforms, so Java maps them to their platform-specific equivalents. Similarly, the fonts Arial, Dialog, DialogInput, Serif, and SansSerif may be used, but each is mapped to one of the three basic fonts listed above.

You can specify additional fonts and change how fonts are mapped. For more information, see Section 2.7, [Customizing Fonts and Font Mappings](#).

2.2.1 Italic and Bold

Both italic and bold formatting is applied to text by passing the `ITALIC` or `Bold` constant before the text that will be flowed into the document. The following code fragment prints text normally, then applies italic and bold formats to selected text:

```
flow.print("This is a simple ");
normal.setFontStyle(Font.BOLD);
flow.print("JClass ServerReport ");
normal.setFontStyle(Font.PLAIN);
flow.print("example which does a number of");
normal.setFontStyle(Font.ITALIC);
flow.print("different");
```



```
normal.setFontStyle(Font.PLAIN);
flow.pring("things.");
flow.newParagraph();
```

2.2.2 Underlining

Underline mode is turned on and off by passing constants `JCTextStyle.LINEMODE_UNDERLINE` and `JCTextStyle.LINEMODE_NONE` to the `JCTextStyle` method called `setUnderlining()`. The following code fragment, taken from *examples/sreport/intro/UnderlineServlet.java*, turns underlining on, prints some underlined text, then returns the flow to normal, non-underlined mode:

```
// create a text style
JCTextStyle underlinedText = new JCTextStyle("Normal");
underlinedText.setUnderlining(JCTextStyle.LINEMODE_UNDERLINE);

// apply the style
flow.setCurrentTextStyle(underlinedText);

// print some text to the document
flow.print("This is some underlined text.");

underlinedText.setUnderlining(JCTextStyle.LINEMODE_NONE);

// apply the style
flow.setCurrentTextStyle(underlinedText);
```

2.2.3 Subscripts and Superscripts

The position of text relative to the baseline is controlled by `setBaselineOffset()`, which takes three parameters: `JCTextStyle.OFFSET_NONE`, `JCTextStyle.OFFSET_SUBSCRIPT`, or `JCTextStyle.OFFSET_SUPERSCRIPT`. Thus, to cause text to appear as a subscript, use

```
// Switch text style to subscript mode
style.setBaselineOffset(JCTextStyle.OFFSET_SUBSCRIPT);
```

to cause text to appear as a superscript, use

```
// Switch text style to superscript mode
style.setBaselineOffset(JCTextStyle.OFFSET_SUPERSCRIPT);
```

and to return text to normal size, use

```
// Return text style to normal mode
style.setBaselineOffset(JCTextStyle.OFFSET_NONE);
```

To control the size of the subscripted or superscripted text, use the method called `setSubscriptRatio()`. It takes a double which specifies the size of the subscripted or superscripted text relative to the size of the current normally-sized text. If this method is not called, a default ratio of 0.75 is used.

2.2.4 Background Color

When JClass ServerReport renders text, there is an implicit bounding box that surrounds the text. By default, the bounding box is not drawn because there is no background color for the text. When a background color is specified for a text style, the bounding box for the text is drawn and filled with the background color before the text is rendered.

You can set the background color either in the `JCTextStyle` constructor or by using the `setBackground()` method. The `Background` property takes a `Color` object.

```
// Add a background color to the text style
style.setBackground(Color.orange);
```

2.3 Defining Paragraph Attributes




You can set a style's font properties to modify individual words or characters in a paragraph, but properties such as alignment, indents, line spacing, and paragraph spacing apply to the entire paragraph.

2.3.1 Alignment

JClass ServerReport supports the usual paragraph alignment options; left, center, right, and justified. Left alignment is the default.

To control the alignment of a paragraph style, call `JCTextStyle.setAlignment()`, for example:

```
style.setAlignment(JCTextStyle.ALIGNMENT_CENTER);
```

Parameter	Result	
ALIGNMENT_LEFT	Paragraph text is left aligned.	
ALIGNMENT_RIGHT	Paragraph text is right aligned.	
ALIGNMENT_CENTER	Paragraph text is center aligned.	
ALIGNMENT_JUSTIFY	Paragraph text is left and right aligned.	

2.3.2 Indents

Indent properties control how far from the edge of the frame JClass ServerReport renders the text.

To control indentation, call the appropriate `JTextStyle` indent method. To define the indentation width, use a `JUnit.Measure` object. In turn, `JUnit.Measure` requires `JUnit.UNITS` to specify the appropriate units of measurement. For example:

```
style.setLeftIndent(new JUnit.Measure(JUnit.INCHES, 0.25));
```

The preceding example creates a left indent at 0.25" from the edge of the frame.

JCTextStyle Method	Description
<code>setLeftIndent()</code>	Defines the amount of space between the left side of the frame and the left edge of every line in the paragraph except for the first line.
<code>setParagraphIndent()</code>	Defines the amount of space between the left side of the frame and the left edge of the first line in the paragraph.
<code>setRightIndent()</code>	Defines the amount of space between the right side of the frame and the right edge of the paragraph.

The following diagram demonstrates how left, paragraph, and right indents are applied.

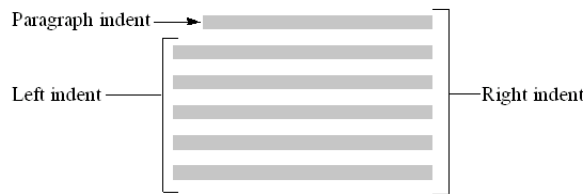


Figure 2 Indentation types.

2.3.3 Paragraph Spacing

Paragraph spacing controls the amount of space between paragraphs in the document. Paragraph spacing is measured between the baseline of the last line of text in the preceding paragraph and the baseline of the first line of text in the following paragraph. Note the difference from line spacing, which controls the amount of space between lines inside the paragraph. Like line spacing, paragraph spacing is defined as a multiple of the height of a line of text in the current style.

```
style.setParagraphSpacing(2);
```

If the text in the current style is 10 points high, JClass ServerReport leaves a gap of 10 points between paragraphs.

Line Spacing

Line spacing controls the amount of space between the baselines of text inside a paragraph. Line spacing is defined as a multiple of the height of a line of text in the current style.

```
style.setLineSpacing(1.2);
```

The preceding example sets the line spacing of the current `JCTextStyle` object to 1.2 times the height of the text. If the text is 10 points high, 12 points of space are left between each line of rendered text.

2.4 Defining Tabs

You create a tab stop by creating a `JCTab` object. When doing so, you can define variables that control alignment, position, and fill.

The following example creates a left-aligned tab stop at the left margin with no fill.

```
tab = new JCTab();
```

You can instantiate a left-aligned `Tab` object at a position you specify:

```
tab = new JCTab(new JUnit.Measure(JUnit.CM, 3));
```

You can instantiate a `Tab` object, aligned and at a position you specify:

```
tab = new JCTab(new JUnit.Measure(new JUnit.CM, 8),  
    JCTab.TAB_ALIGNMENT_CENTER);
```

2.4.1 Adding Tabs to a Style

Typically, you set tab properties as part of a style, meaning that all further occurrences of that style are created with tabs in the same locations. To add tabs to a style, use `JCTextStyle.addTab()` or `setTabs()`.

To align to a defined tab, you must call the `JCFlow.tab()` method after adding your `JCTab` object to a style (that is, you need to call the `JCFlow.tab()` method in order to make use of the tab).

Method	Result
<code>addTab()</code>	Adds a tab to the style in the location specified.
<code>setTabs()</code>	Adds a list of identically aligned tabs to the style.

Using `addTab()`, you can add a single tab to the style, aligned and at a position you specify. The following example creates a left-aligned tab 3 cm from the edge of the frame.

```
JCTextStyle.addTab(JCTab.TAB_ALIGNMENT_LEFT,  
    new JUnit.Measure(JUnit.CM, 3));
```

Using `setTabs()`, you can add a list of regularly spaced tabs with a common alignment. The following example formats the style with eight left aligned tabs, each spaced one centimeter apart.

```
JCTextStyle.setTabs(JCTab.TAB_ALIGNMENT_LEFT,  
    new JUnit.Measure(JUnit.CM, 1), 8);
```

2.4.2 Tab Alignment

Once you have instantiated the `JCTab` object, you can use `setTabAlignment()` to adjust the way text is aligned to it. The following example aligns the right edge of the text to the tab stop location.

```
tab = new JCTab();
tab.setTabAlignment(JCTab.TAB_ALIGNMENT_RIGHT);
```

The following table describes the alignment options:

Field	Result	
TAB_ALIGNMENT_CENTER	Aligns an equal amount of text on either side of the tab stop.	<div>Germany France Switzerland</div>
TAB_ALIGNMENT_LEFT	Aligns the left side (beginning) of the text with the tab stop.	<div>Germany France Switzerland</div>
TAB_ALIGNMENT_RIGHT	Aligns the right side (end) of the text with the tab stop.	<div>Germany France Switzerland</div>
TAB_ALIGNMENT_DECIMAL	Aligns a decimal or period (.) in the text with the tab stop. Primarily used for aligning columns of numbers.	<div>23.65 3.219 587.3</div>

2.4.3 Tab Position

You use `JCTab.setPosition()` in combination with `JCUnit.Measure()` to adjust the horizontal (X-axis) location of the tab stop.

```
tab = new JCTab();
tab.setPosition(new JCUnit.Measure(JCUnit.INCHES, 1.0));
```

The preceding example creates a tab stop at 1". You can use different units of measurement (POINTS or CM), if you prefer. If you do not declare a unit of measurement, `JClass ServerReport` uses the default unit type.

2.4.4 Tab Fill

Often, users want to include a fill or leader between the text before and the text after the tab, as in the example of a Table of Contents:

4.4 Inserting Tab Stops 38

To control fill properties, use `JCTab.setTabFill()`. To create a right-aligned, right-margin tab filled with leader dots (such as the previous example), use the following code:

```
tab = new JCTab();
tab.setPosition(new JUnit.Measure(JUnit.INCHES, 5.5));
tab.setTabAlignment(JCTab.TAB_ALIGNMENT_RIGHT);
tab.setTabFill(JCTab.TAB_FILL_DOTS);
```

Other fill options are `TAB_FILL_NONE` and `TAB_FILL_UNDERLINE`.

2.5 Defining Lists

You can add ordered and unordered lists to your document. Lists can be nested to create sublists within lists.

2.5.1 Understanding Lists and List Items

Both ordered and unordered lists are created using a `JCList` object. You get an ordered or unordered list depending on whether you specify `JCList.ListType.Ordered` or `JCList.ListType.UnOrdered` (default) in the constructor. You can create a list as part of a flow or within a static frame. Each list can contain list items or other lists. The number or bullet that appears before a list item is called the *tag*. You can specify the tag (either a `JCListNumber` for ordered lists or a `JCListBullet` for unordered lists) as well as the tab stops for the tag and text when you instantiate the `JCList` object. Once a `JCList` object is created, it is passed to the `newList()` method of the flow or frame.

Once `newList()` is called, you are ready to create the list's items. This is done by calling the `newListItem()` method. If you attempt to flow anything into a list other than a list item or another list, an error is thrown. You can pass a `JCListItem` object to a `newListItem()` if you want to use a different text style for the item than the current text style defined in the flow. For more information, see Section 2.1.2, [Defining your own JCTextStyle Objects](#).

Once `newListItem()` is called, you can flow items to the list item in the normal way. List items can contain text (with a text style if desired), hyperlinks, and images. They cannot contain tables, frames, contents lists, and horizontal rules. Once the list item is completed, `endListItem()` is called.

For example, the following code creates an unordered list with three list items. The tag tab is at 0.25 inches and the text tab is 0.25 inches further along, placing it at 0.50 inches. Because the tag is not specified in the constructor, the tag will be the default bullet tag, which is a dash.

```
list = new JCList(JCList.ListType.UnOrdered,
                 new JUnit.Measure(0.25)
                 new JUnit.Measure(0.25));
flow.newList(list);
for (int i = 0; i < 3; i++) {
    flow.newListItem();
```

```

        flow.print("This is item " + (i + 1));
        flow.endListItem();
    }
    flow.endList();

```

Once all the list items have been completed, `endList()` is called.

2.5.2 Creating an Ordered List

Ordered lists are numbered sequentially, starting from 1. To create a new ordered list, first define a `JCListNumber` object to set the style of the number tag. Then create a `JCList` object and specify `JCList.ListType.Ordered` in the constructor. Finally, call `newList()` from either `JCFlow` or `JCFrame` and pass in the `JCList` object.

For example, the following code creates an ordered list using Arabic numbers (specified by `JCListNumber(Numbered)`) followed by a bracket in bold-italic style. The tag tab is set at 0.35 points. The numbers are right aligned by default.

```

import static com.klg.jclass.sreport.JCListNumber.NumberingStyle.*;
...

JCListNumber listNumber = new JCListNumber(Numbered);
listNumber.setTagString(")");
listNumber.setTagTextStyle(boldItalicStyle);
olist = new JCList(JCList.ListType.Ordered, listNumber,
    new JUnit.Measure(0.35));
flow.newList(olist);
for (int i = 0; i < 3; i++) {
    flow.newListItem();
    flow.print("This is item " + (i + 1));
    flow.endListItem();
}
flow.endList();

```

Number Style

As shown in the preceding example, you have a fair amount of control over the number tag using `JCListNumber`. The complete `JCListNumber` constructor is:

```

JCListNumber(JCListNumber.NumberingStyle numberingStyle,
    int tabAlignment,
    String tagString,
    JCTextStyle tagTextStyle)

```

For the `NumberingStyle` property, you can choose a `JCListNumber.NumberingStyle` enum from those contained in the following table. You can set the numbering style in the constructor as shown in the example, or you can use the `setNumberingStyle()` method.

Enum	Result	
Numbered	Arabic numbers	1, 2, 3, ...

Enum	Result	
AlphaLower	Lowercase letters	a, b, c, ...
AlphaUpper	Uppercase letters	A, B, C, ...
RomanLower	Lowercase Roman numerals	i, ii, iii, ...
RomanUpper	Uppercase Roman numerals	I, II, III, ...

You can choose the `TabAlignment` for the number tag from the `JCTab` enums shown in the following table.

Field	Result	
TAB_ALIGNMENT_CENTER	Aligns the number tag in the center.	1. 10. 100.
TAB_ALIGNMENT_LEFT	Aligns number tag to the left.	1. 10. 100.
TAB_ALIGNMENT_RIGHT	Aligns the number tag to the right.	1. 10. 100.

The `TagString` property controls what, if anything, is added after the numeral and before the text. By default, each numbering style is followed by a period.

1. list item
2. list item
3. list item

You can change the `TagString` to another character or `String`, such as a bracket, using either the `JCListNumber` constructor or the `setTagString()` method.

- 1) list item
- 2) list item
- 3) list item

You can set the `TagTextStyle` property in either the constructor or the `setTagTextStyle()` method of the `JCListNumber` object. The text style applies to the number and the tag String. For more information, see Section 2.1.2, [Defining your own JTextStyle Objects](#).

2.5.3 Creating an Unordered List

Unordered lists have a bullet rather than a number before each list item. To create an unordered list, create a `JCList` object and specify `JCList.ListType.UnOrdered` in the constructor. Then call `newList()` from either `JCFlow` or `JCFrame` and pass in the `JCList` object.

In the following example, a black dot symbol is used as a bullet and the tab is set at 0.25 inches.

```
JCSymbolStyle symbolStyle =
    new JCSymbolStyle("Dot-black-3", JCSymbolStyle.DOT, Color.black,
        new JUnit.Measure(JUnit.POINTS, 3.0), true);
ulist = new JCList(JCList.ListType.UnOrdered,
    new JCListBullet(symbolStyle),
    new JUnit.Measure(0.25));
flow.newList(ulist);
for (int i = 0; i < 3; i++) {
    flow.newListItem();
    flow.print("This is item " + (i + 1));
    flow.endListItem();
}
flow.endList();
```

As shown in the preceding example, you have a fair amount of control over how the bullet looks using `JCListBullet`. Your primary choice is whether to use a `JCSymbolStyle` object or a String to define the bullet.

Bullet Style Using JCSymbolStyle

The `JCListBullet` constructor for a symbol is:

```
JCListBullet(JCSymbolStyle bulletSymbolStyle)
```

When you define your `JCSymbolStyle` object, you can choose from a variety of shapes and colors and set the size of the bullet in points. The shapes are shown in Figure 3. If you are familiar with `JClass ServerChart`, these are the same symbols you would use in a plot-type chart to represent points in a data series.



Figure 3 Symbols available in *JCSymbolStyle*.

For example, the following code creates a blue triangle that fits within a 4 point area.

```
JCSymbolStyle bulletSymbolStyle =
    new JCSymbolStyle("Blue Triangle", JCSymbolStyle.TRIANGLE,
        Color.blue,
        new JUnit.Measure(JUnit.POINTS,4.0));
```

In addition to setting the display properties, you can promote consistency in your application by choosing to make symbols uneditable. To make a symbol uneditable, set its `Immutable` property to `true` within the constructor or use one of the `makeImmutable()` methods.

You can also define the scope of the symbol's application. By default, symbols are global, that is, all applications within the VM can make use of them. Because symbol style names must be unique within the scope, you may wish to restrict your symbols' scope to a particular `JCDocument` or `ServletConfig`. To set the scope, specify the scope object in the constructor or use one of the `scope()` methods. For more information, look up `com.klg.jclass.sreport.JCSymbolStyle` in the *API Documentation*.

Bullet Style Using a String

The `JCListBullet` constructor for a String is:

```
JCListBullet(String tagString, JCTextStyle tagTextStyle)
```

The `TagString` can be anything you want to use as a bullet, such as a single character (-) or words (Action Item). You can set the `TagTextStyle` property in either the constructor (as shown) or using the `setTagTextStyle()` method. For more information, see Section 2.1.2, [Defining your own JCTextStyle Objects](#).

2.5.4 Nesting Lists

To nest lists, use nested `newList()` and `endList()` calls within a list, between list items. You can mix ordered and unordered lists when nesting lists.

For example:

```
symbolStyle =
    new JCSymbolStyle("Diamond-red-5", JCSymbolStyle.DIAMOND, Color.red,
        new JUnit.Measure(JUnit.POINTS, 5.0), true);
list = new JCList(JCList.ListType.UnOrdered,
    new JCListBullet(symbolStyle),
    new JUnit.Measure(0.25));
flow.newList(list);
for (int i = 0; i < 3; i++) {
    flow.newListItem();
    flow.print("This is an item, numbered " + (i + 1) +
        ", that I hope will take more than one line.");
    flow.endListItem();
    if (i == 1) {
        symbolStyle =
            new JCSymbolStyle("Circle-blue-5", JCSymbolStyle.CIRCLE,
                Color.blue,
                new JUnit.Measure(JUnit.POINTS, 5.0),
                true);
        list = new JCList(JCList.ListType.UnOrdered,
            new JCListBullet(symbolStyle),
            new JUnit.Measure(0.25));

        // Nest another list inside this list
        flow.newList(list);
        for (int j = 0; j < 3; j++) {
            flow.newListItem();
            flow.print("This is a nested item, numbered " + (j + 1) +
                ", that I hope will take more than one line.");
            flow.endListItem();
        }
        flow.endList();
    }
}
flow.endList();
```

2.6 Using Macros

JClass ServerReport ships with some standard macros. You can also create your own macros.

The following table contains the list of standard macros:

Macro	Purpose
PAGE_NUMBER	Inserts the current page number.
ROMAN_NUMBER	Inserts the current page number in Roman numerals.
SECTION_NUMBER	Inserts the current section number.
SECTION_PAGE_NUMBER	Inserts the current page number within the current section.
SECTION_PAGE_TOTAL	Inserts the total number of pages in the current section.
PAGE_TOTAL	Inserts the total number of pages in the document.

As you can see, the standard macros generally fall into two basic types: numbering macros and page total (or counting) macros. The following two sections show you how to use each type of macro. The last section talks about creating a custom macro.

Note: Macros are atomic. Because macros do not wrap, you need to ensure that you have enough space in the frame to hold the macro content or an exception will be thrown. We recommend that macro content be short.

2.6.1 Using Numbering Macros

You can number your pages or sections by embedding one of the numbering macros in the frame that is to contain the page number, usually the header or the footer, but it could be any frame, including the flow frame.

Note: For static frames, you need to add macros to the page template before `JCFLOW` is instantiated. For more information, see [Defining Page Templates](#), in Chapter 7.

For example, in the following code, the `PAGE_NUMBER` macro is added to the footer frame of the specified page template. A text style is applied to the page number. Note that the programmer needs to query the document and the page template for the appropriate Objects.

```
JCPage template_page = doc.stringToTemplate("BookLeft");
JCFrame footer_frame = template_page.stringToFrame("footer");
JCTextStyle style = JCTextStyle.stringToStyle("default text");

// print the page number macro to the frame
try {
    footer_frame.print(style, TextMacro.PAGE_NUMBER);
}
catch (EndOfFrameException e) {}
```

Whenever a new page is generated from the page template, the macro is evaluated.

Repeat this process for every page template used to generate document pages that contain page numbers.

```

template_page = doc.stringToTemplate("BookRight");
footer_frame = template_page.stringToFrame("footer");

// print the page number macro to the frame
try {
    footer_frame.print(style, TextMacro.PAGE_NUMBER);
}
catch (EndOfFrameException e) {}

```

2.6.2 Using Page Total Macros

Depending on how you expect your document to be used, you may find it beneficial to add the total number of pages in a section or in the entire document on each page. For example, if you expect the document to be printed, a document page total helps readers confirm that they have the entire document.

Here are some samples that include the page total macro:

- 1 of 152
- Page 1 of 495
- Section 1, Page 1-1 of 1-17

Page Total Macros and JCDocument's Output Policy

By default, the `JCDocument` `OutputPolicy` property is set to `OUTPUT_POLICY_IMMEDIATE`, which means that when the flow to a page is completed, `JClass ServerReport` outputs the page and releases the memory that was used for that page. However, when you add a page total macro to a page, a page is not deemed to be complete until the page total is inserted. And because the page total is not known until the document completes, the result is that all pages are held in memory until the entire document is flowed. This behavior is not an issue with short documents and may even be desirable in some cases, but generating large documents using this output policy causes `OutOfMemoryErrors`.

Note: If you always want a document to be held in memory until printed, set the `OutputPolicy` property to `OUTPUT_POLICY_ON_REQUEST`.

For PDF documents, you can change how `JClass ServerReport` handles page total macros (or any macro that can only be evaluated at the end of a flow) by setting the `OutputPolicy` property to `OUTPUT_POLICY_ALWAYS`. In this case, a placeholder is used for the page total, which allows the page to be completed. `JClass ServerReport` puts a call to an `XObject` in the PDF file and releases the memory used for the page. When the document is complete and the page total is known, the `XObject` corresponding to the page total macro is created in the PDF file. When the `XObject` is called, it executes the draw commands required to render the page total on each page on which the call was inserted.

Note: `OUTPUT_POLICY_ALWAYS` is not supported for RTF output.

You can affect the size of the placeholder by setting the `MacroSpaceChars` property. `MacroSpaceChars` takes a number between 1 and 10, which represents the number of digits that

you expect in the final page count. The default is 4 digits, which would hold a maximum value of 9999. The font defined for the footer also plays a role in how much space JClass ServerReport allocates for the page total; some fonts and font sizes require more space than others. Finally, you can set the alignment of the page total within the defined macro space using the `MacroSpaceAlignment` property, which takes one of `ALIGNMENT_LEFT` (default), `ALIGNMENT_CENTER`, or `ALIGNMENT_RIGHT`.

Embedding a Page Total Macro

You can embed the page total macro in much the same way as you embed a page number macro. In the following example, the programmer sets the `OutputPolicy` and the `MacroSpace` properties.

```
document.setOutputPolicy(OUTPUT_POLICY_ALWAYS);

JCPage template_page = doc.stringToTemplate("myPage");
JCFrame footer_frame = template_page.stringToFrame("footer");
JCTextStyle style = JCTextStyle.stringToStyle("default text");

DocumentPageCountMacro m = new DocumentPageCountMacro();
m.setMacroSpaceChars(5);
m.setMacroSpaceAlignment(DocumentPageCountMacro.ALIGNMENT_CENTER);

try {
    footer_frame.print("Page ");
    footer_frame.print(style, TextMacro.PAGE_NUMBER);
    footer_frame.print(" of ");
    footer_frame.print(style, m);
}
catch (EndOfFrameException e) {}
```

You can create your own page total or other counting-type macro by implementing the `AbstractPageCountMacro` abstract class. For an example of how to implement this class, see the `DocumentPageCountMacro` class in the *API Documentation*.

2.6.3 Creating Custom Macros

JClass ServerReport allows you to create customized macros that allow the insertion of custom run-time text into a document.

To create your own macro, you must first create a java class that implements the `TextMacro` interface. (`TextMacro` is in the `com.klg.jclass.sreport` package.) `TextMacro` specifies three methods that must be implemented in your macro class: `evaluate()`, `getStatus()`, and `getText()`. Once the macro has been added to a document, each implemented method will be called by JClass ServerReport.

If `getStatus()` and `getText()` are called before `evaluate()`, you must ensure that they return `MACRO_INITIALIZED` and a non-null placeholder `String`, respectively. Once `evaluate()` is called by JClass ServerReport, the method should attempt to construct the text `String` that is represented by the macro.

If the macro can be evaluated given the current flow and page information, `evaluate()` must return `MACRO_EVALUATED`, as must any subsequent calls to `getStatus()`. Subsequent calls to `getText()` must return the evaluated text `String`.

If the macro cannot be evaluated given the current flow and page information, `evaluate()` must return `MACRO_NOT_YET_EVALUATED`, as must any subsequent calls to `getStatus()`. Subsequent calls to `getText()` must return a non-null placeholder `String`.

Note: `evaluate()` may be called by `JClass ServerReport` several times per macro (and may be passed different, potentially null values for the flow and page parameters). Once `evaluate()` returns `MACRO_EVALUATED` for the instance of the macro within the current frame, it is never called again. If the macro is being used in a static frame across multiple pages (for example, in the header of a table or a static page frame), the `evaluate()` method will be called again when any new frame containing the macro instance is created by `JClass ServerReport`.

For example, here is a class that overrides `TextMacro` and prints continued each time it is evaluated, except the first time.

```
import com.klg.jclass.sreport.*;

public class ContinuedMacro implements TextMacro {

    /** The text to which this macro has been evaluated */
    protected String text = "";

    /** The status of the result of the last evaluation */
    protected int status = TextMacro.MACRO_INITIALIZED;

    /** True the first time this macro is evaluated; false otherwise. */
    private boolean firstTime = true;

    public ContinuedMacro() {
    }

    /** Return currently evaluated text. */
    public String getText() {
        // if there is no current value for the macro, return
        // placeholder text
        if (text == null) {
            return ("");
        }
        return (text);
    }

    /** Return current evaluation status. */
    public int getStatus() {
        return (status);
    }

    /** Evaluate macro. Parameters flow and page may be null. */
    public int evaluate(JCFlow flow, JCPage page) {
        // if flow or page is null, don't evaluate
        if (flow == null || page == null) {
            text = null;
        }
    }
}
```



```

        status = TextMacro.MACRO_NOT_YET_EVALUATED;
    } else {
        // first evaluation -- text is blank
        if (firstTime) {
            text = "";
            firstTime = false;
            status = TextMacro.MACRO_EVALUATED;
        } // all other evaluations -- text is "continued"
        } else {
            text = "continued";
            status = TextMacro.MACRO_EVALUATED;
        }
    }
}
// return evaluation status
return status;
}
}

```

Note: This macro can be used in many instances; for example, in table headers.

To add this macro to a frame, use:

```

try {
    frame.print(textStyle, new ContinuedMacro());
} catch (EndOfFrameException eofe) {
}

```

To add this macro to the flow, use:

```

flow.print(new ContinuedMacro());

```

2.7 Customizing Fonts and Font Mappings

You can add fonts, set properties for TrueType fonts, and create your own mappings for fonts.

2.7.1 Adding Your Own Fonts

To use a font other than the standard three fonts supported by PDF and RTF (TimesRoman, Helvetica, and Courier), you must first make the font available to JClass ServerReport. If the font you are adding is a TrueType font, you must tell JClass ServerReport where the corresponding font program file (.TTF) is located. If the font you are adding is a Type 1 font, you must give JClass ServerReport a font metrics file (.AFM) that contains the metric information for the characters contained within the font. Type 1 fonts are only supported for PDF output. TrueType fonts are supported by both RTF and PDF output.

To view most PDF and RTF documents, all fonts used within the document must also be made available to the viewing program (for example, Acrobat Reader, Microsoft Word). However, TrueType fonts may be embedded in PDF output, eliminating the requirement for these fonts to be made available to the viewing program. For more information, see Section 2.7.3, [Setting TrueType Font Properties](#). Type 1 fonts cannot be embedded in PDF output; no fonts can be embedded in RTF output.

To add your own fonts, follow these steps. These steps are based on the font example called *FontExample.java*, which is automatically installed in your *JCLASS_SERVER_HOME/examples/sreport/fonts/* directory when you install JClass ServerReport.

1. Locate the necessary files related to the desired new font.

To use a TrueType font, a legitimate copy of the font must be available on your system. The TTF file must first be found so that its location can be given to JClass ServerReport. On Windows platforms, most font files can be found in *C:/WINNT/FONTS* or *C:/WINDOWS/FONTS* (for Windows XP). On Unix systems, fonts might be found in */usr/lib/X11/fonts* or */usr/share/fonts*.

To use a Type 1 font, an AFM file containing the metrics of the characters within the font is required. The AFM file contains a description of the font, but not the font itself. If your copy of the font did not come with an AFM file, you might find one on the Adobe Web site at <ftp://ftp.adobe.com/pub/adobe/type/>. For example, if you wanted to use the Adobe font Galliard in a JClass ServerReport document, you would acquire the font's AFM files from <ftp://ftp.adobe.com/pub/adobe/type/win/all/afmfiles/001-050/017/>.

2. If JClass ServerReport does not recognize the name of the font you added, you may need to create a *user.properties* font name map file.

You may need to create a file called *user.properties* that will specify a font name map. This file creates a mapping between the AWT font names that you will use in your program (for example, GalliardRoman) and the actual font name, as specified within the font itself (for example, Galliard-Roman). If the name of your font in its plain style does not contain any dashes or spaces, you may not need to create a *user.properties* file; an automatic mapping will be attempted by JClass ServerReport.

Each line of the *user.properties* file consists of two names separated by an equals (“=”) sign. The name on the left must be the AWT name, and the name on the right must be the actual font name found within the font file.

The AWT name (the name on the left) cannot contain spaces. Any spaces must either be escaped with a backslash (“\”), or replaced with underscores. Styled fonts must be listed in the file as *fontName-Style*, where *fontName* is the AWT font name (mentioned above), and *Style* is one of Bold, Italic, or BoldItalic.

If you require a list of available AWT font names, call `java.awt.GraphicsEnvironment.getAllFonts()` to retrieve a list of fonts available on your system, and then call `getName()` on each font in the list.

The actual font name (the name on the right) must appear exactly as it does in the font file. Therefore, spaces must be left as they are. To determine the actual font names for True Type fonts, call `FontLibrary.getTTFFontNames(fontfileName)` on the font in question.

Here is an example of the *user.properties* font name map file for a program that will use the GalliardRoman and CaslonRoman fonts. The name before the equals sign is the AWT font

name and the name after the equals sign is the actual font name. Note how styled fonts have been specified and the way that spaces in the AWT font name have been dealt with.

```
GalliardRoman = Galliard-Roman
GalliardRoman-Bold = Galliard-Bold
GalliardRoman-Italic = Galliard-Italic
GalliardRoman-BoldItalic = Galliard-BoldItalic
Caslon\ Roman = CaslonRoman
```

3. Tell JClass ServerReport where to find these files.

By calling `JCDocument.addFontPackage(String packagePath)`, where `packagePath` is a String representing the **absolute** path of a directory, all fonts represented by the TTF, TTC, or AFM files in the directory (and all subdirectories) will be added for use in JClass ServerReport and all font names from the *user.properties* file found in these same directories will be mapped. By default, JClass ServerReport automatically calls this method on *C:/WINNT/Fonts* in Windows, *C:/WINDOWS/Fonts* in Windows XP, and */usr/share/fonts* and */usr/lib/x11/fonts* in UNIX. If all of your fonts and font name map files reside in these directories, you do not need to call the method yourself. To turn auto loading off, call `FontLibrary.setAutoLoad(false)` before your first use of either `FontLibrary` or `JCDocument`.

Fonts may also be added one at a time.

- The `FontLibrary.addFont(String fileLocation)` method adds a single font when passed an absolute path to a TTF, TTC, or AFM file.
- The method `FontLibrary.addFont(URL fontURL, int fontType)` loads a single font of the type specified in `fontType` from the passed URL object. `fontType` may be equal to either AFM or TTF.
- The method `FontLibrary.addRelativeFont(String fileLocation)` adds a single font when passed the location of a TTF, TTC, or AFM file relative to the CLASSPATH.

Font name map *.properties* files may also be added one at a time.

- The method `FontLibrary.addFontNameMap(String fileLocation, String name, int fileType)` reads the font name map file found in the directory specified by the absolute path in `fileLocation` and whose name begins with `name` and ends with *.properties*.
- The method `FontLibrary.addFontNameMap(URL fontURL, String name, int fileType)` reads the font name map file found in the directory specified by the passed URL object and whose name begins with `name` and ends with *.properties*.
- The method `FontLibrary.addFontNameMap(URL location, int fileType)` reads the font name map found at the location specified by the passed URL object.
- The method `FontLibrary.addRelativeFontNameMap(String location, String name, int fileType)` reads the font name map file found in the specified directory relative to the CLASSPATH and whose name begins with `name` and ends with *.properties*.

The `fileType` parameter specifies the type of font file that is providing the concrete implementation for the font names referenced within the font name map file. Possible values are AFM and TTF.

Note that once fonts and font name maps have been loaded, they are globally available to all programs running within the same class loader.

2.7.2 Creating Your Own Font Mappings

If you create documents using JClass ServerReport on more than one operating system, you may want to map a font name or font alias to a list of fonts. Also, if a font is found on a system but does not contain a glyph from a string you want to render, you may want to specify an alternative list of fonts to be searched. Finally, in certain cases, a missing font can be synthesized from an existing font.

Mapping a Font Alias to Multiple Fonts

You can create custom font mappings in which you map a font alias (either one of the standard Java logical names or a name of your choice) to a list of fonts. When the document is run on an end-user system, JClass ServerReport searches for fonts on the system in the order they are listed.

For example, you could create the following mapping:

```
SansSerif -> {Arial, LucidaSans, Helvetica}
```

If an end user has both LucidaSans and Helvetica on his system but not Arial, LucidaSans is used. When ServerReport matches font aliases, the case of the font alias does not matter. If the end user does not have any of the fonts installed, JClass ServerReport reverts to the default Java mapping behavior described in Section 2.2, [Defining Font Attributes](#).

To create a font mapping, you can use the `addFontMapToList()` method in the `FontLibrary` class. The method takes a `Map<String, List<String>>` object, which specifies a mapping of a font alias `String` to a list of fonts, and an `int` that specifies the `OutputType` (PDF or RTF). It also takes an `AddStyles` boolean that determines whether JClass ServerReport will add mappings for bold, italic, and boldItalic styles for the given font alias.

The following code creates the SansSerif mapping shown in the preceding example. The `OutputType` is PDF and `AddStyles` is turned on.

```
String aliasFont = "SansSerif";
String[] mappedFonts = {"Arial", "LucidaSans", "Helvetica"};
List<String> mappedFontList =
    new ArrayList<String>(Arrays.asList(mappedFonts));

// Define the mapping
Map<String, List<String>> map = new HashMap<String, List<String>>();
map.put(aliasFont, mappedFontList);

// Add mapping to the list
```

```
FontLibrary.addFontMapToList(map, PDF, true);
```

The font mapping is added to the beginning of the library's list of font mappings for the given output type (PDF or RTF). The new mapping overrides any previous mappings for the same font alias. Because the `AddStyle` property was set to `true`, the following mappings were also added:

```
SansSerif-bold -> {Arial-bold, LucidaSans-bold, Helvetica-bold}
SansSerif-italic -> {Arial-italic, LucidaSans-italic, Helvetica-italic}
SansSerif-boldItalic -> {Arial-boldItalic, LucidaSans-boldItalic,
                        Helvetica-boldItalic}
```

To use the mapping, create a font and use it in a `JTextStyle`.

```
// This uses the "SanSerif-bold" mapping added above
Font f = new Font("SanSerif", Font.BOLD, 14);
JTextStyle textStyle = new JTextStyle("bold");
textStyle.setFont(f);
```

Mapping Ranges of Characters

The selected font on your system may not contain all the characters you need. For example, a font may not contain the Euro symbol. Or you may need to insert Chinese characters in an otherwise English document. In this case, you can specify an alternative font list to search for given character ranges. The first font found on a user's system that contains glyphs for the character will be used.

To map character ranges to font lists for a given font alias, use the `addFontListForCharacterRange()` method in the `FontLibrary` class. The method takes a `String` for the font alias name, followed by a `List<CharacterRange>` object for character ranges, and a `List<String>` object for the list of fonts. It also takes the `AddStyles` boolean that determines whether `JClass ServerReport` should create bold, italic, and boldItalic style mappings as well. The `CharacterRange` class has two properties, `StartChar` and `EndChar`. The characters between the two characters (including the end points) define a character range. If `StartChar` and `EndChar` are the same, then range defines exactly one character.

For example, the following code defines a mapping from a list of character ranges to a list of fonts for Chinese characters for the font alias `sansserif`. The characters are defined using their Unicode representation.

```
String[] chFontNames = {"SimSun", "MS Gothic"};
List<String> chFontList =
    new ArrayList<String>(Arrays.asList(chFontNames));
List<CharacterRange> chRanges = new ArrayList<CharacterRange>();
CharacterRange chrangle1 = new CharacterRange('\u3400', '\u4dbf');
CharacterRange chrangle2 = new CharacterRange('\u4e00', '\u9fbf');
CharacterRange chrangle3 = new CharacterRange('\u2f00', '\u2fff');
chRanges.add(chrangle1);
chRanges.add(chrangle2);
chRanges.add(chrangle3);
FontLibrary.addFontListForCharacterRange("sansserif", chRanges,
                                         chFontList, false);
```

Synthesizing Fonts

Some users may have a base font, but not the bold, italic, or boldItalic styles of a font installed on their system. If the base font is a Unicode TrueType font and the output type is PDF, JClass ServerReport attempts to synthesize the missing styles.

For example if the Tahoma font is available on your system but Tahoma-italic is not available, the italic version of the Tahoma font can be synthesized. Font synthesis is done by slanting or widening the font glyphs appropriately. However, the font metrics used for the synthesized fonts are always taken from the base font, even though the bold or italic glyphs are now different from the base font. In some cases, this may cause characters to be crowded, overlapped, clipped, or blurred. You will need to experiment with each font you want to use to see if the results are acceptable to you. The alternative is to specify a font that is on the system.

2.7.3 Setting TrueType Font Properties

There are some options available to users when using TrueType fonts with JClass ServerReport. These options include the ability to embed parts or the whole of a TrueType font within PDF output and the ability to specify character sets that may affect the size of PDF output.

These options are contained within the `TrueTypeFontProperties` class. An instance of the class that contains the values of properties for a particular font may be obtained by calling the method `FontLibrary.getTrueTypeFontProperties(String userFontName)`, where `userFontName` is the same AWT name of the TrueType font that is used to reference the font within the user program. If an instance of the class has not yet been associated with the specified font, one will be created.

The properties in the following table are available in the `TrueTypeFontProperties` class. Values for each property may be set via a `setPropertyName()` method and retrieved via a `getPropertyName()` method.

Property Name	Description
EmbeddingRules	<p>Specifies whether a TrueType font should be embedded within PDF output. A user may wish to embed a TrueType font if the desired font is not expected to be available on all systems on which the created PDF document will be viewed or printed.</p> <ul style="list-style-type: none">■ <code>DO_NOT_EMBED</code> (default) will not embed the font.■ <code>EMBED_NEEDED</code> will embed only the characters from the font that are used within the document.■ <code>EMBED_ENTIRE_FONT</code> will embed the entire font program within the resulting PDF file. <p>Note: This attribute is not supported for RTF output, as RTF does not support embedding.</p>

Property Name	Description
CharacterRange	<p>Specifies the range of font characters that will be used within the user's document. The property is used to determine which method should be used to encode text within the resulting PDF document. Most users will never need to set this property.</p> <ul style="list-style-type: none"> ■ AUTO_DETECT (default) indicates that JClass ServerReport should automatically detect changes between the basic ANSI character set (character codes 0-255) and the larger Unicode character set (character codes 0-65535) and write out text in the format to which the character corresponds. ■ ANSI will write out all characters in the single-byte ANSI format. This value should be used only if the user is certain that the document will never use a character outside the range 0-255. ■ UNICODE will write out all characters as hexadecimal representations of the multi-byte Unicode format. Its use may result in larger PDF files than the other two settings in some cases, although it may result in a smaller PDF file if the majority of characters the user is referencing lie outside of the ANSI character set.
IncludeUnicodeMap	<p>Specifies whether a ToUnicode character map will be included in PDF output. The default value is <code>true</code>. The ToUnicode character map is used by the PDF viewer program to translate between glyph codes and the characters they represent. It is necessary to allow functions such as cut-and-paste to work correctly. If such functionality is not needed by the user, setting this property to <code>false</code> will result in a smaller PDF file.</p> <p>Note: This attribute is not necessary for RTF output, and as such is not supported.</p>

2.8 Internationalization

Internationalization is the process of making software that is ready for adaptation to various languages and regions without engineering changes. JClass ServerViews products have been internationalized.

Localization is the process of making internationalized software run appropriately in a particular environment.

In JClass ServerViews, all Strings that may be seen by a typical user have been internationalized and are ready for localization. These Strings are in resource bundles in every package that requires them. You need to create additional resource bundles for each of the locales that you want to support.

Note: Localizations that are built into the Java platform – such as number and date formatting – are handled by JClass ServerReport, without the need for you to do any extra work.

To localize your JClass ServerReport, you need the JClass ServerReport source code (requires a source code license). The packages that require localization have a *resources* subdirectory that contains the resource bundles, called *LocaleInfo* (or some similar variation, such as *LocaleBeanInfo*). You may want to perform an automated search of the package structure to find all the resource bundles.

To create a new resource bundle, copy the *LocaleInfo.java* file (staying within the same *resources* directory) and change its name to include standard language and country identifiers for the locale that you want to support. For example, if you want to support French as spoken in France, rename the copy of *LocaleInfo.java* to *LocaleInfo_fr_FR.java*. You can then replace the Strings in the copied file with the French translations.

To use a localized resource bundle, you pass the language and country identifiers to the `setLocale()` method. For example, `setLocale(new Locale(fr, FR))` means that the Strings will be read from *LocaleInfo_fr_FR.java*.

For more information, including standard language and country identifiers, see <http://java.sun.com/j2se/1.5.0/docs/guide/intl/index.html>.

2.8.1 Euro Symbol

In order to use the Euro symbol (€), first check whether the font you are using contains the Euro symbol.

If your font includes the Euro symbol

If the font you are using contains the Euro symbol, you can use this character in JClass ServerReport documents by printing the Unicode value of the Euro character (U+20AC) to a JCFLOW or JCFrame object.

For example:

```
flow.print("\u20ac");
```

The default fonts used by the Adobe Acrobat (for instance, Helvetica, TimesRoman, Courier) and most fonts used by Microsoft Word now contain the Euro symbol. If you are using a custom font, ensure that the font contains the Euro symbol and that the symbol's metrics are included in the font's AFM file under the character name "Euro". For more details about working with custom fonts, please see Section 2.7.1, [Adding Your Own Fonts](#).

If your font does not include the Euro symbol

If the font you are using does not contain the Euro symbol, there are two options with JClass ServerReport:

- make use of the Adobe Euro font package
- use a GIF file for the Euro symbol

The Adobe Euro font package contains free, downloadable font families comprising only the Euro symbol. Thus, when you want to use the Euro symbol, simply switch to one of the fonts in this package, print the Unicode value of the Euro character (U+20AC) to the desired `JCFLOW` or `JCFrame` object, and then switch back to your previous font. The Adobe Euro font package is available as a free download from Adobe at <http://www.adobe.com/type/eurofont.html>

Alternatively, if the font you are using does not contain the Euro symbol, `JClass ServerReport` provides the Euro symbol as a GIF file. The *euro.gif* file is found in *sreport.jar* (`/com/klg/jclass/sreport/resources/`).

You can use the code in this sample to create and reference an `Image` object of the symbol and scale it to your desired size (for instance, to reflect the current point size of your `textstyle`).

Embedding the Euro GIF in text

To embed the Euro GIF in a line of text, call:

```
java.net.URL url = document.getClass().getResource("/com/klg/jclass/  
sreport/resources/euro.gif");  
java.awt.Image euro = java.awt.Toolkit.getDefaultToolkit().getImage(url)
```

Then, when you want to add the Euro symbol to a line of text, call:

```
flow.embedImage(euro, JCDrawStyle.POSITION_ON_BASELINE, new JUnit.  
Dimension(.12, .12));
```

where the `JUnit.Dimension` argument represents a suitable size for the `currentTextStyle`.

For more detailed information, please see the `EuroExample Servlet` example, automatically installed in `JCLASS_SERVER_HOME/examples/sreport/fonts/`

Defining Image-Based Content

Adding Raster-Based Image Content ■ *Adding Vector-Based Image Content*
Importing JClass ServerChart and Other Components ■ *Adding Media Clips*
Handling Items that Overflow the Frame ■ *Adding Image-Based Content in XML*

JClass ServerReport allows the addition of raster-based and vector-based images in a document. A raster-based image can be a static file or a component; a vector based image is one that is likely drawn by JClass ServerReport. You can also add components, such as JClass ServerCharts, and media clips.

3.1 Adding Raster-Based Image Content

3.1.1 Image Resolution

In JClass ServerReport, an image's default resolution is 72 dpi. However, if 72 dpi does not appear fine enough in your output, you can improve the perceived resolution by embedding the image at a size smaller than the actual size. For instance, an 300x300 image embedded at size 150x150 will quadruple the resulting resolution.

3.1.2 Importing Static Image Files

To add an Image to a document, the first step is to load the image file. For example:

```
Image image = Toolkit.getDefaultToolkit().getImage("image.jpg");
```

This example instantiates the image as a `java.awt.Image` object and uses `Toolkit.getDefaultToolkit().getImage()` to load the image from its file source.

Next, you use a `JCFrame` or `JCFlow` `embed()`, `float()`, or `paste()` method to render the image into the current frame or flow. Recall that `JCFrame` methods render content apart from the main flow of the document, while their `JCFlow` counterparts render content into the flow.

Embedding places the image on the current line of text, which will wrap if there is not enough space on the current line to hold the image. For this reason, the `embed` method is often used for smaller graphics.

Floating an image inserts it on its own line. If there is not enough space on the page to hold the image, it “floats” to a roomier location, for example, the top of the next page. For this reason, the `float` method is often used for larger graphics.

Pasting an image locks the image at a set of coordinates you define. The `paste()` method is used to import images that must always appear at the same location, such as a logo in company letterhead. Pasting is only available as a `JCFrame` method.

In the following example, the image is resized to 50 by 50 points and embedded on the current line.

```
try {
    flow.embedImage(image, new JUnit.Dimension(JUnit.POINTS,
        50, 50));
}
catch (EndOfFrameException e) {
    System.out.println(e.toString());
}
```

The following table describes the types of methods available for importing images.

JCFlow Method	Result
<code>embedIcon()</code>	Imports the image specified by <code>javax.swing.icon</code> and places it on the current line of text.
<code>embedImage()</code>	Imports the image specified by <code>java.awt.Image</code> and places it on the current line of text.
<code>floatIcon()</code>	Imports the image specified by <code>javax.swing.icon</code> and places it on its own line.
<code>floatImage()</code>	Imports the image specified by <code>java.awt.Image</code> and places it on its own line.
<code>JCFrame.pasteIcon()</code>	Imports the image specified by <code>javax.swing.icon</code> and locks it to a specified location on the page.
<code>JCFrame.pasteImage()</code>	Imports the image specified by <code>java.awt.Image</code> and locks it to a specified location on the page.

3.1.3 Importing Swing Icons

You can import icons from the `javax.swing.icon` class into a `JClass ServerReport` document in much the same manner you import other images.

```
Icon icon = new ImageIcon(image);
flow.embedIcon(icon);
```

The preceding example creates a Swing icon based on the image defined earlier, and places it on the current line in the flow.

3.2 Adding Vector-Based Image Content

Draw styles define the appearance of objects drawn on a page, such as the lines used in tables. `JCDrawStyle` provides methods and procedures for defining draw styles you can use to control the appearance of the lines and fills of the drawn objects in your document.

Draw styles can be associated with a scope to prevent collisions between identically named styles in a multi-threaded environment, such as a servlet. Scope is defined by the object that the styles will be used within; options include an instance of `JCDocument`, an instance of `ServletConfig`, or null (which specifies global scope).

The following example defines a draw style named `ds`.

```
JCDrawStyle ds = JCDrawStyle.LINE;
```

Stock Draw Styles

The stock draw styles in `JCDrawStyle` comprise the following (please review [JCDrawStyle](#) in the [API](#) for full details):

Style	Name property	Description
BLANK	default blank	no line
LINE	default line	thin black lines
LINE_1POINT	default 1pt line	medium (1 point) black lines
LINE_2POINT	default 2pt line	thick (2 point) black lines
LINE_DASHED	dashed line	thin dashed line
LINE_DOUBLE	default double line	thin black double lines

Setting Scope and Cleaning Up JCDrawStyle Objects

To set the scope on a draw style, simply set it at the same time the style is named with the `setNameAndScope(scope)` method.

After a program is finished with a scope object that was used to create a draw style, it should be cleaned up using the `JCDrawStyle.cleanup(scope)` method, where `scope` refers to the scope object. For example, `JCDrawStyle.cleanup(document)` cleans up all draw styles that are set at the document level, and should be called after the document has been generated.

3.2.1 Setting Line Properties

`JCDrawStyle` provides methods you can use to control the appearance of lines drawn in the draw style you've created. For example, to adjust the thickness of the line to 5 pts, enter:

```
ds.setLineWidth(new JUnit.Measure(JUnit.POINTS, 5));
```

To change the solid line to a dashed line, enter:

```
ds.setLineType(JCDrawStyle.LINE_TYPE_BROKEN);
```

The following table describes the `JCDrawStyle` methods you can use to modify line styles.

JCDrawStyle Method	Result
<code>setDashLength</code>	Controls the length of the dashes and the spaces between them when line type is set to <code>LINE_TYPE_BROKEN</code> .
<code>setForegroundColor</code>	Controls the color of the line.
<code>setLineSpacing</code>	In a multi-line style, such as <code>LINE_TYPE_DOUBLE</code> , controls the amount of space left between the lines.
<code>setLineType</code>	Selects the appearance of the line. Options include: <code>LINE_TYPE_BROKEN</code> <code>LINE_TYPE_DOUBLE</code> <code>LINE_TYPE_SINGLE</code>
<code>setLineWidth</code>	Uses <code>JCUnit.Measure</code> to control the width of the line.

3.2.2 Setting Fill Properties

By altering the draw style, you can adjust the fill color of two-dimensional objects, such as circles, rectangles, or polygons. `JCDrawStyle` provides separate methods for the specification of line and fill colors.

The following example modifies the draw style created in the previous section (`ds`), by setting its fill foreground color to yellow. You can specify any fill color you have defined using `java.awt.Color`.

```
ds.setFillForegroundColor(yellow);
```

3.2.3 Drawing Shapes

Using `JClass ServerReport`, your Java application can draw and print a variety of geometric shapes, including lines, circles, rectangles, rounded rectangles, and polygons.

Note: This feature is not supported in RTF output.

To draw a shape, you must provide a `JCDrawStyle` that describes its appearance. (For more information, refer to Section 3.2, [Adding Vector-Based Image Content](#).) In simple cases such as the following examples, you can use a default style, for example:

```
JCDrawStyle ds = JCDrawStyle.LINE;
```

Drawing Lines

To draw a line, you create an array of points, then call `JCFrame.drawLine()` to connect those points with a line drawn in the current `JCDrawStyle`.

```
ArrayList list = new ArrayList();
```

```
list.add(new JUnit.Point(JUnit.CM, 2, 3.5));
list.add(new JUnit.Point(JUnit.CM, 2, 5.5));
frame.drawLine(ds, list);
```

The preceding example draws a line between the two points defined in the array, as illustrated in the following diagram.

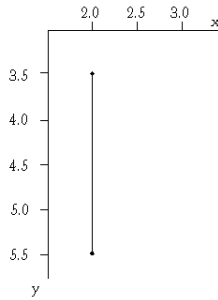


Figure 4 A drawn line.

Drawing Rectangles

To draw a rectangle, use `JCFrame.drawRectangle()`.

```
frame.drawRectangle(ds, new JUnit.Point(JUnit.POINTS, 25, 100),
    new JUnit.Dimension(JUnit.POINTS, 50, 50));
```

The `drawRectangle()` method creates the outline of a rectangle. `JUnit.Point` places the upper-left corner of the rectangle on the *x*- and *y*-coordinates of 25 by 100 points.

`JUnit.Dimension` makes the rectangle a square by setting its size to 50 by 50 points.

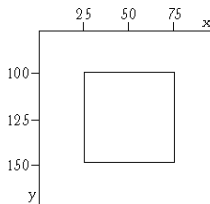


Figure 5 An outlined square (50 by 50).

To draw a filled rectangle, use `fillRectangle()`. The color used to fill the rectangle is the color defined by `JCDrawStyle.setFillForegroundColor()`.

```
frame.fillRectangle(ds, new JUnit.Point(JUnit.POINTS, 100, 100),
    new JUnit.Dimension(JUnit.POINTS, 50, 50));
```

The sample code draws a square with a 100% black fill at the *x*- and *y*-coordinates of 100 by 100 points, with dimensions of 50 by 50 points.

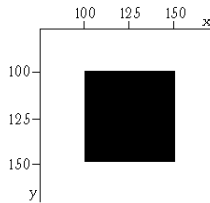


Figure 6 A filled square (50 by 50).

Drawing Rounded Rectangles

You can also draw outlined and filled rectangles with rounded corners. For rounded rectangles, you need to specify the radius of the rounded corners using `JCUnit.Measure`.

To draw the outline of a rounded rectangle:

```
frame.drawRoundedRectangle(ds, new JCUnit.Point(JCUnit.POINTS,  
350, 350), new JCUnit.Dimension(JCUnit.POINTS, 50, 50),  
new JCUnit.Measure (JCUnit.POINTS, 5));
```

The preceding example produces a rounded rectangle with dimensions of 50 by 50 points and a corner radius of 5 points:

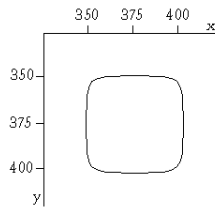


Figure 7 An outlined square with rounded corners.

To draw a filled, rounded rectangle:

```
frame.fillRoundedRectangle(ds, new JCUnit.Point(JCUnit.POINTS,  
400, 400), new JCUnit.Dimension(JCUnit.POINTS, 50, 50),  
new JCUnit.Measure (JCUnit.POINTS, 5));
```


The preceding example produces the following result:

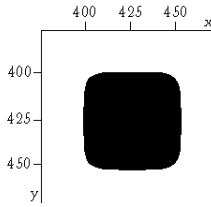


Figure 8 A filled square with rounded corners.

Drawing Circles

To draw a circle, use the `JCFrame.drawCircle()` method. For instance, here is the code to place the center of the circle at the *x*- and *y*-coordinates of 175 by 175 points, and to set the radius of the circle to 25 points:

```
frame.drawCircle(ds, new JUnit.Point(JUnit.POINTS, 175, 175),  
                new JUnit.Measure(JUnit.POINTS, 25));
```

`JUnit.Point` places the center of the circle at the *x*- and *y*-coordinates of 175 by 175 points. `JUnit.Measure` sets the radius of the circle to 25 points.

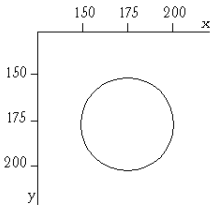


Figure 9 A circle with a radius of 25 points.

As with rectangles, you can draw circles filled with the color defined by the `JCDrawStyle`.

```
frame.fillCircle(ds, new JUnit.Point(JUnit.POINTS, 225, 225),  
                new JUnit.Measure(JUnit.POINTS, 25));
```

The preceding example produces the following result:

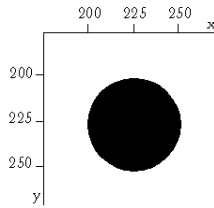


Figure 10 A filled circle with a radius of 25 points.

Drawing Polygons

JClass ServerReport also allows you to draw angular shapes with more than two sides — polygons. To draw a polygon, you create an `ArrayList` of points that define the *x*- and *y*-coordinates of each of the polygon's corners. You then instruct JClass ServerReport to draw the polygon by connecting those points with lines.

For example, you could use the following code to draw a triangle:

```
ArrayList list = new ArrayList();  
list.add(new JUnit.Point(JUnit.POINTS, 275, 250));  
list.add(new JUnit.Point(JUnit.POINTS, 300, 300));  
list.add(new JUnit.Point(JUnit.POINTS, 250, 300));  
frame.drawPolygon(ds, list);
```

Using the *x*- and *y*-coordinates as a guide, JClass ServerReport draws the following object:

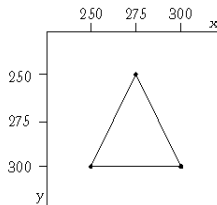


Figure 11 Constructing a polygon from a list of *x*- and *y*-coordinates.

To draw more complex polygons, extend the list. For example, to draw a hexagon, define a total of six points in the list.

3.3 Importing JClass ServerChart and Other Components

In addition to displaying images defined by image files, JClass ServerReport is capable of importing and displaying a visual object, such as a JClass ServerChart, as long as the object is of type `java.awt.Component`. The process is simple: instantiate the component and pass it to the current flow as a parameter in the flow's `embedComponent()` method.

Note: If you choose to embed a component using `embedComponent()`, the component cannot be changed until after the `JCDocument.print()` method has been executed. Because a document is only a series of information and references until it is printed, changing a component before it has been printed may change the output of the component, resulting in it being drawn differently than what was originally intended.

An example of a page containing a JClass ServerChart can be found in *Chart.java* (*JCLASS_SERVER_HOME/examples/sreport/main/*). Once a component that knows how to draw itself is instantiated, one line is all that is needed to embed it in the flow:

```
// Create new chart instance.
chart = new JCServerChart();
chart.setSize(200,200)
// Load the chart's data from a data source
// so there is something to display, then embed it:
// ...
flow.embedComponent(chart); // Places the chart in the flow

// The following adds a caption to the embedded image:
flow.newLine();
flow.setCurrentTextStyle(JCTextStyle.ITALIC);
flow.print("Figure 1.1 A Simple Chart");
```

The image of the component may also be drawn using:

```
flow.floatComponent(chart);
```

In this case, the image is drawn after the current line. If the command is encountered while there is a partial line being output, that line will be completed before the image is positioned on the page. The method takes alignment parameters to further control its position.

If you need to separate the image from the current line, bracket the call to `embedComponent()` with new lines, as per the code below. This will cause the image to begin on a new line, with subsequent text also beginning on a new line.

```
flow.newLine();
flow.embedComponent(chart);
flow.newLine();
```

Method `embedComponent()` takes a `java.awt.Component` as a required parameter and two optional parameters: alignment and size. The alignment parameter takes one of the alignment constants in `JCDrawStyle` for positioning an object vertically relative to the line it is on. This parameter is useful for positioning a small component's image. The size parameter is specified with a `JCUnit.Dimension` object and permits scaling the image both horizontally and vertically.

Besides setting the size by passing a size parameter to `embedComponent()`, a component's printable size is determined by the size that is set using the component's `setSize()` method.

Note: A component must have a size, either set using the `setSize()` method or passed in the `embed` or `float` or `paste` commands.

JClass ServerReport can embed other components only if they are simple (that is, without children) and can be drawn without being seen on the screen. The latter restriction is due to Swing/AWT limitations.

3.4 Adding Media Clips

As with images and components, you can choose to embed or float media clips in a frame or the flow. Media clips can be in video format or audio format, and can be played using any media player that supports the MIME type of the clip.

The following table contains a list of the tested types of media clips. Other types may work as well.

Type of Media Clip	File Extension	MIME type
MP3 audio	.mp3	audio/mpeg
ASF video	.asf	video/x-ms-asf
MPEG-4 video	.mp4	video/mp4
Quicktime video	.mov	vide/quicktime

3.4.1 Methods and Parameters

To add a media clip, you can use the `embedMediaClip()` or `floatMediaClip()` methods from JCFflow or JCFrame.

To add a media clip on the current line, use:

```
embedMediaClip(Object mediaClipSource, String filespec, Image poster,
               String clipTitle, String mimeType, String description,
               int alignment, String clipName, JUnit.Dimension size)
```

To add a media clip in the next place it fits (subject to the existence of preceding floating objects), use:

```
floatMediaClip(Object mediaClipSource, String filespec, Image poster,
               String clipTitle, String mimeType, String description,
               int alignment, String clipName, JUnit.Dimension size)
```

where

- `mediaClipSource` is the source of the media clip, either a `File` or an `URL`.
- `filespec` is the name of the embedded media file, including the extension.
- `poster` is the image you want to see when the clip is not running. You need to supply this image. Note: `JClass ServerReport` does not currently offer a means to extract a frame from a video-based media clip.
- `clipTitle` is a title for the clip. The title is not displayed, but might be used by assistive technologies.
- `mimeType` is the MIME type of the clip. See the table in the preceding section for examples.
- `description` is a description of the clip. Like the title, the description is not displayed, but might be used by assistive technologies.
- `alignment` is the horizontal alignment of the clip. Alignment values can be any of the alignment enums located in `JTextStyle`, such as `ALIGNMENT_CENTER` or `ALIGNMENT_LEFT`.
- `clipName` is a unique name used to keep track of media clips within a document. If null, a name is automatically generated. The `clipName` is internal only and is not displayed.
- `size` is the width and height of the media clip within the document.

3.4.2 Example

The following example (taken from *JCLASS_SERVER_HOME/examples/sreport/EmbedClip.java*) loads an image to use for the `poster` parameter, defines parameters, and then calls `embedMediaClip()` with the parameters.

```
// Load the poster frame for the video clip.JClass LiveTable
Image poster = null;
URL url = null;
Class cl = getClass();
url = cl.getResource("poster.png");
if (url != null) {
    ImageIcon icon = new ImageIcon(url);
    poster = icon.getImage();
}

// Define the method parameters
String source = "Hilton_Falls.mp4";
File mediaClipFile = new File(source);
String clipTitle = "Video of Hilton Falls";
String mimeType = "video/mp4";
String description = "A test video";
int alignment = JTextStyle.ALIGNMENT_CENTER;
String clipName = "clip1";
JUnit.Dimension size = new JUnit.Dimension(2.667, 2);

// Embed the media clip.
flow.embedMediaClip(mediaClipFile, source, poster, clipTitle, mimeType,
    description, alignment, clipName, size);
```

3.5 Handling Items that Overflow the Frame

JClass ServerReport provides you with different methods to handle items (images and components) that are too large for the frame. An item that is too large for the frame is said to overflow its size constraints.

There are different rules that govern how JClass ServerReport will deal with overflowed items, depending on whether the item has been added to the frame or the flow, or if the image is embedded or floating.

3.5.1 Items Embedded in the Flow

For items embedded in the flow, JClass ServerReport attempts to find a frame in the current flow section that will hold the item. If a large enough frame does not exist, the `OverflowRules` property (which contains an instance of the `OverflowRules` class) of the `JCDocument` class is consulted.

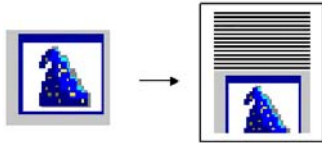
The `OverflowRules` property defines how an embedded item will be handled if it overflows its size constraints. To set this property, call `JCDocument.setOverflowRules()`; to retrieve the property, call `JCDocument.getOverflowRules()`.

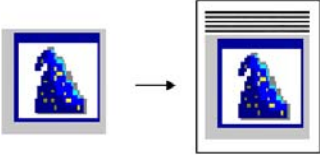
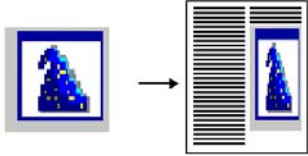
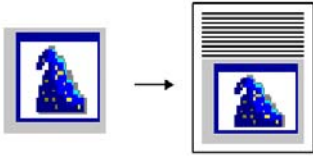
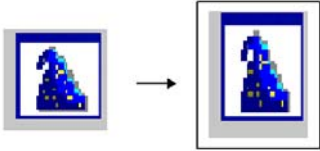
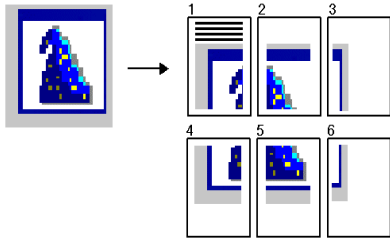
The default behavior for overflowing items embedded in the flow is to place the item in the next column of the current flow frame, and to fit it to the column without altering the item's original aspect ratio. In a case where there are no more columns in the current flow frame, the item is placed in the first column of the next flow frame. The flow is continued in the column following the one containing the overflowed item.

The following properties exist to override the default behaviors: `Method`, `Placement`, and `NextItemPlacement`.

Method

The `Method` property will define how the overflowed item will be displayed.

Possible Values	Example
<p>TRUNCATE</p> <ul style="list-style-type: none">■ Draws the item at the current position, truncating it when the end of the current frame is reached.	

Possible Values	Example
<p><code>FIT_MAINTAIN_ASPECT</code></p> <ul style="list-style-type: none"> ■ The default; fits the item to the current frame while maintaining the item's aspect ratio so as not to stretch or squeeze the item. 	
<p><code>FIT_MAINTAIN_VERTICAL</code></p> <ul style="list-style-type: none"> ■ Fits the item to the current frame, shrinking or stretching the horizontal dimension as needed, but attempting to leave the vertical dimension untouched. If the vertical dimension is larger than the remaining size of the frame, this method behaves in the same way as <code>FIT</code>. 	
<p><code>FIT_MAINTAIN_HORIZONTAL</code></p> <ul style="list-style-type: none"> ■ Fits the item to the current frame, shrinking or stretching the vertical dimension as needed, but attempting to leave the horizontal dimension untouched. If the horizontal dimension is larger than the remaining size of the frame, this method behaves in the same way as <code>FIT</code>. 	
<p><code>FIT</code></p> <ul style="list-style-type: none"> ■ Fits the item to the current frame, shrinking or stretching both the vertical and horizontal dimensions as required. 	
<p><code>DIVIDE_HORIZONTAL</code></p> <ul style="list-style-type: none"> ■ Divides the item across multiple pages in rows. <p>Note: When using this option, ideal results are obtained when all the flow frames are identical in size.</p>	

Possible Values	Example
<p>DIVIDE_VERTICAL</p> <p>■ Divides the item across multiple pages in columns.</p> <p>Note: When using this option, ideal results are obtained when all the flow frames are identical in size.</p>	

Placement and NextItemPlacement

The `Placement` property defines where the overflowing item will be placed; and the `NextItemPlacement` property defines the point after the overflowing item where the flow will continue.

These properties have the same possible values.

Possible Values	
AUTOMATIC	Default. When the <code>Method</code> property is set to either <code>DIVIDE_HORIZONTAL</code> or <code>DIVIDE_VERTICAL</code> , the effective value is <code>NEW_PAGE</code> . Otherwise, it is <code>NEW_COLUMN</code> .
NEW_LINE	The overflowed item is placed at the beginning of a new line. A new line is started if the insertion point of the current frame is not already at the beginning of a new line.
NEW_COLUMN	The overflowed item is placed at the beginning of a new column. A new column is started if the insertion point of the current frame is not already at the beginning of a new column.
NEW_FRAME	The overflowed item is placed at the beginning of a new flow frame. A new flow frame is started if the insertion point of the current frame is not already at the beginning of a new flow frame.
NEW_PAGE	The overflowed item is placed at the beginning of a new page. A new page is started if the insertion point of the current frame is not already at the beginning of a new page.
NEXT_SECTION	A new section is started, and the overflowed item will be placed as the first item in this new section
THRESHOLD	The overflowed item is placed according to the value of the <code>ThresholdPlacement</code> property. See Threshold and Conditional Settings .

Threshold and Conditional Settings

You can use the `THRESHOLD` value to create a conditional setting to define the placement of the overflowed items or of the next object in the flow. When using a threshold, these types of items will only be placed on a new line in the current frame if the defined threshold position in the frame has not been passed. If the threshold has been passed, the overflowed object or the next object will be placed according to the value of either the `ThresholdPlacement` or `NextItemThresholdPlacement` property, respectively.

The `ThresholdPlacement` and `NextItemThresholdPlacement` properties function in the same fashion; either can have the same value as the `Placement` property, with the exception that neither can be set to `THRESHOLD`.

To define the threshold position for placement of overflowed items, use either the `ThresholdPercentage` or `ThresholdMeasure` properties of the `OverflowRules` class. `ThresholdMeasure` uses a `JCUnit.Measure` object to set the threshold as an absolute position on the frame, for example:

```
ThresholdMeasure = new JCUnit.Measure(JCUnit.INCHES, 9.0);
```

`ThresholdPercentage` sets the threshold as a percentage of the total frame size, for example:

```
ThresholdPercentage = 75.0;
```

Note: Only one of these properties should be set. If both are set, `ThresholdMeasure` will override the `ThresholdPercentage` property.

To determine if the placement of the overflowed item has caused the current frame to pass a threshold, and what action should take place before the flow continues, use either the `NextItemThresholdMeasure` or `NextItemThresholdPercentage` properties of the `OverflowRules` class. These function in the same fashion as `ThresholdMeasure` and `ThresholdPercentage`, respectively.

3.5.2 Floating Items

Like items embedded in the flow, `JClass ServerReport` attempts to find a frame in the current document that will hold a floating image. If a large enough frame does not exist, or for some reason does not become current, the `OverflowRules` property (which contains an instance of the `OverflowRules` class) of the `JCDocument` class is consulted.

The `OverflowRules` class defines how a floating item will be handled if it overflows its size constraints. To set this property, call `JCDocument.setOverflowRules()`; to retrieve the property, call `JCDocument.getOverflowRules()`.

In the case of floating items, the default behavior is to shrink the item to fit it in the current frame, if the frame's contents have not passed the 75% mark. If the 75% mark has been passed, the item will be shrunk to fit on the next flow frame. The item's original aspect ratio is preserved in both cases.

The following properties exist to override the default behaviors: `FloatMethod`, `FloatThresholdPercentage` and `FloatThresholdMeasure`.

Note: Floating items will always behave as if there is a threshold dictating whether they can be placed in the current frame, or need to be held for the next one. The threshold should be specified for all floating items. For further information about thresholds, please see [Threshold and Conditional Settings](#).

FloatMethod

The `FloatMethod` property will define how the overflowed item will be displayed. It can have the same values as the `Method` property.

For more information on the possible values, see [Method](#).

FloatThresholdPercentage and FloatThresholdMeasure

`FloatThresholdMeasure` uses a `JCUnit.Measure` object to set the threshold as an absolute position on the frame, for example:

```
FloatThresholdMeasure = new JCUnit.Measure(JCUnit.INCHES, 8.5);
```

`FloatThresholdPercentage` sets the threshold as a percentage of the total frame size, for example:

```
FloatThresholdPercentage = 70.0;
```

Note: Only one of these properties should be set. If both are set, `FloatThresholdMeasure` will override the `FloatThresholdPercentage` property.

3.5.3 Items Embedded in a Frame

When an item that is larger than its size constraints is embedded in a frame, an `ObjectWillNotFitException` is thrown. When this exception is caught, you have the option of using one of the following methods to either resize the item, truncate the item, or paste a portion of the item in the current frame.

The `Alignment` property is used to specify the vertical alignment of the item on the current line. Possible values may be found in the `JCDrawStyle` API.

The `Size` property represents the desired size of the entire item. If `size` is null, then the original size of the item is used.

Fitting the Item

To embed and fit an item to the size of the current frame, use one of the following methods:

- `embedFittedImage(JCTextStyle style, Image image, int alignment, JCUnit.Dimension size, int fitMethod)`
- `embedFittedIcon(JCTextStyle style, Icon icon, int alignment, JCUnit.Dimension size, int fitMethod)`
- `embedFittedComponent(JCTextStyle style, Component component, int alignment, JCUnit.Dimension size, int fitMethod)`

`fitMethod` defines how the image will be altered to fit in the desired space. It must be one of the following:

- `OverflowRules.FIT_MAINTAIN_ASPECT_RATIO` – This will allow the image to retain its aspect ratio.
- `OverflowRules.FIT_MAINTAIN_VERTICAL` – Fits the item to the current frame, shrinking or stretching the horizontal dimension as needed, but attempting to leave the vertical dimension untouched. If the vertical dimension is larger than the remaining size of the frame, this method behaves in the same way as `FIT`.
- `OverflowRules.FIT_MAINTAIN_HORIZONTAL` – Fits the item to the current frame, shrinking or stretching the vertical dimension as needed, but attempting to leave the horizontal dimension untouched. If the horizontal dimension is larger than the remaining size of the frame, this method behaves in the same way as `FIT`.

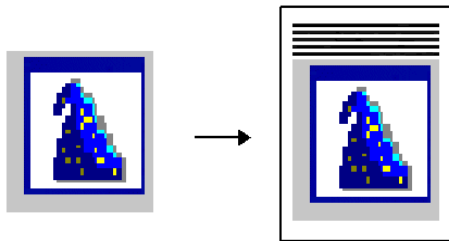


Figure 12 Example of a fitted image that retains its aspect ratio.

Truncating the Item

To embed and truncate an item to fit it in the current frame, use one of the following methods:

- `embedTruncatedImage(JCTextStyle style, Image image, int alignment, JUnit.Dimension size)`
- `embedTruncatedIcon(JCTextStyle style, Icon icon, int alignment, JUnit.Dimension size)`
- `embedTruncatedComponent(JCTextStyle style, Component component, int alignment, JUnit.Dimension size)`

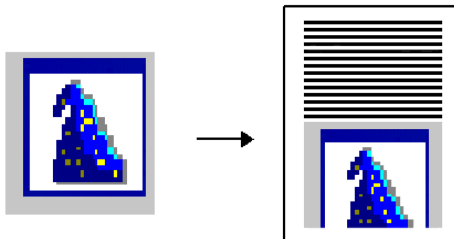


Figure 13 Example of a truncated image in JClass ServerReport.

Embedding or Pasting a Portion of the Item

A portion of an overflowing item can be embedded or pasted. When embedding a portion of an item, the portion of the item will be embedded at the current position; when pasting a portion of an item, the position where the portion of the item will be pasted must be specified.

To embed the portion of an item, use one of the following methods:

- `embedPartialImage(JCTextStyle style, Image image, int alignment, JUnit.Dimension size, JUnit.Point partialOrigin, JUnit.Dimension partialSize)`
- `embedPartialIcon(JCTextStyle style, Icon icon, int alignment, JUnit.Dimension size, JUnit.Point partialOrigin, JUnit.Dimension partialSize)`
- `embedPartialComponent(JCTextStyle style, Component component, int alignment, JUnit.Dimension size, JUnit.Point partialOrigin, JUnit.Dimension partialSize)`

To paste the portion of an item at a specific position, use one of the following methods:

- `pastePartialImage(Image image, JUnit.Dimension size, JUnit.Point position, JUnit.Point partialOrigin, JUnit.Dimension partialSize)`
- `pastePartialIcon(Icon icon, JUnit.Dimension size, JUnit.Point position, JUnit.Point partialOrigin, JUnit.Dimension partialSize)`
- `pastePartialComponent(Component component, JUnit.Dimension size, JUnit.Point position, JUnit.Point partialOrigin, JUnit.Dimension partialSize)`

Note: `partialOrigin` and `partialSize` define the part of the item that will be pasted at the defined origin.

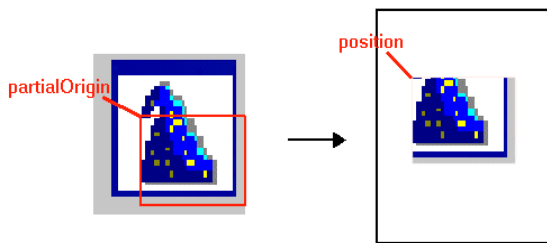


Figure 14 Example of a pasted image.

3.6 Adding Image-Based Content in XML

You can embed or float images, components, and media clips in XML and you can also specify rules for handling overflows. For XML tags and attributes, see the following XML tags in Chapter 14, “JClass ServerReport DTD Tags”:

- [embed-image](#)
- [embed-chart](#)
- [embed-media-clip](#)
- [float-image](#)
- [float-chart](#)
- [float-media-clip](#)
- [overflow-rules](#)

3.6.1 Adding Base64-Encoded Images in XML

Base64 is an encoding method used to transmit or store non-text data in a text-based medium, such as XML. In JClass ServerReport, you can embed or float a Base64-encoded image in an XML file. One benefit of using an encoded image is that you do not need to link to an external image. You can specify which format you want the image to take, including PNG, JPG, and GIF.

To add an encoded image, set the `source-url` attribute of either the `<embed-image>` or `<float-image>` tag. You will need to add the following text to the beginning of your image data (where xxx is the format):

```
<embed-image source-url="data:image/xxx;base64,..." ...></embed-image>
```

The following code shows how to create a document with a Base64-encoded image as a PNG. Because the encoded image data is very lengthy, most of it has been removed for the purposes of this example.

```
<?xml version="1.0"?>
<!DOCTYPE document SYSTEM "JCServerReport.dtd">
<document>
  <!-- this is the root of the document -->
  <page-template title="simple">
    <!-- define the page templates -->
    <simple-page name="simple" unit="inches">
      <size width="8.5" height="11"/>
      <body top-offset="1.0" bottom-offset="1.0">
        </body>
        <flow-section name="simple"/>
      </simple-page>
    </page-template>

    <flow>
      Embedded Image Example<new-line/>
      <!-- Embed the image data -->
      <embed-image
        source-url="data:image/png;base64,iVBORwOKGGoAAAANSUHEUGAAAYEA
```

```
AAE4CAYAAABMqPyJAAAjsklEQVR42u3dzess2V3H8fpLZuPf4EJw4cKNCxdZuJ
iFCxeCEBduXAYCXBEEEQZEXFjxjxNz0TJzMT0+BRjHnQ0NzAgwoAMDEggEISA
CAEXbd/pdNLpdFWd73mo01X1+sCburd/3fXc3efd330qhp01i1i1iMiCGewCER
<!--Snip...removed the rest of the data for space reasons.-->
unit="inches" align="center-in-line" width="5.34"height="2.087">
</embed-image>
</flow>
</document>
```

Defining Linked Content

[Named Locations for Internal Links](#) ■ [Text and Images Links](#)
[Bookmarks](#) ■ [Table of Contents and Other Reference Lists](#)

Online documents such as PDFs typically include links (also called *hyperlinks*) to help readers easily navigate through a document. Links can be internal, such as a cross-reference to related content within the document, or external, such as a link to content on a Web page.

JClass ServerReport supports internal links originating from text, images, the table of contents, other contents lists, and PDF bookmarks. It also supports external links from text and images. Both internal and external links require a destination. In JClass ServerReport, you specify the destination for internal links using locations. Locations mark a position in the document and have a unique name. External links are created using URLs. This chapter describes how to create the various kinds of links.

4.1 Named Locations for Internal Links

In order for an internal link to function, the link must know where it is pointing in the document. It does so through the use of a unique name associated with the position in the document where the link should point. Locations are used to mark those positions and associate the unique names with them.

Note: Locations should not be marked in static frames, such as headers and footers, since it is not possible to associate multiple locations with a single name.

There are several methods that can be used to mark a location, all of which are on either the JCFLOW, JCFrame, or JCPage objects:

Method	Use
■ JCFLOW.markCurrentLocation(String name)	Associates the current insertion point (that is, the location after the previous item added to the flow or frame) with the given name.
■ JCFrame.markCurrentLocation(String name)	

Method	Use
<ul style="list-style-type: none"> ■ <code>JCFlow.markNextLocation(string name)</code> ■ <code>JCFrame.markNextLocation(String name)</code> 	Associates the location of the next item that will be added to the flow or frame with the given name.
<ul style="list-style-type: none"> ■ <code>JCFlow.markNextFloatLocation(String name)</code> ■ <code>JCFrame.markNextFloatLocation(String name)</code> 	Associates the location of the next item added to the list of floating objects (this is accomplished with methods like <code>floatImage()</code>) with the given name.
<ul style="list-style-type: none"> ■ <code>JCFlow.markExactFrameLocation(String name, JUnit.Point location)</code> ■ <code>JCFrame.markExactLocation(String name, JUnit.Point location)</code> 	Associates the given location in the current frame with the given name.
<ul style="list-style-type: none"> ■ <code>JCFlow.markExactPageLocation(String name, JUnit.Point location)</code> ■ <code>JCFrame.markExactPageLocation(String name, JUnit.Point location)</code> ■ <code>JCPage.markExactLocation(String name, JUnit.Point location)</code> 	Associates the given location on the current page with the given name.

<mark-location> XML tag

The `<mark-location>` tag is used to specify a location when using XML. It has two required attributes: `location-name` and `location-type`.

- `location-name` defines a specific name for that location. The name must be unique.
- `location-type` must be one of `current`, `next`, `next-float`, `exact-frame`, or `exact-page`. Each value functions in the same fashion as the method to which the names correspond. When using one of `exact-frame` or `exact-page`, the locations should be specified using `exact-location-x` and `exact-location-y` attributes, where the units of the measurements is specified in the `unit` attribute.

For example:

```
<mark-location location-name="ch1" location-type="next" />
```

For more information on XML and the `<mark-location>` XML tag, see Chapter 13, [XML and JClass ServerReport](#).

4.2 Text and Images Links

You can create internal or external links from text or images in your document. The link can originate in the flow frame of the document or in a static frame. The link's destination can be any named location in the document or a URL to an external document.

To create a link, you use the `beginHyperlink()` and `endHyperlink()` methods in either `JCFlow` or `JCFrame`. Data added to the flow or frame between calls to `beginHyperlink()` and `endHyperlink()` are part of the link.

The basic method used to start a link is:

```
beginHyperlink(String destination, boolean external),
```

where `external` determines whether the link is internal or external to the document. If `external` is `false`, the link is internal and the destination is a named location within the document. When `external` is `true`, the destination is a URL (for example, `http://www.quest.com`).

After a `beginHyperlink()` method has been called in the flow, the hyperlink will continue until an `endHyperlink()` call is hit. This means that a hyperlink begun in the flow can carry across frames and pages. Hyperlinks begun in the frame are only active as long as that frame exists.

Note: Setting a link on a frame overrides any links that have been set on the flow. For example, in the following code, "There are" and "here!" would point to `link1`, while "two different links" would point to `link2`:

```
flow.beginHyperlink("link1");
flow.print("There are");
flow.getCurrentFrame().beginHyperlink("link2");
flow.print("two different links");
flow.getCurrentFrame().endHyperlink();
flow.print("here!");
flow.endHyperlink();
```

4.2.1 Using `beginHyperlink` to Format Hyperlink Text

The appearance of linked text depends entirely on the parameters defined in the initial `beginHyperlink()` method. There are three possibilities to choose from:

- `beginHyperlink(String destination, boolean external)`

This method turns on the default auto styling feature and draws all linked text blue and underlined, while retaining the other properties of the text style normally used to draw the text.

- `beginHyperlink(String destination, boolean external, JTextStyle linkStyle)`

This method uses the `linkStyle` parameter to define the text style applied to the linked text.

- `beginHyperlink(String destination, boolean external, boolean autoStyle)`

When `autoStyle` is `false`, this method turns off the default auto styling feature and draws the text normally. Therefore, the linked text will appear the same as all regular text in the document.

Note: Images, hRules, and other drawn objects are not modified when included in a link, regardless of which method was used to start the link. Therefore, they appear as they would normally.

4.2.2 <hyperlink> XML tag

By placing the `<hyperlink>` tags around text or other tags in the flow or frame, you have created a link. `<hyperlink>` has the following properties: `destination-name`, `destination-type`, `link-text-style-name`, and `auto-style`.

- `destination-name`, a required property, defines the name of the location or URL where the link points.
- `destination-type`, a required property, determines if the link is internal or external. Possible values are `internal` and `external`.
- `link-text-style-name` states which text style will be applied to linked text.
- `auto-style` turns on or off the auto styling feature. When set to `false`, it allows linked text to appear as regular text.

For example:

```
<hyperlink destination-name="http://www.quest.com"
           destination-type="external" auto-style="false">
  click here to visit the Quest Software web site.
</hyperlink>
```

For more information on XML and the `<hyperlink>` XML tag, see Chapter 13, [XML and JClass ServerReport](#).

4.3 Bookmarks

Note: This feature is supported only for PDF output.

Bookmarks (otherwise known as Outlines) are the collection of links pointing inside a document that resides outside of the document itself. These links are similar to a table of contents, and are accessed in Acrobat Reader in the Bookmarks tab.

Bookmarks are stored within JClass ServerReport as a tree of information and are set with the `JCDocument.setBookmarkTree()` method. A series of bookmarks must be organized as a tree, which must be an implementation of the `javax.swing.tree.TreeModel` interface. The root child node must be an instance of `DefaultMutableTreeNode`, but because it is ignored, it should have no user object associated with it.

All child nodes, except the root node, must have the `UserObject` property set to an instance of `com.klg.jclass.sreport.BookmarkNode`. (It is settable either through the constructor or the

`setUserObject()` method of `DefaultMutableTreeNode`.) This property should contain the text to draw with the node in the bookmark tree and the name of the link that the node points to.

Note: Because bookmarks are essentially internal links, a location must be set up for each bookmark. See Section 4.1, [Named Locations for Internal Links](#), for more information.

You can make the contents of the Bookmarks tab immediately visible in the Acrobat Viewer by making the following call:

```
JCDocument.setBookmarkTreeVisible(true)
```

By default, the value is `false`. Therefore, the Bookmarks tab is present, but not immediately selected.

4.3.1 BookmarkNode constructor

The `BookmarkNode` constructor has three properties that should appear in the following order:

- `text`, which determines the text drawn in the bookmark tree and displayed in the PDF viewer.
- `destination`, which determines the name of the resulting link.
- `expanded`, which determines if the node will be expanded or closed by default. The default value for `expanded` is `false`, meaning that the nodes will be closed.

Please note that the value for `expanded` has no effect on nodes that have no children.

4.3.2 Example

The following is a code excerpt that shows how to set bookmarks in your PDF. This example assumes that locations exist for `ch1`, `sec1-1`, and `ch2`.

```
JCDocument document;  
JCFlow flow;  
...  
DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode(" ");  
TreeModel bookmarkTree = new DefaultTreeModel(rootNode);  
DefaultMutableTreeNode ch1Node =  
    new DefaultMutableTreeNode(  
        new BookmarkNode("Chapter 1", "ch1", true));  
rootNode.add(ch1Node);  
DefaultMutableTreeNode sec1Node =  
    new DefaultMutableTreeNode(  
        new BookmarkNode("Section 1.1", "sec1-1", true));  
ch1Node.add(sec1Node);  
DefaultMutableTreeNode ch2Node =  
    new DefaultMutableTreeNode(  
        new BookmarkNode("Chapter 2", "ch2", true));  
rootNode.add(ch2Node);  
  
document.setBookmarkTree(bookmarkTree);
```

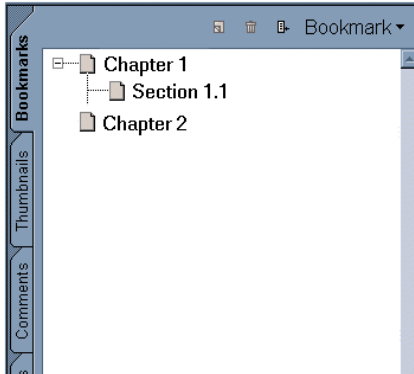


Figure 15 The appearance of the previous bookmark code in the Acrobat Viewer.

<bookmark-tree> and <bookmark> XML tags

`<bookmark-tree>` only has one attribute, `visible`, which can make the contents of the Bookmarks tab immediately visible in the Acrobat Viewer.

A `<bookmark-tree>` can contain any number of `<bookmark>` tags, which are used to specify the nodes in the bookmark tree. (`<bookmark>` tags can contain any number of other `<bookmark>` tags; hence the tree structure.) The following parameters are available for `<bookmark>` tags:

- `bookmark-text`, a required property, provides the text to be displayed and associated with the link in the Acrobat Viewer.
- `destination-name`, a required property, defines the location that the bookmark points to.
- `expanded`, an optional property, determines if the node will be expanded or closed by default in the Acrobat Viewer.

The following example would produce the same output as Figure 15:

```
<bookmark-tree visible="true">
  <bookmark bookmark-text="Chapter 1" destination-name="ch1"
    expanded="true">
    <bookmark bookmark-text="Section 1.1"
      destination-name="sec1-1" />
  </bookmark>
  <bookmark bookmark-text="Chapter 2" destination-name="ch2" />
</bookmark-tree>
```

For more information on XML and the `<bookmark-tree>` XML tag, see Chapter 13, [XML and JClass ServerReport](#).

4.4 Table of Contents and Other Reference Lists

Note: This feature is supported only for PDF output.

In a PDF, a table of contents is a set of links to the important headings within a document. The table of contents can be included within the document itself as part of the front matter and can also be displayed in the Bookmarks tab of Acrobat Reader (or a similar tab in any PDF viewer). Often there will be other reference lists included after the table of contents, such as a list of figures or list of tables. A table of contents and other reference lists usually include page numbers and are often hierarchical.

In JClass ServerReport, the table of contents and other reference lists are referred to as *contents lists*. You create contents lists using the `JCContentsList` and `JCContentsListEntry` objects. Contents lists are part of the front matter of the document, rather than the body of the document.

Note: You can only include contents lists in the front matter of a document. If you want a contents list within the body of a document, such as a chapter table of contents, you need to create a macro or use lists with hyperlinks.

All the examples in this section are taken from `JCLASS_SERVER_HOME/examples/sreport/PolarBearsBase.java`. The output of the table of contents is shown in Figure 16.

Table of Contents		
	Polar Bears	1
1.1	Where Polar Bears are found	2
1.2	How Polar Bears give birth	2
1.3	What kind of house the Polar Bear has	3
1.4	Special things about a Polar Bear's fur	3
1.5	How big a Polar Bear is	3
1.6	Special feet	3
1.7	What a Polar Bear eats	4
1.8	How a Polar Bear gets around	4
1.9	Conclusion	4

Figure 16 Sample table of contents

4.4.1 Creating Entries with `JCContentsListEntry`

For each entry in a contents list, you create a `JCContentsListEntry` object, set its properties, and add it to the contents list.

Note: `JCContentsListEntry` objects can only be associated with content in the body flow of the document. Any entries in the front matter flow are ignored.

Constructors and Properties

The minimum constructor for `JCContentsListEntry` requires three properties:

```
public JCContentsListEntry(int level, String text, String location)
```

Other constructors are:

```
public JCContentsListEntry(int level, String tag, String text,  
                           String location)
```

and

```
public JCContentsListEntry(int level, String tag, String text,  
                           String location, JCTextStyle textStyle,  
                           int autoNumbering)
```

where

- `level` is the entry's position in the hierarchy. For example, in a book's table of contents, level 0 may be the chapter titles, level 1 the section headings, level 2 the sub-sections, and so forth for as many levels as are required. Value must be non-negative.
- `tag` contains text that will appear before the text of the entry. For example, in a book's table of contents, the chapter level entries could be prefixed with the tag "Chapter." In this case, we probably also want to set the `autoNumbering` property to `AUTO_NUMBERING_ON`, so that the chapter entries are numbered sequentially.
- `text` is the entry description. In a book, this might be the same as a chapter title or a section heading. To treat the text as a link, set `hyperLinkUsed` to `true`.
- `location` is a named location within the document. The location determines the page number that appears for the entry. For more information, see Section 4.1, [Named Locations for Internal Links](#).
- `textStyle` defines the style to use for the entry. The left indent, first two tab stops, and the font are used to render the entry in the contents list. If null, the default `textStyle` for the level is used. Default text styles exist for the first four levels. You can also set text styles using `contentsList.setDefaultTextStyle(level, textStyle)`, where `level` is a non-negative integer. For more information, see Chapter 2, [Defining Text-Based Content](#).
- `autoNumbering` controls whether and how entries are numbered. The `autoNumbering` property can be set to one of the following enums: `AUTO_NUMBERING_NONE` (no numbering), `AUTO_NUMBERING_ON` (sequential numbering by level), and `AUTO_NUMBERING_PARENT` (takes the tag of an entry one level up and appends a number, so if the previous entry was 4.8, this entry might be 4.8.1).

In addition, `JCContentsListEntry` offers the following properties and convenience methods:

- `tagPrinted` prints the tag in the contents list when `true` (default).
- `hyperLinkUsed` generates a link for the entry when set to `true`. Default is `false`.
- `pageBreak` generates a page break when `true`. Default is `false`. All other fields of the entry are ignored. You can also insert a page break using `addPageBreak()` located in `JCContentsList`.

- `blankLine` inserts a blank line when `true` (`pageBreak` must be `false`). Default is `false`. All other fields of the entry are ignored. You can also insert a blank line entry using `addBlankLine()` located in `JCContentsList`.

Example

The following examples shows a `doHeading()` method that creates headings and adds them as entries to a table of contents. The method takes the current `JCFlow` object, the text for the heading entry, a tag to appear before the heading text in the contents, and the location of the heading in the document. Notice that the programmer uses the `flow.markNextLocation()` method to create unique locations for headings. In the `JCContentsListEntry` object, all entries are treated as level 1. Entries are also hyperlinked and autonumbered.

Note: While this example creates entries for a table of contents, you can create a list of images or list of figures in exactly the same way. For an example, see the `PolarBearBase.java` sample program.

```
protected void doHeading(JCFlow flow, String text, String tag,
                        String location) {
    // Set heading style and flow the heading
    // Note: It is important to follow the sequence:
    // (i) set style
    // (ii) new line (makes the new style active)
    // (iii) set location marker for contents list
    // (iv) flow the content
    // Set the location marker immediately before the content that it marks
    flow.setCurrentTextStyle(JCTextStyle.HEADING3);
    flow.newLine();
    flow.markNextLocation(location);
    flow.print(text + ":");
    flow.newLine();
    // Set up the ToC entry
    JCContentsListEntry entry = new JCContentsListEntry(1, tag, text,
                                                         location);
    // Define additional properties on the entry
    entry.setAutoNumbering(JCContentsListEntry.AUTO_NUMBERING_PARENT);
    entry.setHyperlinkUsed(true);
    tableOfContents.add(entry);
}
```

The last line of code adds the entry to a `JCContentsList` object called `tableOfContents`. `JCContentsList` is covered in the following section.

4.4.2 Creating a Contents List with JCContentsList

JCContentsList is an `ArrayList` that stores the contents list entries. Normally, you add entries while the document body is being flowed, but they can be added any time before flowing the front matter. Entries are displayed in the contents list in the order in which they were added to the array.

Constructors and Properties

The JCContentsList constructor can take one of two forms:

```
JCContentsList(String name, JCDocument parent)
```

or

```
JCContentsList(String name, JCDocument parent, List<JCPage> templates)
```

where

- `name` is the name of the contents list.
- `parent` is the document to which the contents list belongs.
- `templates` is a list of page templates to use for the contents list. If the contents list does not have at least one page template associated with it, JClass ServerReport uses the first template in the list of templates for the front matter flow. For more information, see Chapter 7, [Defining Page Templates](#).

You can also set the name of the page template to use after the contents list is printed, and a list of default text styles to use for each heading level. For more information, look up JCContentsList in the *API documentation*.

Note: JCContentsList extends `ArrayList<JCContentsListEntry>`, so it also offers all the properties of an `ArrayList`.

Example

The following example shows how to instantiate a JCContentsList object and define its properties:

```
// Instantiate a JCContentsList object
protected JCContentsList tableOfContents;
...
// Define the properties (where templateList is a list of templates)
tableOfContents = new JCContentsList("toc", document, templateList);
```

Alternatively, you can create a JCContentsList object on the fly. In this case, you add an entry using the parent document's `addContentsListEntry()` method. The method takes a name for the contents list and the entry and returns a JCContentsList object of that name containing one entry in its array.

```
// Create a contents list on the fly
JCContentsList listOfFigures =
    document.addContentsListEntry("ListOfFigures", entry);
```



```
// You add subsequent entries as usual
listOfFigures.add(newEntry);

// Or by another call to addContentsListEntry
document.addContentsListEntry("ListOfFigures", newEntry);
```

4.4.3 Printing the Contents List

While processing the body flow, JClass ServerReport determines the page number for each entry in each contents list. When JClass ServerReport prints the contents list to the front matter flow, it uses the styles associated with the heading level plus the settings of other JCContentsListEntry properties to lay out the contents list on the page.

By default, the entries in a contents list are laid out as follows:

- The tag, if visible, prints at the left indent of the text style.
- The text begins at the first tab stop of the text style or, in the absence of tabs, two spaces after the tag. The text wraps if necessary. If `hyperlinkUsed` is true, the text is a link and is shown with the default text style for linked text.
- The page number is displayed at the second tab stop or, in the absence of tabs, two spaces after the end of the text. If the text wraps, the page number is added after the wrapped text.

Example

The following code snippet shows the start of the front matter flow and the calls required to print a table of contents. In this example, a page template called “text” is specified. If a page template is not specified, the default page template is used.

```
// Begin the front matter flow.
flow = document.beginFrontMatter(templatelist, null);

// Write the title page.
writeTitlePage(flow);

// Specify which template should be used next
// (after the contents list is printed)
tableOfContents.setFlowPageTemplate("text");

// Print out the table of contents.
flow.print(tableOfContents);

...
// End the flow
document.endFrontMatter();
```

4.4.4 Adding a Contents List to the Bookmark Tab

JCContentsList offers a convenience method, called `createBookmarkTree()`, that you can use to create a bookmark tree from the contents list. You need to end the front matter flow before creating the bookmark tree. For more information, see Section 4.3, [Bookmarks](#).

Example

The following example shows how a table of contents and a list of figures can be added to the Bookmark tab of Acrobat Reader.

```
// Create a bookmark tree from the two contents lists. Create a root and
// graft the two trees into the root.
DefaultMutableTreeNode bookmarkRoot = new DefaultMutableTreeNode("");

// JCContentsList has a convenience method to create a bookmark tree
// from its contents.
tableOfContents.createBookmarkTree(bookmarkRoot, true);

// Create a temporary bookmark root for the figures tree.
DefaultMutableTreeNode tempBookmarkRoot = new DefaultMutableTreeNode("");
listOfFigures.createBookmarkTree(tempBookmarkRoot, true);

// Since the all the elements of the Figures list are at level 1, the
// convenience method will have created a dummy parent node at level 0 to
// hold the children. Get this node, replace the dummy text, and add it
// into the original bookmark root.
DefaultMutableTreeNode n =
    (DefaultMutableTreeNode)tempBookmarkRoot.getLastChild();
n.setUserObject(new BookmarkNode("Figures", null, true));
bookmarkRoot.add(n);
tempBookmarkRoot.removeAllChildren(); // So it can be GCed

// For debugging, JCPrinter has a method to print out a bookmark tree.
// try {
//     JCPrinter.outputBookmarkTree(System.out, bookmarkRoot, -1);
// }
// catch (IOException e) {
//     e.printStackTrace();
// }

// From the bookmark root create a tree model to send to the document.
DefaultTreeModel bookmarkTree = new DefaultTreeModel(bookmarkRoot);
document.setBookmarkTree(bookmarkTree);
document.setBookmarkTreeVisible(true);
```

Defining Table-Based Content

[Table Structure](#) ■ [JCFlowTable versus JCPageTable](#) ■ [JCFlowTable Overview](#)
[JCPageTable Overview](#) ■ [Customizing Tables](#) ■ [Customizing Cells](#) ■ [Table Wrapping](#)
[Converting Tables](#) ■ [Using Tables With XML](#)

The `JCFlowTable` and `JCPageTable` classes provide methods and attributes for printing tables from your Java application. This chapter shows you how to:

- add a table to your document
- flow data into the table
- add a header row to the table
- add borders and background colors
- add a footer row to the table
- customize cell vertical alignment, margins, and borders
- span cells across columns
- span cells across rows (`JCPageTable` only)
- wrap table rows
- wrap table columns (`JCPageTable` only)
- convert table data from other Java sources

Please see the [API](#) (automatically installed in `JCLASS_SERVER_HOME/docs/api/`) for complete programming details about these classes.

Though `JCFlowTable` and `JCPageTable` appear to have many similarities, their differences are vast. Before deciding which method to use for creating your table, see Section 5.2, [JCFlowTable versus JCPageTable](#).

5.1 Table Structure

The following information applies to both `JCFlowTable` and `JCPageTable`. Exceptions are clearly noted in the text.

A table is created with a given number of columns, and a column object is created for each column. The number of rows in a table is not fixed, and row objects are added to the table explicitly by using a method such as `JCPageTable.addRow()`. For `JCPageTable` only, objects can also be added to the table implicitly, through reference to cell objects beyond the current last row of the table. The table's `rowList` attribute stores all the rows created within the table.

`JCFlowTable` only holds onto a **minimal number of rows at a time**, disposing of them as they are printed. For `JCFlowTable`, the current row can be obtained by calling the `getRow()` method.

With `JCPageTable`, **rows are created individually, but are then saved in memory**, being printed/flowed when requested.

Cells in a row are stored in lists associated with that row. A row object with no cells containing content will have a cell list with no entries. As cells are created, the cell object will be stored at the correct position in the row's `cellList`. As with the table's `rowList`, blank (null) entries are created for non-existent cells. The content of a cell can be stored in a `JCFrame` object allocated by the cell; as a standard `JCFrame`, it can contain any content that is valid for a `JCFrame` object, up to and including a `JCPageTable` (*not* a `JCFlowTable`, which can only be added to the flow).

Note: If you want to create an accessible PDF document, you cannot use `JCFrame` objects within cells. For more information, see [Tables in Structured PDF Documents](#), in Chapter 10.

In addition to ordered lists of row and column objects, the table object can also (conditionally) own a `JCPageTable` object. This object defines the headers of the table, but a header `JCPageTable` cannot own a header table. The structure of a header is identical to that of a normal `JCPageTable`, except that the column layout of the header table is defined implicitly by reference to the parent table.

There are also dummy cell objects attached to the table, and to each row and column object. These cell objects store column-, row-, and table-wide default cell attributes.

5.2 JCFlowTable versus JCPageTable

Tables in `JClass ServerReport` are implemented in two classes: `JCFlowTable` and `JCPageTable`.

With `JCFlowTable`, **rows are created individually and are printed/flowed as each one is finished**. `JCFlowTable` allows you to modify only one row at a time. `JCFlowTable` also allows column spanning, but not row spanning.

With `JCPageTable`, **rows are also created individually, but are then saved in memory, being printed/flowed when requested**. `JCPageTable` allows you to make alterations to cells in rows and columns before printing. `JCPageTable` allows both row and column spanning. With `JCPageTable`, you can work with formulas.

Here is a summary of the differences:

	JCFlowTable	JCPageTable
Rows are flowed/printed as each row is created.	Yes	No
Entire table needs to be in memory before being passed to the document flow.	No	Yes
Allows modification of more than one row at a time.	No	Yes
Allows row spanning.	No	Yes
Allows column spanning.	Yes	Yes
Allows alteration of any area of table before printing.	No	Yes
Allows you to incorporate formulas.	No	Yes
Prints to the flow.	Yes	Yes
Prints to a frame.	No	Yes
Allows use of vertical cell borders of varying widths.	No	Yes

Should you use JCFlowTable or JCPageTable?

Use JCFlowTable if memory is an issue or if the table is large.

Use JCPageTable if precise formatting is important, if the data is nonlinear, or if you want to be able to reuse the JCPageTable.

5.2.1 Attribute Overview

For both JCFlowTable and JCPageTable, attributes such as background color, cell margins, and border styles can be set on any object in the table hierarchy (JCFlowTable, JCPageTable, Row, Column, or Cell). The most specific setting available in a particular cell is used when the table is drawn, according to the inheritance order. The standard accessor methods (Cell.getBackgroundColor(), for instance) will return the setting that will be applied to that object, even if the value is inherited from a higher-level object. All get() methods for inheritable attributes are shadowed by a method that determines whether the value returned by the get() method is inherited (versus being locally specified, for instance, Cell.isBackgroundColorInherited()).

Because of the ambiguity created by inheriting Cell attributes from both the Row and Column, the setRowColumnDominance(int) method takes the constants ROW_DOMINANCE and COLUMN_DOMINANCE to define whether values are inherited first from the Row or the Column, respectively.

5.3 JCFlowTable Overview

JCFlowTable provides methods for creating and customizing tables into which data is flowed. Content may be added in the form of formatted, flowed text. Note that cell renderers may not be used with JCFlowTable.

JCFlowTable contains an ordered set of AbstractPageTable.Column objects that define the columns of the table and their attributes, along with one row object – AbstractPageTable.Row – that contains the cells of the current row. A row contains both the row attributes and the list of AbstractPageTable.Cell objects defined in that row. Table content is stored in JCFrame objects belonging to individual cells. Each row is printed to the flow when finished. Headers are implemented using a nested JCPageTable object.

Note: JCFlowTable does not support the use of a TableDataModel.

5.3.1 Using JCFlowTable

1. Create a JCFlowTable object and pass in:
 - the document to which this table belongs
 - the number of columns in this table
 - column widths
2. Create table header by calling createHeaders(), if desired.
3. Customize the table by setting borders and margins on the table or on individual columns, if desired.
4. Size table to frame using fitToFrame(), if desired.
5. Add a row to the table by calling addRow().

You can use the addRow() method to pass in a JTextStyle class, which specifies the appearance of the text in the cells of the table row. You can also pass text into cells using this method.

Alternatively, once you have added a row to the table by calling addRow(), you can customize the cells in the row by setting row or cell borders or by setting up column spans. (**Note:** The vertical cell borders can be no larger than the border set on the corresponding column.) Afterwards, you can add text to each cell in the row by calling addCells(), which lets you pass in a JTextStyle class that will control the appearance of text in the cells of the table row.

6. Call endRow() when you are done with all changes in that row.
7. Repeat steps 2 and 3 for all rows in the table.
8. Call endTable() when you are done with the table.

5.3.2 Creating a Table with JCFLOWTable

To create a table using `JCFLOWTable`, you need to create a `JCFLOWTable` object then call the methods required to build and modify one row at a time.

The `getRow()` method returns the current working row if one exists; if not, this method creates a row. Note that both the `getCell()`¹ and `getCellFrame()` methods cause `getRow()` to be called.

The `addRow()` method closes off the current working row if one exists and creates a new one.

The `endRow()` method closes off the current working row but does not create a new row. When a row is closed off, the row before it is sent to the flow. The `endTable()` method notifies the `JCFLOWTable` class to close off the entire table, sending the remaining rows to the flow.

To create the table in the above example, define a table with three columns, each of which is one inch wide. Here is the relevant code.

```
JCFLOWTable table = new JCFLOWTable(document, 3,  
    new JUnit.Measure(JUnit.INCHES, 1.0));
```

The `JCFLOWTable` parameters indicate the `JCDocument` to which this table belongs and that this table should contain three columns, each one inch wide.

Creating Body Rows

With `JCFLOWTable`, rows are created individually and are printed/flowed as each one is finished. This means that cell styles and contents must be individually set before the row is flowed. The user will need to inform the `JCFLOWTable` when finished with adding rows to it. The flow is retrieved from the current document.

Methods such as `printToCell()` will affect the current row only. A call to `endRow()` will send the current row to the flow. Likewise, if `addRow()` is called twice in a row without `endRow()` being called, the previously created row will be automatically flowed. For example:

```
table.addRow();  
table.printToCell(0, style, "Labrador");  
table.printToCell(1, style, "Persian");  
table.printToCell(2, style, "Shetland");  
table.endRow();  
table.addRow();  
table.printToCell(0, style, "Collie");  
table.printToCell(1, style, "Siamese");  
table.printToCell(2, style, "Arabian");  
table.endRow();  
table.addRow();  
table.printToCell(0, style, "Terrier");  
table.printToCell(1, style, "Calico");  
table.printToCell(2, style, "Clydesdale");  
table.endRow();
```

1. The number specified in `getCell` is the column number, not the row number.

populates the following table:

Labrador	Persian	Shetland
Collie	Siamese	Arabian
Terrier	Calico	Clydesdale

Figure 17 Table with body rows.

When you are finished with the table, call the `table.endTable()` method.

JClass ServerReport identifies each cell by numbering cell columns, starting from zero (0).

Note that if a row cannot fit in the current page, then that row will be placed on the following page. However, if the height of that row is larger than the height of the entire page (for example, in the case that the row contains particularly extensive data, such as a long String), then that row is truncated.

Adding Body Rows

You can populate an entire row of cells to the end of a table using `JCFlowTable.addRow()` or `JCPageTable.addRow()`.

Here is a code snippet showing this method in `JCFlowTable`:

```
table.addRow(style, newString[]{"Setter", "Tabby", "Palomino" });
table.endRow();
```

Here is a code snippet showing this method in `JCPageTable`:

```
table.addRow(style, newString[]{"Setter", "Tabby", "Palomino"});
```

The example produces the following results:

Labrador	Persian	Shetland
Collie	Siamese	Arabian
Terrier	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 18 Table with additional body row.

Adding Header Rows

A header is an instance of `JCPageTable`. A header row is a separate table unto itself. You build the header table using `JCFlowTable.createHeaders()`. Once you have created the header table, you can populate its cells, much as you did the body rows.

```
JCPageTable header = table.createHeaders();

try {
```



```

JCFrame frame = header.getCellFrame(0, 0);
frame.print(style, "Dogs");
frame = header.getCellFrame(0, 1);
frame.print(style, "Cats");
frame = header.getCellFrame(0, 2);
frame.print(style, "Horses");
}
catch (EndOfFrameException e) {}

```

Adding the header row produces the following results:

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Terrier	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 19 Table with header row.

Printing data into specific cells

To print data into specific cells, use the `JCFlowTable.printToCell()` method. It prints text to the cells you specify in the current row, generating cells if necessary.

The content of a cell can be stored in a `JCFrame` object allocated by the cell; as a standard `JCFrame`, it can contain any content that is valid for a `JCFrame` object, up to and including a `JCPageTable`. Please note that `table.printToCell()` can not be used when adding an object to a cell; instead, get the frame of the cell (`JCFrame frame = table.getCellFrame(0,0);`) and then add the object as normal to the frame; for example:

```
frame.embedComponent(table.getDefaultStyle(), new JButton("example"));
```

Note: A `JCFlowTable` cannot be added to a cell; it can only be added to the flow.

For instance, for `JCFlowTable`, to print the text “Calico”, using the current `JCTextStyle` (`style`), to the cell found in column 1 of the current row, you would use this code:

```
table.printToCell(1, style, "Calico");
```

Note: Cell indices start at column 0, independent of the presence of a header.

5.4 JCPageTable Overview

`JCPageTable` also provides methods for creating and customizing tables into which data is flowed. Content may be added in the form of formatted, flowed text, or it may be added as content objects to the underlying `TableDataModel`, in which case the appropriate `CellRenderers` are used to render/lay out the content.

A `JCPageTable` consists of an ordered set of `JCPageTable.Column` objects that define the columns of the table and their attributes, along with an ordered set of `JCPageTable.Rows`. A row contains both the row attributes and the list of `JCPageTable.Cell` objects defined in that row. Headers are implemented using a nested `JCPageTable` object.

5.4.1 Using `JCPageTable`

1. Create a `JCPageTable` object and pass in:
 - the document to which this table belongs
 - the number of columns in this table
 - column widths
2. Create table header by calling `createHeaders()`, if desired.
3. Customize cells by setting borders and margins on table, rows, columns, or individual cells and by setting up any cell spans that are required.
4. Size table to frame using `fitToFrame()`, if desired.
5. Add data to tables (for example, call `JCPageTable.printToCell()` or `JCPageTable.addRow()`).
6. Print the table by calling `flow.print(table)`, where `flow` is an instance of `JCFlow`.

5.4.2 Creating a Table Using `JCPageTable`

To create a table using `JCPageTable`, you need to create a `JCPageTable` object then call the methods required to build and modify cells within the table.

The `getRow()` method gets a row from the table if one exists; if not, this method creates a row.

The `addRow()` method adds a row of cell values (or as many values as are provided) to the table starting at a given position.

The `print()` method in the `JCFlow` object prints the table to the flow.

For example, to define a table with three columns, each of which is one inch wide, the following code may be used:

```
JCPageTable table = new JCPageTable(document, 3,  
    new JCUnt.Measure(JCUnt.INCHES, 1.0));
```

The `JCPageTable` parameters indicate the `JCDocument` to which this table belongs and that this table should contain three columns, each one inch wide.

Creating Body Rows

With `JCPageTable`, rows are created individually, but are then saved in memory, being printed/flowed when requested.

When you print to a cell frame that does not yet exist, you create a row of cells that correspond to the number of columns in the table. For example:

```

table.printToCell(0, 0, style, "Labrador");
table.printToCell(0, 1, style, "Persian");
table.printToCell(0, 2, style, "Shetland");
table.printToCell(1, 0, style, "Collie");
table.printToCell(1, 1, style, "Siamese");
table.printToCell(1, 2, style, "Arabian");
table.printToCell(2, 0, style, "Terrier");
table.printToCell(2, 1, style, "Calico");
table.printToCell(2, 2, style, "Clydesdale");

```

populates the following table:

Labrador	Persian	Shetland
Collie	Siamese	Arabian
Terrier	Calico	Clydesdale

Figure 20 Table with body rows.

To print the table, call

```
flow.print(table);
```

where `flow` is an instance of `JCFlow`.

`JClass ServerReport` identifies each cell by numbering cells in a row, column format, starting from (0, 0).

Note that if a row cannot fit in the current page, then that row will be placed on the following page. However, if the height of that row is larger than the height of the entire following page (for example, in the case that the row contains particularly extensive data, such as a long String), then that row is truncated.

Adding Body Rows

You can populate an entire row of cells to the end of a table using `JCFlowTable.addRow()` or `JCPageTable.addRow()`.

Here is a code snippet showing this method in `JCFlowTable`:

```

table.addRow(style, newString[]{"Setter", "Tabby", "Palomino" });
table.endRow();

```

Here is a code snippet showing this method in JCPageTable:

```
table.addRow(style, newString[]{"Setter", "Tabby", "Palomino"});
```

The example produces the following results:

Labrador	Persian	Shetland
Collie	Siamese	Arabian
Terrier	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 21 Table with additional body row.

Adding Header Rows

A header is an instance of JCPageTable. A header row is a separate table unto itself. You build the header table using JCPageTable.createHeaders(). Once you have created the header table, you can populate its cells, much as you did the body rows.

```
JCPageTable header = table.createHeaders();

try {
    JFrame frame = header.getCellFrame(0, 0);
    frame.print(style, "Dogs");

    frame = header.getCellFrame(0, 1);
    frame.print(style, "Cats");

    frame = header.getCellFrame(0, 2);
    frame.print(style, "Horses");
}
catch (EndOfFrameException e) {}
```

Adding a header row produces the following result:

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Terrier	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 22 Table with header row.

Adding Footer Rows

A footer is much like a header, it is also an instance of `JCPageTable`. A footer row is a separate table unto itself. You build the footer table using `JCPageTable.createFooters()`. Once you have created the footer table, you can populate its cells, much as you did the body rows.

```
JCPageTable footer = table.createFooters();

try {
    JCFrame frame = footer.getCellFrame(0, 0);
    frame.print(style, "Dogs");

    frame = footer.getCellFrame(0, 1);
    frame.print(style, "Cats");

    frame = footer.getCellFrame(0, 2);
    frame.print(style, "Horses");
}

catch (EndOfFrameException e) {}
```

Adding a footer row produces the following result, where the first row is a header and the last row is a footer.

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Terrier	Calico	Clydesdale
Setter	Tabby	Palomino
Dogs	Cats	Horses

Figure 23 Table with a footer row.

Printing data into specific cells

To print data into specific cells, use the `JCPageTable.printToCell()` method. It prints text to the cells you specify in the current row, generating cells if necessary.

The content of a cell can be stored in a `JCFrame` object allocated by the cell; as a standard `JCFrame`, it can contain any content that is valid for a `JCFrame` object, up to and including a `JCPageTable`. Please note that `table.printToCell()` can not be used when adding an object to a cell; instead, get the frame of the cell (`JCFrame frame = table.getCellFrame(0,0);`) and then add the object as normal to the frame; for example:

```
frame.embedComponent(table.getDefaultStyle(), new JButton("example"));
```

For `JCPageTable`, to print the text “Calico”, using the current `JCTextStyle(style)`, to the cell found in row 2, column 1, you would use this code:

```
table.printToCell(2, 1, style, "Calico");
```

Note: Cell indices start at (0, 0), independent of the presence of a header row.

5.4.3 Table Styles

Select a table style before anything is flowed to the table.

The simplest way to customize a table is to choose one of the numerous built-in table styles in `JTableStyle`. Table styles are constants, such as `JTableStyle.STYLE_DEFAULT` or `JTableStyle.STYLE_n`, where *n* ranges from 1 to 17.

Scope can be applied to table styles to prevent collisions between identically named styles in a multi-threaded environment, such as a servlet. Scope is defined by the object in which the styles are used. For example, a `JCDocument`, and instance of `ServletConfig`, or null (specifying global scope).

Table styles are cloneable. You can further customize a table style by cloning the one you wish to use as a base for making changes.

```
// Create a table style with orange headers and alternating colored
// rows
JTableStyle tableStyle =
    (JTableStyle)JTableStyle.STYLE_DEFAULT.clone();
tableStyle.getHeaderStyle().setBackground(Color.orange);
tableStyle.setAlternate( new JCAlternate(Color.lightGray,
    Color.white, true));
```

The following table lists the stock table styles and shows a sample of each style.

Style (name property)	Description	Example																				
STYLE_DEFAULT (default)	<div><div>■ thin left, right, top, bottom, horizontal, header, and column borders</div><div>■ no column shading</div><div>■ regular heading font</div></div>	<div>-- default --</div> <table><tr><td>Column 1</td><td>Column 2</td><td>Column 3</td><td>Column 4</td><td>Column 5</td></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_0 (Style 0)	<div><div>■ no left, right, top, bottom, header, horizontal, or column borders</div><div>■ no column shading</div><div>■ regular heading font</div></div>	<div>-- Style 0 --</div> <table><tr><td>Column 1</td><td>Column 2</td><td>Column 3</td><td>Column 4</td><td>Column 5</td></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

Style (name property)	Description	Example																				
STYLE_1 (Style 1)	<ul style="list-style-type: none">thick header border; no other bordersno column shadingregular heading font	<div>-- Style 1 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_2 (Style 2)	<ul style="list-style-type: none">thin header border; thick top and bottom borders; no other bordersno column shadingregular heading font	<div>-- Style 2 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_3 (Style 3)	<ul style="list-style-type: none">thick top, bottom, and header borders; thin horizontal borders; no column, left, or right bordersno column shadingregular heading font	<div>-- Style 3 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_4 (Style 4)	<ul style="list-style-type: none">no left, right, top, bottom, horizontal, or column borders; thick header borderheader colored (black); reverse type for header textno column shading	<div>-- Style 4 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_5 (Style 5)	<ul style="list-style-type: none">thick right, left, bottom, and header borders; no top, column, or horizontal bordersheader colored (black); reverse type for header textno column shading	<div>-- Style 5 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

Style (name property)	Description	Example																				
STYLE_6 (Style 6)	<ul style="list-style-type: none">thick right, left, bottom and header borders; thin horizontal and column borders; no top borderheader colored (black); reverse type for header textno column shading	<div>-- Style 6 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_7 (Style 7)	<ul style="list-style-type: none">thick right, left, top, bottom and header borders; no horizontal or column bordersheader colored (gray); reverse type for header textno column shading	<div>-- Style 7 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_8 (Style 8)	<ul style="list-style-type: none">thick right, left, top, bottom, and header borders; thin horizontal and column bordersheader colored (gray); reverse type for header textno column shading	<div>-- Style 8 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_9 (Style 9)	<ul style="list-style-type: none">thin header border; thick top, and bottom borders; no right, left, horizontal, and column bordersheader colored (gray); reverse type for header textno column shading	<div>-- Style 9 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_10 (Style 10)	<ul style="list-style-type: none">thick right, left, top, and bottom borders; thin header, horizontal, and column bordersregular heading fontno column shading	<div>-- Style 10 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

Style (name property)	Description	Example																				
STYLE_11 (Style 11)	<ul style="list-style-type: none">no right, left, top, and bottom borders; thin header, horizontal, and column bordersplain headersno column shading	<div>-- Style 11 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_12 (Style 12)	<ul style="list-style-type: none">thick right, left, top, bottom, and header borders; no horizontal or column bordersheader colored (gray); reverse type for header textgray column shading	<div>-- Style 12 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_13 (Style 13)	<ul style="list-style-type: none">thick right, left, bottom, and header borders; thin horizontal border; no top or column bordersheader colored (black); reverse type for header textno column shading	<div>-- Style 13 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_14 (Style 14)	<ul style="list-style-type: none">thin right, left, top, bottom, and horizontal borders; thick header border; no column borderplain headerno column shading	<div>-- Style 14 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_15 (Style 15)	<ul style="list-style-type: none">thin right, left, top, bottom, header, and column borders; no horizontal borderplain headerno column shading	<div>-- Style 15 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

Style (name property)	Description	Example																				
STYLE_16 (Style 16)	<ul style="list-style-type: none">■ thin top, bottom, left, right, and horizontal borders; thick header border; thin border between last two columns■ header colored (dark gray); reverse type for header text■ no column shading■ alternate row shading (light gray and dark gray)	<p>-- Style 16 --</p> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
STYLE_17 (Style 17)	<ul style="list-style-type: none">■ thin top, bottom, right, left, and horizontal borders; thick header border; thin border between last two columns■ header reverse type■ alternate column shading (light gray and dark gray)	<p>-- Style 17 --</p> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

JTableStyle Methods

Method	Description
setAlternate()	See Section 5.5.1, Alternating Row or Column colors , for details.
setBackground()	Sets the background color for a table.
setBottomBorder()	Sets the bottom perimeter border of a table.
setBottomBorderEnabled()	Turns the bottom perimeter border of a table on or off.
setColumnBorder()	Sets the vertical border used by the internal table cells.
setHeaderBorder()	Sets the border between the header and the first row of the table.
setHeaderBorderEnabled()	Turns the border between the header and the first row of the table on or off.
setHeaderStyle()	Sets a JTableStyle instance to be used as a table style for the header table.
setLeftBorder()	Sets the left perimeter border of a table.

Method	Description
<code>setLeftBorderEnabled()</code>	Turns the left perimeter border of a table on or off.
<code>setName()</code>	Sets the name of the table style.
<code>setRightBorder()</code>	Sets the right perimeter border of a table.
<code>setRightBorderEnabled()</code>	Turns the right perimeter border of a table on or off.
<code>setRowBorder()</code>	Sets the default border between table rows.
<code>setTextStyle()</code>	Sets the default text style to be used when drawing text in table cells.
<code>setTopBorder()</code>	Sets the top perimeter border of a table.
<code>setTopBorderEnabled()</code>	Turns the top perimeter border of a table on or off.

Setting Scope and Cleaning Up JTableStyle Objects

To set the scope of a table style, simply set it at the same time the style is named, using the `setNameAndScope(scope)` method.

After a program is finished with the scope object used to create the table style, it should be cleaned up using the `JTableStyle.cleanup(scope)` method, where `scope` refers to the scope that was applied to the style when it was created. For example, `JTableStyle.cleanup(document)` cleans up all table styles that are set at the document level, and should be called after the document has been generated.

Row/Column Dominance

To control the order of border and cell color drawing, use the `RowColumnDominance` property of the table. A value of `ROW_DOMINANCE` forces column background colors and borders to draw before the equivalent properties of the table's row. In contrast, a value of `COLUMN_DOMINANCE` forces row background colors and border to draw before the equivalent properties of the table's columns. This property is set on the table class (`JCPageTable` or `JCFlowTable`) with the `setRowColumnDominance()` method.

5.5 Customizing Tables

Once the table is laid out and has had data flowed into it, you may want to customize its appearance by defining borders and background colors, or by defining various line styles and shadings. This section describes how.

Again, please recall that with `JCFlowTable`, you can apply changes to only the current row.

Before anything is flowed to the table, you need to:

- select a table style (see Section 5.4.3, [Table Styles](#));
- choose a border style (see Section 5.5.2, [Adding Borders](#));
- adding a header border (see Section 5.5.3, [Adding Header Borders](#));
- apply background colors (see Section 5.5.4, [Applying Background Colors](#)).

5.5.1 Alternating Row or Column colors

It is often desirable to shade every other row or column differently for easier reading. The `JCAIternate` class is used for this purpose, and the `setAIternate()` method in `JCTableStyle` takes an instance of `JCAIternate` to specify the alternating colors. There are two default styles: one for columns and one for rows.

To show rows alternating between gray and light gray, use `JCAIternate.ROW`. To show columns alternating between gray and light gray, use `JCAIternate.COLUMN`. To choose your own colors, create an instance of `JCAIternate` and pass its constructor the two colors you want, and a Boolean flag specifying whether the alternation is to take place over rows or over columns. If the flag is set to `true`, rows alternate in color, if the flag is set to `false`, columns alternate in color. For example, the following causes rows to alternate between red and blue:

```
tableStyle.setAIternate(new JCAIternate(
    Color.red, Color.blue, true));
```

5.5.2 Adding Borders

Add borders before anything is flowed to the table.

To define the appearance of a table border, you must select or create a `JCDrawStyle` with the required attributes and apply the style to the border. For information on controlling the appearance of borders, please refer to Chapter 3, [Defining Image-Based Content](#).

Borders may be set on the `JCTableStyle` class (see Section 5.4.3, [Table Styles](#)) or via the convenience methods on the table itself.

```
table.setAllBorders(JCDrawStyle.LINE);
table.setLeftBorder(JCDrawStyle.LINE_2POINT);
table.setRightBorder(JCDrawStyle.LINE_2POINT);
```

This example uses `setAllBorders()` to apply the default single line style to all borders in the table, and then uses `setLeftBorder` and `setRightBorder` to apply thick borders to the external left and right sides of the table. If we were to apply this style to the table we developed earlier, the result would be as follows:

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Scottish	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 24 Table with borders.

JCDrawStyle makes the following line styles available for table borders:

Enum	Description
LINE_TYPE_BROKEN	Applies a dotted line to the table border.
LINE_TYPE_DOUBLE	Applies a double line to the table border.
LINE_TYPE_NONE	Blanks out the table border.
LINE_TYPE_SINGLE	Applies a single line to the table border.

JCFlowTable and JCPageTable provide numerous methods for applying line styles to table borders:

Method	Result
setAllBorders()	Applies the style to every border in the table.
setBottomBorder()	Applies the style to the perimeter bottom border of the table.
setColumnBorder()	Applies the style to all column borders, except for the perimeter borders Left and Right.
setEdgeBorders()	Applies the style to all borders on the perimeter of the table (Top, Bottom, Left, and Right).
setHeaderBorder()	Applies the style to the border between the header table and the table's first row.
setHorizontalBorder()	Applies the style to the Top, Bottom, and header borders.
setInternalBorders()	Applies the style to all row and column borders except for the perimeter borders.
setLeftBorder()	Applies the style to the table's left-hand perimeter border.
setRightBorder()	Applies the style to the table's right-hand perimeter border.

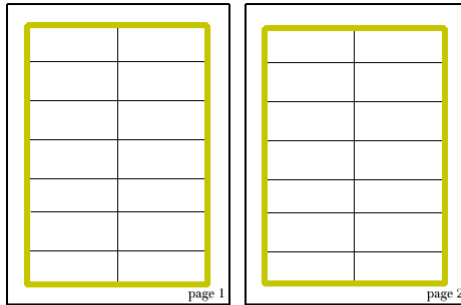
Method	Result
<code>setRowBorder()</code>	Applies the style to all table row borders except for the header border and the perimeter borders Top and Bottom.
<code>setTopBorder()</code>	Applies the style to the table's perimeter top border. If there is a header table, applies the style to the header table's top perimeter.
<code>setVerticalBorders()</code>	Applies the style to column, Left, and Right borders.

In addition to methods for specifying borders, `JCPageTable` and `JCFlowTable` also provide methods for turning perimeter borders off. When perimeter borders are turned off, they are replaced with the borders of the individual cells that lie underneath.

Method	Result
<code>setTopBorderEnabled()</code>	Turns the top perimeter border of the table on or off.
<code>setBottomBorderEnabled()</code>	Turns the bottom perimeter border of the table on or off.
<code>setLeftBorderEnabled()</code>	Turns the left perimeter border of the table on or off.
<code>setRightBorderEnabled()</code>	Turns the right perimeter border of the table on or off.
<code>setHeaderBorderEnabled()</code>	Turns the border between the header and the first row of the table body on or off.
<code>setEdgeBordersEnabled()</code>	Turns the top, bottom, left, and right perimeter borders on or off.

The behavior of perimeter borders at page breaks is determined by the `BorderMode` property of the table. A value of `BORDER_USE_EXTERNAL` forces perimeter borders to be used on each section of the multi-page table. A value of `BORDER_USE_INTERNAL` forces perimeter borders to be used only at the beginning and end of the table. Internal cell borders will be used for the table sections broken across pages. For example:

External



Internal

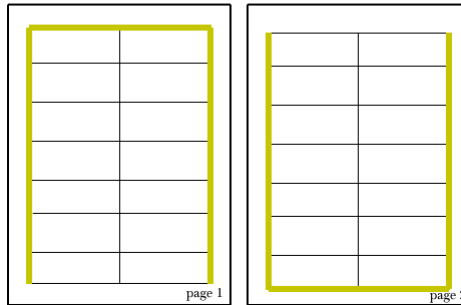


Figure 25 External and internal border behavior.

5.5.3 Adding Header Borders

Add header borders before anything is flowed to the table.

As mentioned in [Adding Header Rows](#), the header row is a separate table unto itself. Modifying header borders is just like modifying borders in the main table. For example, to apply a double line to the top and bottom of the header row, enter:

```
table.setTopBorder(JCDrawStyle.LINE_DOUBLE);  
table.setHeaderBorder(JCDrawStyle.LINE_DOUBLE);
```

The table appears as follows:

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Scottish	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 26 Table with double header borders.

5.5.4 Applying Background Colors

For `JCFlowTable`, apply background colors for columns before anything is flowed to the table. Background colors for individual rows or cells can be applied when the row in question is the current row.

You can further customize tables by applying background colors to cells, rows, columns, or the entire table. Colors and greyscale are defined by `java.awt.Color`.

For example, suppose you wanted to apply a green background to all of the cells in the first column of the table.

```
table.getColumn(0).setBackgroundColor(Color.green);
```

This example uses the `getColumn()` method to identify the column to be colored, and then uses the `setBackgroundColor()` method to apply the specified static color.

5.5.5 Adjusting the Size of a Table

Adjust the size of the table before anything is flowed to the table and after the table and its cells have been customized.

If you wish to adjust the size of a table so that it occupies all the space available in its parent frame, use the `fitToFrame()` method. The method takes two parameters: the frame to which the table should fit itself and the current `JTextStyle`. It should only be called after all table borders, cell borders, and margins have been finalized.

5.6 Customizing Cells

For `JCFlowTable`, customize cells in a row before anything is flowed to that row, and customize columns before anything is flowed to the table. Vertical cell borders can be no larger than the corresponding column border.

For `JCPageTable`, customize cells, rows, and columns before anything is flowed to the table.

`JCFlowTable` and `JCPageTable` each has three inner classes. These classes allow for more precise control over the description of individual table components.

Class	What methods in the class allow
<code>JCFlowTable.Cell</code> and <code>JCPageTable.Cell</code>	Allow customization of cell appearance, including settings for vertical alignment, borders, margins, and spans.
<code>JCFlowTable.Column</code> and <code>JCPageTable.Column</code>	Allow customization of the cells in an entire column by modifying their alignment, background color, borders, and margins.
<code>JCFlowTable.Row</code> and <code>JCPageTable.Row</code>	Allow customization of the cells in an entire row by modifying their alignment, background color, borders, and margins.

5.6.1 Setting the Vertical Alignment

You can use the `JCFlowTable.Cell.setCellAlignment()` or `JCPageTable.Cell.setCellAlignment()` method to adjust the vertical alignment of text in a cell.

Adding the words “Retriever” and “Scottish” to cells 0,0 and 2,0 increases the height of all of the cells in rows 0 and 2. By default, the other cells in the same row align their text to the top of the cell.

Dogs	Cats	Horses
Labrador Retriever	Persian	Shetland
Collie	Siamese	Arabian
Scottish Terrier	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 27 Table cells with vertical alignment set to Top.

Vertically aligning text to middle of the cell

Suppose, for example, you want to vertically align that text to the middle of the cell. Here is the relevant code and how the altered table will look:

```
// for JFlowTable only
table.setDefaultCellAlignment(JFlowTable.CELL_ALIGNMENT_CENTER);
// for JPageTable only
table.setDefaultCellAlignment(JPageTable.CELL_ALIGNMENT_CENTER);
```

Dogs	Cats	Horses
Labrador Retriever	Persian	Shetland
Collie	Siamese	Arabian
Scottish Terrier	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 28 Table cells with vertical alignment set to Center.

Aligning individual cells

You may also control the alignment in cells *individually*.

The first three code snippets below show, using `JFlowTable`, how to align a cell to the bottom, top, and center of a cell, respectively. The fourth code snippet shows how to revert to the last alignment specified. Recall that with `JFlowTable`, rows are created individually and are printed/flowed as each one is finished.

Before adding text to the first row, call

```
cell = table.getCell(1);
cell.setCellAlignment(JFlowTable.CELL_ALIGNMENT_BOTTOM);
```

Before adding text to the first row, call

```
cell = table.getCell(2);
cell.setCellAlignment(JFlowTable.CELL_ALIGNMENT_TOP);
```

Before adding text to the third row, call

```
cell = table.getCell(2);
cell.setCellAlignment(JFlowTable.CELL_ALIGNMENT_CENTER);
```

Before adding text to the third row, call

```
cell = table.getCell(1);
cell.setCellAlignment(JFlowTable.CELL_ALIGNMENT_NONE);
```

The next three code snippets show, using `JPageTable`, how to align a cell to the bottom, top, and center of a cell, respectively. The fourth code snippet shows how to revert to the last alignment specified.

```
cell = table.getCell(0,1);
```

```

cell.setCellAlignment(JCPageTable.CELL_ALIGNMENT_BOTTOM);

cell = table.getCell(0,2);
cell.setCellAlignment(JCPageTable.CELL_ALIGNMENT_TOP);

cell = table.getCell(2,2);
cell.setCellAlignment(JCPageTable.CELL_ALIGNMENT_CENTER);

cell = table.getCell(2,1);
cell.setCellAlignment(JCPageTable.CELL_ALIGNMENT_NONE );

```

JCFlowTable.getCell() and JCPageTable.getCell() identify the cells containing text we want to vertically realign. JCFlowTable.Cell.setCellAlignment() and JCPageTable.Cell.setCellAlignment() adjust their vertical alignment in the cell, in this case, to the center of the cell. Our adjustments produce the following results:

Dogs	Cats	Horses
Labrador Retriever	Persian	Shetland
Collie	Siamese	Arabian
Scottish Terrier	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 29 Table cells with cell alignment set individually.

JCFlowTable and JCPageTable provide the following cell vertical alignment options:

CELL_ALIGNMENT_CENTER	Aligns cell text to the middle of the row.
CELL_ALIGNMENT_TOP	Aligns cell text to the top of the row.
CELL_ALIGNMENT_BOTTOM	Aligns cell text to the bottom of the row.
CELL_ALIGNMENT_NONE	No specific cell alignment.

5.6.2 Defining Cell Margins

Define cell margins in a cell before anything is flowed to that cell.

To control the space between the cell border and the text it contains, you adjust the cell margins. An enlarged version of one of the cells in the sample table provides a good example.

Siamese

Figure 30 Enlarged table cell.

The program has left gaps between the text and the left, right, top, and bottom borders of the cell. You can adjust the size of these gaps using the `JCFlowTable.Cell` or `JCPageTable.Cell` methods.

If you are using `JCFlowTable`, you must implement this code before adding text to the second row.

```
//JCFlowTable only
cell = table.getCell(1);
//JCPageTable only
cell = table.getCell(1,1);

cell.setBottomMargin(new JUnit.Measure(JUnit.POINTS, 8));
cell.setTopMargin(new JUnit.Measure(JUnit.POINTS, 8));
cell.setLeftMargin(new JUnit.Measure(JUnit.POINTS, 2));
cell.setRightMargin(new JUnit.Measure(JUnit.POINTS, 2));
```

For `JCFlowTable`, the preceding example identifies the cell column in the current row, while for `JCPageTable` identifies the cell (1,1 — remember that the header row is numbered separately), for which margins are to be adjusted, and then calls the appropriate `JCPageTable.Cell` methods to set the top and bottom margins to 8 points and the left and right margins to 2 points.

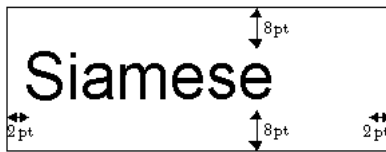


Figure 31 Enlarged table cell with margins illustrated.

5.6.3 Customizing Cell Borders

For `JCFlowTable`, customize cell borders in the current row before anything is flowed to that row.

For `JCPageTable`, customize cell borders before anything is flowed to the table.

You can customize individual cells by overriding the border style specified by the `JCDrawStyle` and applied by the appropriate `JCFlowTable` or `JCPageTable` method. To override the style for a particular cell's borders, identify the cell, create a new `JCDrawStyle`, and apply it using a method from the `JCFlowTable.Cell` or `JCPageTable.Cell` class.

For JCFLOWTable, here is sample code showing how to customize cell borders. Again, for JCFLOWTable, you must customize the cell borders before adding text to the second row.

```
1 JCDrawStyle cellstyle = (JCDrawStyle) JCDrawStyle.stringToStyle
   ("default line").clone();
2 cellstyle.setLineType(JCDrawStyle.LINE_TYPE_SINGLE);
3 cellstyle.setLineWidth(new JCUUnit.Measure(JCUUnit.POINTS, 2));
4 // Must customize cell borders before adding text to the second row
5 JCFLOWTable.Cell cell = table.getCell(1);
6 cell.setBottomBorderStyle(cellstyle);
7 cell.setRightBorderStyle(cellstyle);
8 cell.setTopBorderStyle(cellstyle);
9 cell.setLeftBorderStyle(cellstyle);
```

Once you have finished these lines of code, the results are shown in Figure 32.

Similar code can be used for JCPageTable, except that the fourth line should be removed, and the fifth line should be replaced with the following:

```
JCPageTable.Cell cell = table.getCell(1, 1);
```

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Scottish	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 32 Table cell with customized borders.

Drawing Borders and Precedence

Despite recording user choices for borders on all sides of a cell, JClass ServerReport tables draw only one border between each pair of cells. Borders are generally thought of as the bottom and right borders of a cell, except for the top-most and left-most borders of a table section.

If the bottom border of a cell has been set, this border is drawn between the selected cell and the corresponding cell in the row directly below. If the top border of a cell has been set, then that border is chosen for drawing as the bottom border of the corresponding cell in the row directly above. If neither the bottom border of a cell nor the top border of the cell directly below has been set, the value of the Border property for the corresponding Row object is used.

For example, to make the bottom border of the *Siamese* cell thicker, you can either set it directly:

```
JCDrawStyle cellstyle = (JCDrawStyle) JCDrawStyle.StringToStyle
    ("default line").clone();
cellstyle.setLineType(JCDrawStyle.LINE_TYPE_SINGLE);
cellstyle.setLineWidth(new JUnit.Measure(JUnit.POINT 2));
JCPageTable.Cell cell = table.getCell(1, 1);
cell.setBottomBorderStyle(cellstyle);
```

or you can set the top border of the *Calico* cell:

```
JCDrawStyle cellstyle = (JCDrawStyle) JCDrawStyle.StringToStyle
    ("default line").clone();
cellstyle.setLineType(JCDrawStyle.LINE_TYPE_SINGLE);
cellstyle.setLineWidth(new JUnit.Measure(JUnit.POINT 2));
JCPageTable.Cell cell = table.getCell(2, 1);
cell.setTopBorderStyle(cellstyle);
```

Both code examples will produce the following:

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Scottish	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 33 Setting the bottom border of the Siamese cell will produce the same output as setting the top border of the Calico cell.

Note: This may also be done using `JCFlowTable`.

If both the bottom border of a cell and the top border of the cell directly below have been set, precedence is determined via the `BottomBorderPrecedence` property of the `Cell` object. This property is set with the `setBottomBorderPrecedence()` method and retrieved with the `getBottomBorderPrecedence()` method. Possible values are:

- `AbstractPageTable.THIS_CELL`: the bottom border of the current cell will be chosen.
- `AbstractPageTable.NEXT_CELL`: the top border of the corresponding cell in the next row will be chosen.

For example, the following code will produce the table in Figure 34:

```
JCDrawStyle cellstylethick = (JCDrawStyle) JCDrawStyle.StringToStyle
    ("default line").clone();
cellstylethick.setLineType(JCDrawStyle.LINE_TYPE_SINGLE);
cellstylethick.setLineWidth(new JUnit.Measure(JUnit.POINT 2));
JCDrawStyle cellstyledash = (JCDrawStyle) JCDrawStyle.StringToStyle
    ("default line").clone();
cellstyledash.setLineType(JCDrawStyle.LINE_TYPE_DASHED);
JCPageTable.Cell cell1 = table.getCell(1, 1);
cell1.setBottomBorderStyle(cellstylethick);
JCPageTable.Cell cell2 = table.getCell(2, 1);
```

```
cell12.setTopBorderStyle(cellstyledash);
cell11.SetBottomBorderPrecedence(AbstractPageTable.THIS_CELL);
```

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Scottish	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 34 The table when BottomBorderPrecedence is set to AbstractPageTable.THIS_CELL.

while the same block of code will produce the table in Figure 35 if the final line is replace with:

```
cell11.SetBottomBorderPrecedence(AbstractPageTable.NEXT_CELL);
```

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Scottish	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 35 The table when BottomBorderPrecedence is set to AbstractPageTable.NEXT_CELL.

Left/right cell borders behave similarly. Their precedence is defined by the RightBorderPrecedence property of the Cell object, which is set with the setRightBorderPrecedence() method and retrieved with the getRightBorderPrecedence() method.

When a table is interrupted by a page break, the top border of the cell in the row following the page break is not taken into consideration when drawing the bottom border of the row directly before the page break. Similarly, the bottom border of the cell in the row before the page break is not taken into consideration when drawing the top border of the row following the page break.

For example, if the following code placed your table in the middle of a page break:

```
JCDrawStyle cellstyle = (JCDrawStyle) JCDrawStyle.StringToStyle
    ("default line").clone();
cellstyle.setLineType(JCDrawStyle.LINE_TYPE_SINGLE);
cellstyle.setLineWidth(new JUnit.Measure(JUnit.POINT 2));
JCPageTable.Cell cell = table.getCell(1, 1);
cell.setBottomBorderStyle(cellstyle);
JCPageTable.Cell cell = table.getCell(2, 2);
cell.setTopBorderStyle(cellstyle);
```

the following will result:

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabic

page 1

Scottish	Galico	Clydesdale
Seller	Tabby	Pelermio

Figure 36 Cell border behavior when the table is divided by a page break.

5.6.4 Spanning Cells

Spanning cells is the process by which the borders between specified cells are eliminated, creating one merged cell. The merged cell can cross multiple row and column borders, but does not alter the structure of the remaining cells, rows or columns in the table. Note that you should span cells before populating the cells.

JCFlowTable allows column spanning but not row spanning; JCPageTable allows both.

To span cells in a column using JCFlowTable, pass the column number of the cell that is the left-hand side of the span to the spanCells() method, along with the number of columns to span to the right. Follow this format:

```
table.spanCells(startColumn, numColumns);
```

For example, to span three cells across in the current row, call

```
table.spanCells(0, 3);
```

Here is the result:

Dogs	Cats	Horses
Labrador	Persian	Shetland
Collie	Siamese	Arabian
Scottish	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 37 Table with spanned cells.

To span cells using JCPageTable, pass the row and column number of the cell that is the upper left-hand corner of the span, along with the number of rows to span down and the number of columns to span to the right, to the JCPageTable.spanCells() method. Follow this format:

```
table.spanCells(startRow, startColumn, numRows, numColumns);
```

For example, to span four cells down the first column of a table, beginning with the upper left-hand corner cell in the table, enter:

```
table.spanCells(0, 0, 4, 1);
```

Dogs	Cats	Horses
Labrador	Persian	Shetland
Retriever	Siamese	Arabian
Collie	Calico	Clydesdale
Scottish		
Terrier		
Setter	Tabby	Palomino

Figure 38 Table with spanned cells.

5.7 Table Wrapping

If a table is too long for the current frame, the extra rows flow to the next frame, along with the header row, which appears at the top of every frame to which the table rows flow.

Table column wrapping is allowed only in `JCPageTable`. `JCFlowTable` clips columns to the width of the frame.

If `JCPageTable` is too wide for the current frame, use `JCPageTable.setOverflowMode()` to handle the extra column data.

To wrap the extra column(s) so that they appear below the table:

```
table.setOverflowMode(JCPageTable.OVERFLOW_WRAP_COLUMNS);
```

Dogs	Cats
Labrador Retriever	Persian
Collie	Siamese
Scottish Terrier	Calico
Setter	Tabby

Horses
Shetland
Arabian
Clydesdale
Palomino

Figure 39 Table with wrapped column.

To wrap the extra column(s) onto the following page:

```
table.setOverflowMode(JCPageTable.OVERFLOW_WRAP_COLUMNS_NEXT);
```

Dogs	Cats
Labrador Retriever	Persian
Collie	Siamese
Scottish Terrier	Calico
Setter	Tabby

Horses
Shetland
Arabian
Clydesdale
Palomino

Figure 40 Table with column wrapped to the next page.

To clip an equal amount of data from the left and right-most columns in the table:

```
table.setOverflowMode(JCPageTable.OVERFLOW_CLIP_COLUMNS);
```

Dogs	Cats	Horse
Labrador Retriever	Persian	Shetland
Collie	Siamese	Arabian
Scottish Terrier	Calico	Clydesdale
Setter	Tabby	Palomino

Figure 41 Table with clipped columns.

5.8 Converting Tables

You can use `JClass ServerReport` to convert data from other Java table classes into `JCFlowTable` and `JCPageTable` tables. You can extract table data along with simple text formatting from `JClass JCTables` and `Swing JTables`. You can also extract the data from a `JDBC` result set and reconstruct it as a `JCFlowTable` or `JCPageTable`.

5.8.1 Converting JClass LiveTables

If you purchased JClass LiveTable (available as part of JClass DesktopViews), you can flow data from a JClass LiveTable JCTable into a JCFlowTable or JCPageTable in your JClass ServerReport application.

Creating a New JCFlowTable or JCPageTable

To flow JCTable data into a new instance of a JCFlowTable or JCPageTable, use JCFlowTableFromJCTable.createTable() or JCPageTableFromJCTable.createTable(). You have the following options:

Method	Description
<code>createTable(JCDocument doc, JCTable jcTable, boolean populate)</code>	If populate is true, creates a JCFlowTable or JCPageTable by duplicating all of the data and text formatting stored in the JCTable data source. If populate is False, an empty JCFlowTable or JCPageTable is created.
<code>createTable(JCDocument doc, TableDataModel tableDataModel, boolean populate)</code>	If populate is true, creates a JCFlowTable or JCPageTable by duplicating the data stored in the JCTable data source. No text formatting is converted. If populate is False, and empty JCFlowTable or JCPageTable is created.

Populating an Existing JCFlowTable or JCPageTable

To flow JCTable data into an existing JCFlowTable or JCPageTable, use JCFlowTableFromJCTable.populateTable() or JCPageTableFromJCTable.populateTable(). You have the following options:

Method	Description
JCFlowTable: <code>populateTable(JCFlowTable table, JCTable jcTable)</code> JCPageTable: <code>populateTable(JCPageTable table, JCTable jcTable)</code>	Populates a JCFlowTable or JCPageTable with all of the data and text formatting stored in the JCTable data source.

Method	Description
JCFlowTable: populateTable(JCFlowTable table, TableDataModel tableDataModel) JCPageTable: populateTable(JCPageTable table, TableDataModel tableDataModel)	Populates a JCFlowTable or JCPageTable with the data stored in the JTable data source. No text formatting is converted.

5.8.2 Converting Swing JTables

You can also convert tables created with `javax.swing.JTable` into `JCFlowTable` or `JCPageTable` instances.

Creating a New JCFlowTable or JCPageTable

To create a new `JCFlowTable` or `JCPageTable` from an existing Swing `JTable` or `JTableTableModel`, use the `JCFlowTableFromJTable.createTable()` or `JCPageTableFromJTable.createTable()` methods. You have the following options:

Method	Description
createTable(JCDocument doc, javax.swing.JTable jTable, boolean populate)	If populate is True, creates a JCFlowTable or JCPageTable by duplicating all of the data, cell fonts, and text alignments from the view of the source JTable. If populate is False, an empty JCFlowTable or JCPageTable, with the same number of columns as the view of the source JTable, is created.
createTable(JCDocument doc, javax.swing.table.TableModel tableModel, boolean populate)	If populate is True, creates a JCFlowTable or JCPageTable by duplicating all of the data from the source TableModel. No text formatting is performed. If populate is False, an empty JCFlowTable or JCPageTable, with the same number of columns as the source TableModel, is created.

Populating an Existing JFlowTable or JPageTable

To flow data and text styles from an existing Swing JTable or JTable TableModel into an existing JFlowTable or JPageTable, use the JFlowTableFromJTable.populateTable() or JPageTableFromJTable.populateTable() methods. You have the following options:

Method	Description
JFlowTable: populateTable(JFlowTable table, javax.swing.JTable jTable, boolean applyJTableStyles) JPageTable: populateTable(JPageTable table, javax.swing.JTable jTable, boolean applyJTableStyles)	Populates an existing JFlowTable or JPageTable with all of the data from the view of the source JTable. If applyJTableStyles is true, cell fonts and text alignments from the source JTable will be copied to the JFlowTable or JPageTable.
JFlowTable: populateTable(JFlowTable table, javax.swing.table.TableModel tableModel) JPageTable: populateTable(JPageTable table, javax.swing.table.TableModel tableModel)	Populates a JFlowTable or JPageTable with all of the data from the source TableModel. No text formatting is performed.

Examples

To perform a complete conversion of an existing JTable to a JClass ServerReport FlowTable:

```
JFlowTable flowTable =  
    JFlowTableFromJTable.create(document, jTable, true);
```

To convert an existing JTable to a JClass ServerReport FlowTable, without preserving text formatting from the existing JTable so that new formatting can be applied:

```
// create JFlowTable with same number of columns as JTable  
JFlowTable flowTable =  
    JFlowTableFromJTable.create(document, jTable, false);  
// apply our own text formatting to headers and body  
JPageTable headerTable = flowTable.getHeaders();  
headerTable.getRow(0).setDefaultStyle(JCTextStyle.Heading5);  
flowTable.setDefaultStyle(JCTextStyle.Code);  
// populate JFlowTable with data from JTable  
JFlowTableFromJTable.populate(flowTable, jTable, false);
```

5.8.3 Converting JDBC Databases

JDBC (Java DataBase Connectivity) is the part of the Java API (java.sql) that allows you to send SQL queries to a database. You can format the result set of an SQL query into a JFlowTable or JPageTable; please follow these steps.

1. Create the JFlowTable or JPageTable and the ResultSet.

For JFlowTable:

```

    public JCFlowTable createTable(JCDocument doc) {
        ResultSet resultSet = null;
        JCFlowTable table = null;

```

For JCPageTable:

```

    public JCPageTable createTable(JCDocument doc) {
        ResultSet resultSet = null;
        JCPageTable table = null;

```

2. Load the JDBC driver.

```

try {
    Class.forName(driver); // Example: sun.jdbc.odbc.JdbcOdbcDriver
} catch (ClassNotFoundException cnfe) {
    cnfe.printStackTrace();
}

```

3. Conduct the SQL query and return the JCFlowTable or JCPageTable.

```

try {
    Connection connection = DriverManager.getConnection(url,
        login, password);
    Statement statement = connection.createStatement();
    resultSet = statement.executeQuery(query);

    // these next two lines for JCFlowTable only
    table = com.klg.jclass.sreport.JCFlowTableFromJDBC.
        createTable(doc, resultSet, true);
    // these next two lines for JCPageTable only
    table = com.klg.jclass.sreport.JCPageTableFromJDBC.
        createTable(doc, resultSet, true);

    // clean up
    resultSet.close();
    statement.close();
    connection.close();

} catch (SQLException sqle) {
    JOptionPane.showMessageDialog(null, sqle.toString(), "SQL error",
        JOptionPane.ERROR_MESSAGE);
    return null;
}

// return the JCFlowTable or JCPageTable
return table;

```

5.9 Using Tables With XML

5.9.1 Tags Supported in Cells

Please note that only certain tags are supported between cell beginning and cell ending tags. Tags that are supported in cells are:

- `bold`
- `current-text-style`
- `embed-image`
- `external-java-code`
- `float-image`
- `horizontal-rule`
- `italic`
- `new-line`
- `new-paragraph`
- `paragraph`
- `space`
- `use-text-style`

Regarding `current-text-style` and `use-text-style`, when either of these tags is placed outside of a table, neither has any effect on the text inside the table. If `use-text-style` is used within a cell, it is used only for the text that it contains and cannot span cell boundaries. If `current-text-style` is used within a cell, it will set the default text style to be used for the remainder of the table, although it will be overridden by any use of the `use-text-style` tag or the table cell's own `text-style-name` attribute.

5.9.2 Setting Borders on Tables

You can set border properties in one of two ways:

- via the table style specified by the `table-style-name` attribute
- via `flow-table` and `page-table` border attributes

If you set `table-style-name`, you do not need to set the border attributes specifically. Alternately, you may set the border attributes via the `flow-table` and `page-table` tags.

Flow-table and page-table Tags – Setting Border Attributes

The `flow-table` and `page-table` tags have the following attributes for setting the borders which can be used in place of setting `table-style-name`:

- `border-style-name` – all borders in the table
- `horizontal-border-style-name` – all horizontal borders in the table
- `vertical-border-style-name` – all vertical borders in the table
- `edge-border-style-name` – the borders on the outside of the table
- `internal-border-style-name` – the borders on the inside of the table
- `top-border-style-name` – the top border of the table
- `bottom-border-style-name` – the bottom border of the table

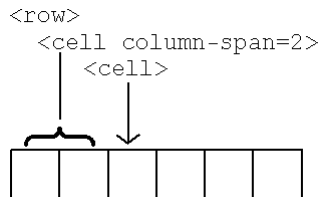
- `left-border-style-name` – the left border of the table
- `right-border-style-name` – the right border of the table
- `header-border-style-name` – the dividing border between the header and the rest of the table

The attributes are applied in the order listed above, from the property that covers the most separate borders to the one that covers the fewest separate borders. This means that if you were to set `border-style-name`, `horizontal-border-style-name`, and `vertical-style-name`, you would not see the setting for `border-style-name` because it would be covered by the settings for `horizontal-border-style-name`, and `vertical-style-name`, which are applied later.

The [API](#) (Javadoc) – which is automatically installed in `JCLASS_SERVER_HOME/docs/api/` – provides further details.

5.9.3 Spanning Cells and Rows in XML

With JClass ServerReport’s XML functionality, in a spanned cell definition, the next `<cell>` tag refers to the cell after the spanned cell, meaning that you are not able to access this “hidden” (spanned) cell.



If you use API methods, however, you can access cells made hidden via spanning cells, although the data in those cells will be obscured by the span.

The same situation applies to spanning rows, that is, if you use API methods, you can access rows that are hidden via spanning rows. If you use XML, you will not be able to access rows that are hidden via spanning cells.

Please note that while `flow-table` can span only columns, `page-table` and `header-table` can span rows as well as columns.

Adding Content in Other Ways

Adding Content Using Subframes ■ *Adding Content Using XML Fragments*

You may find that you need some flexibility in how you add content to your document. For example, you may have text that you want to treat as a unit, such as a case study within a research document, that needs to be rendered in a separate sub-frame within the current frame. Or maybe you want to be able to accept fragments of XML from another source and insert them directly into a document. JClass ServerReport offers you this flexibility with subframes and XML fragment handling.

6.1 Adding Content Using Subframes

Restrictions: This feature is not supported in RTF output. In addition, there are no corresponding XML elements in the JClass ServerReport DTD to embed or float subframes; you need to define subframes programmatically.

You can place frames within another frame, such as the header, the body frame, or any other frame on the page template. For example, you may want to add a few frames to a line of text in the body frame and have the frames wrap with the text. For clarity, we refer to this use of a frame as a subframe, but the subframe is just a basic `JCFrame` object and could be used like any other frame.

6.1.1 Defining a Subframe and its Content

To define a subframe, you create a `JCFrame` object exactly as you would for a frame that is placed on the page. Be sure to specify the document as a parameter in the `JCFrame` constructor. You then specify the contents of the subframe and place the subframe into the desired frame. The content of the subframe can be any type of content, including text, tables, images, and components.

Note: Subframes are static frames, not `flowFrames`. If the content does not fit in the subframe, it is truncated and `endOfFrameExceptions` are thrown.

The following example (modified from the *InnerFrame.java* example in the `JCLASS_SERVER_HOME/examples/sreport/main/` directory) defines a 1" x 1" subframe with a

red border, specifies text for the contents, and calls `frame.complete()`. The programmer then gets the current frame and embeds the subframe in its flow using the `embedFrame()` method.

```
flow.print("This example demonstrates the use of internal frames. ");
// Creates and embeds five frames in the text flow
for (int i = 0; i < 5; i++) {
    // Define the frame
    JFrame frame = new JFrame("NewFrame" + i, document);
    frame.setBorder(redLine);
    frame.setSize(new JUnit.Dimension(JUnit.INCHES, 1.0, 1.0));
    try {
        // Add content (note that "code" is a text style)
        frame.print(code, "Embedded Frame" + i);
        frame.complete();
        // Embed the frame
        flow.embedFrame(frame);
    }
    catch (EndOfFrameException e) {
        e.printStackTrace();
    }
}
```

When placing a frame within another frame, you can embed it (as above) or add it as a floating frame. The options are described in the following subsections.

6.1.2 Embedding a Subframe

When embedding a subframe (as shown in the preceding example), the subframe is flowed into the line of text. It is centered vertically on the current line and its size matches the size specified in the `JCFrame` object. The height of the current line is adjusted to accommodate the size of the subframe. The subframe will wrap with the contents of the line as necessary.

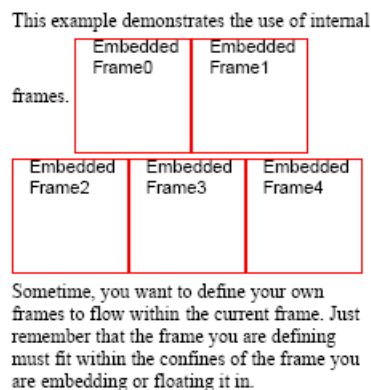


Figure 42 Embedded frames are placed within the line of text.

`JCFlow.embedFrame()` takes a `JCFrame` object. You can also choose to specify the vertical alignment (as an enumeration) and/or the size of the frame (as a `JCUnit.Dimension` object). Behind the scenes, `JCFlow.embedFrame()` calls `JCFrame.embedFrame()`. While you can use `JCFrame.embedFrame()` directly, it is preferable to use `JCFlow.embedFrame()` because it contains the logic necessary to move subframes to the next page and to handle overflows. For more information, look up `com.klg.jclass.sreport.JCFrame` and `com.klg.jclass.sreport.JCFlow` in the *JClass ServerViews API Documentation*.

The valid values for the vertical alignment parameter are:

- `JCDrawStyle.ALIGN_TO_TOP`
- `JCDrawStyle.ALIGN_TO_BOTTOM`
- `JCDrawStyle.POSITION_ON_BASELINE`
- `JCDrawStyle.POSITION_BELOW_BASELINE`
- `JCDrawStyle.CENTER_ON_BASELINE`
- `JCDrawStyle.CENTER_ABOVE_BASELINE`
- `JCDrawStyle.CENTER_IN_LINE`

When setting the size, ensure that the subframe will fit inside the desired frame. `JClass ServerReport` handles subframes that are too large for the target frame in the same way that it handles oversized images and components. For more information, see [Handling Items that Overflow the Frame](#), in Chapter 3.

6.1.3 Floating a Subframe

Floatable subframes can be added into the document flow as independent paragraph-level objects that do not break lines and allow normal flowed content to be added while the subframe is awaiting sufficient space. Subframes are placed in the order in which they are added.

When you define a new frame, it is useful to set the size in the constructor, although this does require you to set a location, the location will be ignored when floating

*This is a
floated Frame.*

Don't forget to call `frame.complete()` on your frame once you have finished printing/drawing to it.

Figure 43 Floatable frames are inserted between paragraphs when space permits.

`JCFlow.floatFrame()` takes a `JCFrame` object. You can also choose to specify the vertical alignment (as an enumeration) and/or the size of the frame (as a `JCUnit.Dimension` object). Behind the scenes, `JCFlow.floatFrame()` calls `JCFrame.floatFrame()`. While you can use `JCFrame.floatFrame()` directly, it is preferable to use `JCFlow.floatFrame()` because it contains the logic necessary to move subframes to the next page and to handle overflows. For more information, look up `com.klg.jclass.sreport.JCFlow` and `com.klg.jclass.sreport.JCFrame` in the *JClass ServerViews API Documentation*.

The valid values for the vertical alignment parameter are the same as for the `embedFrame()` method (see Section 6.1.2, [Embedding a Subframe](#)). When setting the size, ensure that the subframe will fit inside the desired frame. `JClass ServerReport` handles subframes that are too large for the target frame in the same way that it handles oversized images and components. For more information, see [Handling Items that Overflow the Frame](#), in Chapter 3.

Tip: It is preferable to set the size of the subframe in the `JCFrame` object, rather than in the method. If you set the `JCFrame location` parameter (as in the following example), it is ignored for floating frames.

The following example from *InnerFrame.java* creates a subframe called `frameFlow` and places it into the flow using the `floatFrame()` method. See Figure 43.

```
JCFrame frameFlow = new JCFrame("Biff", document,
                                new JCUnit.Point(1.0, 1.0),
                                new JCUnit.Dimension(1.0, 1.0));
frameFlow.setBorder(redLine);
try {
    frameFlow.print(italic, "This is a floated Frame.");
    frameFlow.complete();
}
catch (EndOfFrameException e) {
    e.printStackTrace();
}
try {
    flow.floatFrame(frameFlow);
}
catch (ObjectWillNotFitException e) {
    e.printStackTrace();
}
```

6.2 Adding Content Using XML Fragments

Using XML fragments, you can add almost any content to the flow that you would be able to add using XML. You cannot define styles within a fragment, however you can use styles that are defined outside of the fragment if a flow element takes a style as an attribute.

6.2.1 Defining and Using an XML Fragment

You define an XML fragment by creating a String object. Within the String, you add the XML elements, such as `<paragraph>`, and the text or other content that you want to appear in your document.

You then add the fragment to the document using the `renderXMLFragment()` method from either `JCFrame` or `JCFlow`, whichever is suitable for the content of the fragment. The `JCFrame.renderXMLFragment()` method takes a text style (`com.klg.jclass.sreport.JCTextStyle`) and an XML fragment (String). The `JCFlow.renderXMLFragment()` method takes only an XML fragment (String).

Example of a Frame-based XML Fragment

The following example (taken from the *XMLFragment.java* example in the *JCLASS_SERVER_HOME/examples/sreport/main/* directory) defines some text to flow into the frame. The programmer uses the `renderXMLFragment()` method from `JCFrame` to flow the text into the document.

```
//Create an XML fragment
String fragment = "<paragraph>XML Fragment Input</paragraph>\n" +
    "    <paragraph>Sometimes we may wish to just add some of the \n" +
    "content of a document through XML. We can do this \n" +
    "using the renderXMLFragment methods \n" +
    "of frame or flow.</paragraph>\n" +
    "    <paragraph>In the flow.renderXMLFragment method, \n" +
    "we can use the tag for any object which can be " +
    "added to a flow object. </paragraph>\n" +
    "    <paragraph>The same goes for frame \n" +
    "but we do need to be careful that what we are adding does \n" +
    "really belong in the current spot.\n" +
    "    </paragraph>\n" +
    "    <paragraph>For example, the following flow-table cannot be\n" +
    "added using the frame.renderXMLFragment method: </paragraph>\n";

...
//Render the xml fragment
try {
    // Note that "normal" is a text style object that is defined elsewhere
    flow.getCurrentFrame().renderXMLFragment(normal, fragment);
}
catch (IOException e) {
    e.printStackTrace();
}
catch (ParserConfigurationException e) {
    e.printStackTrace();
}
catch (SAXException e) {
    e.printStackTrace();
}
```

XML Fragment Input

Sometimes we may wish to just add some of the content of a document through XML. We can do this using the `renderXMLFragment` methods of `frame` or `flow`.

In the `flow.renderXMLFragment` method, we can use the tag for any object which can be added to a flow object.

The same goes for `frame` but we do need to be careful that what we are adding does really belong in the current spot. For example, the following flow-table cannot be added using the `frame.renderXMLFragment` method:

Figure 44 Text output created by `XMLFragment.class`

Example of a Flow-based XML Fragment

The following example (taken from the `XMLFragment.java` example in the `JCLASS_SERVER_HOME/examples/sreport/main/` directory) defines a flow table. Note that the programmer is able to assign a predefined style called “tacky” to the top border. This is because she defined the style programmatically (see `XMLFragment.java`) and the `header-table` element takes the `top-border-style-name` as an attribute. This example uses the `renderXMLFragment()` method from `JCFlow` to flow the table and text into the document.

```
//Create an XML fragment
String flowFragment =
    "<flow-table num-columns=\"3\">\n" +
    "<header-table top-border-style-name=\"tacky\">\n" +
    "<row>\n" +
    "  <cell>Column 1</cell>\n" +
    "  <cell>Column 2</cell>\n" +
    "  <cell>Column 3</cell>\n" +
    "</row>\n" +
    "</header-table> \n" +
    "<row>\n" +
    "  <cell>\n" +
    "    <italic>Just to demonstrate\n" +
    "    </italic>, but there is a plain font, the opening phrase\n" +
    "    in this cell just didn't use it.\n" +
    "  </cell>\n" +
    "  <cell>This is the data in another cell.\n" +
    "    We are using this data to demonstrate what a flow table \n" +
    "    looks like when it has been added via the renderXMLFragment\n" +
    "    method. It really doesn't matter what the data is.\n" +
    "  </cell>\n" +
    "  <cell>$16.95\n" +
    "  </cell>\n" +
    "</row>\n" +
    "<row border-style-name=\"restrained\">\n" +
    "  <cell column-span=\"2\">Here we are.\n" +
    "    <new-line />Another bunch of cells in another row.\n" +
    "  </cell>\n"
```



```

        "    <cell>$29.95\n" +
        "    </cell>\n" +
        "</row>\n" +
        "</flow-table>\n" +
        "...
// Render the xml fragment
try {
    flow.renderXMLFragment(flowFragment);
}
catch (IOException e) {
    e.printStackTrace();
}
catch (ParserConfigurationException e) {
    e.printStackTrace();
}
catch (SAXException e) {
    e.printStackTrace();
}

```

Column 1	Column 2	Column 3
<i>Just to demonstrate</i> , but there is a plain font, the opening phrase in this cell just didn't use it.	This is the data in another cell. We are using this data to demonstrate what a flow table looks like when it has been added through the renderXMLFragment method. It really doesn't matter what the data is.	\$16.95
Here we are. Another bunch of cells in another row.		\$29.95

Figure 45 Table output created by XMLFragment.class

6.2.2 List of Supported XML Elements for XML Fragments

The supported elements are found in *JCServerReport.dtd*. The elements contained in the other DTDs are not supported for this purpose. This means that you need to define and set things like text styles, draw styles, and table styles programmatically before rendering the fragment.

Within the *JCServerReport.dtd*, only the elements that can be used in the flow are supported. These elements are:

alt-flow	float-media-clip	new-page
bold	flow-table	new-paragraph
cell	header-table	new-section
chart-data	horizontal-rule	page-table
column-info	hyperlink	paragraph
contents-list-entry	italic	row
current-text-style	list	section
embed-chart	list-item	space
embed-image	macro	tab
embed-media-clip	mark-location	unsupported-operation
float-chart	new-column	use-text-style
float-image	new-line	

Defining Page Templates

XML-Based Page Templates ■ *The JCPage Object* ■ *The JCFrame Object*
The Interactions of JCFlow and JCFrame Objects ■ *A Sample Template*
Page Template Techniques ■ *Rotating Frame Contents*
Adding a Watermark ■ *Applying Page Templates*

Before you can flow content into a document, you need a page template that defines how the page is laid out. Page templates specify:

- the physical size of the page
- the location and size of the frames that will hold text and images
- the order text is to progress through those frames
- the next page to generate when the existing page and/or section is full

Most applications require more than one type of output page. If your application requires custom output pages, the template must provide a definition for each type of page.

7.1 Page Template Formats

JClass ServerReport allows you to select between two different types of page templates:

- **Simple:** A simple page template allows you to easily define a basic page, including header, footer, body frame, gutters, and margins. Control over the layout is limited; most frame position and size information is inferred from other frames on the same page or from default values. You can not add extra frames. This format can be applied to PDF or RTF documents, and corresponds to the `<simple-page>` tag. This is the default.
- **Full:** A full page template allows you to fully specify the layout of a page, and corresponds to the `<page>` tag. This format is used strictly for PDFs.

Please note that both page template formats must be defined within a `<page-template>` element. The `<page-template>` tag has only a `title` attribute, which is optional and is used to name the page template.

7.1.1 Simple Page Template Tags

As previously mentioned, the simple page template allows only header, footer, and body frames, as well as margins and gutters.

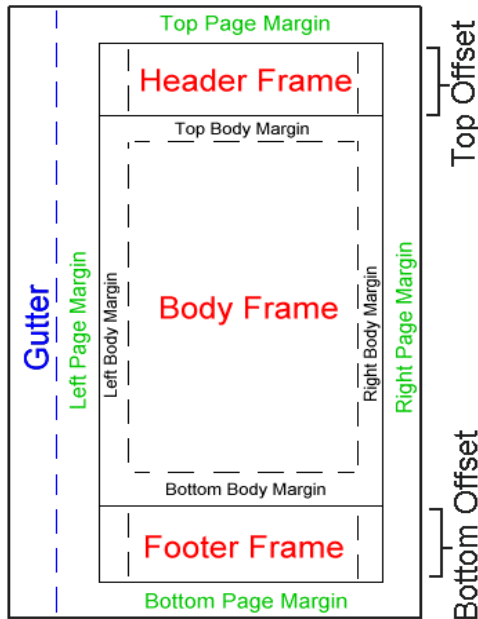


Figure 46 A Simple Page template.

The space defined by the top offset determines the height of the header frame; likewise, the space defined by the bottom offset determines the height of the footer frame.

A gutter is a vertical band of space that is left empty. This band appears on the left-hand side of a page, between the left most side of the page and the left-hand page margin. Please note that gutters will behave differently, depending on whether or not facing pages is enabled. See [Facing Pages](#) for more information.

The following table lists the simple page attributes:

Element	Description and Attributes
<simple-page>*	<p>name: Required. The name of this page template, referenced by other page definitions using the <flow-page> tag.</p> <p>unit: The unit of measurement used to plot out this page. Choose from inches, points, cm, cms, centimetres, and centimeters. The default is inches.</p> <p>orientation: Choose from automatic, portrait, and landscape.</p> <p>first: A Boolean attribute that indicates whether or not this page template is used for the first page in the document. The default is false.</p>
<size>*	<p>width: Required. Specifies the width of the page, measured in the units defined by the unit attribute of the <simple-page> tag.</p> <p>height: Required. Specifies the height of the page, measured in the units defined by the unit attribute of the <simple-page> tag.</p>
<margin>	<p>bottom: Optional. Specifies the bottom width of the margin.</p> <p>top: Optional. Specifies the top width of the margin, measured in the units defined by the unit attribute of the <simple-page> tag.</p> <p>left: Optional. Specifies the left width of the margin, measured in the units defined by the unit attribute of the <simple-page> tag.</p> <p>right: Optional. Specifies the right height of the margin, measured in the units defined by the unit attribute of the <simple-page> tag.</p>
<first-page-different>	<p>Optional. Specifies that the first page in the section will have different formatting than the other pages in the section.</p> <p>Currently, the different formatting refers to headers and footers. For more information, see Accessing Header and Footer Frames.</p>
<gutter>	<p>Optional. Indicates the existence of a gutter on the page.</p> <p>width: Required, if <gutter> tag exists. Specifies the width of the gutter, measured in the units defined by the unit attribute of the <simple-page> tag.</p> <p>Please note that gutters will behave differently, depending on whether or not facing pages is enabled. See Facing Pages for more information.</p>
<header>	Indicates that a header exists for this page template.
<footer>	Indicates that a footer exists for this page template.
<body>*	<p>top-offset: Optional. Distance between the top page margin and the start of the body frame. Note that a default value will be provided if none is specified.</p> <p>bottom-offset: Optional. The distance between the bottom page margin and the bottom of the body frame. Note that a default value will be provided if none is specified.</p>

Element	Description and Attributes
<flow-section>	name: Required. The name of the simple page template that this template will change to on a section break. Each section of the document is defined by a single <simple-page> tag.
<border>	type: Specifies the style used to draw a frame border. Choose from blank, broken, dashed, double, none, plain, regular, or single. The default is blank. color: Specifies the border color using either hexadecimal notation or a color from java.awt.Color. The default is black. thickness: Optional. Specifies the border width in pixels. The default is 0.1.
<column>	count: Required. Specifies the number of columns in the frame. spacing: Optional. Specifies the amount of space left between columns, measured in the units defined by the unit attribute of the <page> tag.
<watermark>	text: Required. Specifies the text of the watermark. style: Specifies the attributes to apply to the text.
* These tags are required.	

Facing Pages

Documents using the simple page template have the ability to change the appearance of the header, footer, and gutter, depending on whether the page is a right-hand or left-hand page. Use the JCDocument's `setFacingPages()` and `hasFacingPages()` methods to define this property.

When using facing pages, the gutter will flip from the left-hand side of a right-hand page to the right-hand side of a left-hand page; headers and footers are defined through the special header and footer frames (for more information, see [Accessing Header and Footer Frames](#)).

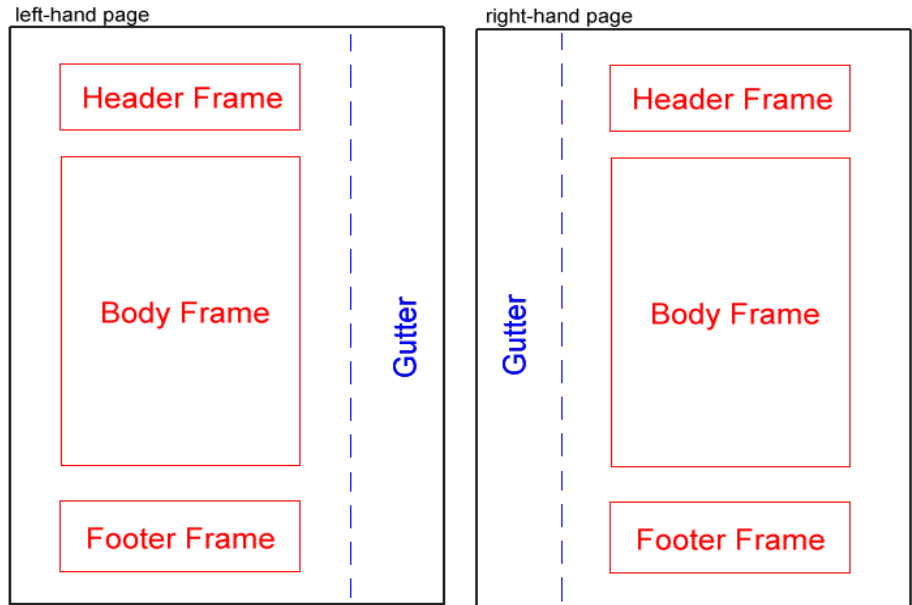


Figure 47 The appearance of the gutter for left and right-hand pages when using facing pages.

Accessing Header and Footer Frames

To access the actual header and footer frames that have been defined by your page template, either XML tags or JCPPage methods can be used.

Use the `<header-frame>` and `<footer-frame>` tags of the *JCServerReport.dtd* at document creation time (see section [JClass ServerReport DTD Tags](#), in Chapter 14). Although a page template contains only a single header and footer definition, multiple header and footer frames may have been created in the document depending on whether you are using the JCDocument `FacingPages` attribute or the page template's `<first-page-different>` tag. Use the following attributes of the `<header-frame>` and `<footer-frame>` tags to ensure that you retrieve the frame that you desire:

- `name`, which is the name of the page template containing the header or footer frame definition.
- `type`, which specifies the header or footer frame to retrieve. Valid values depend on your settings of the JCDocument `FacingPages` attribute and the page template `<first-page-different>` tag.

The following table indicates valid values for this attribute:

Facing Pages	First-Page Different	Valid Values for the Type Attribute
On	On	■ first-page-in-section for the first page ■ facing-left-page for left page ■ facing-right-page for right page
Off	On	■ first-page-in-section for the first page ■ non-facing-page for all other pages
On	Off	■ facing-left-page or facing-right-page
Off	Off	■ non-facing-page

Use the `getHeaderFrame(type)` and `getFooterFrame(type)` `JCPage` methods to retrieve the information programatically. The `type` value can be `JCFrame.NON_FACING_PAGE` when facing page is off, `JCFrame.FACING_LEFT_PAGE` for the left page, `JCFrame.FACING_RIGHT_PAGE` for the right page, and `JCFrame.FIRST_PAGE_IN_SECTION` for the first page when the template specifies that the first page is different from the rest of the document.

7.1.2 Full Page Template Tags

The following table describes the elements and attributes used to define a full page template.

Element	Attributes	Child Elements
<page>	<p>name: Required. The name of this page type, referenced by other page definitions using the <flow-page> tag.</p> <p>unit: The unit of measurement used to plot out this page. Choose from inches, points, cm, cms, centimetres, and centimeters. The default is inches.</p> <p>color: Optional. Specifies a default background color for this page using hexadecimal notation or a color from <code>com.klg.jclass.util.swing.JCSwingTypeConverter</code>.</p> <p>orientation: Choose from automatic, portrait, and landscape.</p> <p>first: A Boolean attribute that indicates whether or not this page template is used for the first page in the document. The default is false.</p>	<location>, <size>, <watermark>?, <frame>+, <flow-frame>*, <flow-page>, <flow-section>

+ Required and repeatable. * Optional and repeatable. ? Optional and non-repeatable.

Element	Attributes	Child Elements
<frame>	<p>name: Required. The name of this frame type, referenced by other page definitions using the <flow-frame> tag.</p> <p>unit: The unit of measurement used to plot out this frame. Choose from inches, points, cm, cms, centimetres, and centimeters. The default is inches.</p> <p>color: Optional. Specifies a default background color for this frame using hexadecimal notation or a color from <code>com.klg.jclass.util.swing.JCSwingTypeConverter</code>.</p> <p>contents-rotation: Optional. Specifies the rotation of all elements in the frame (for example, text, hyperlinks, images, etc). Options are 0, 90, 180, and 270; default is 0.</p>	<location>, <size>, <border>?, <column>?
<margin>	<p>bottom: Optional. Specifies the bottom width of the margin.</p> <p>top: Optional. Specifies the top width of the margin.</p> <p>left: Optional. Specifies the left width of the margin.</p> <p>right: Optional. Specifies the right width of the margin.</p>	None.
<location>	<p>x: Required. Specifies the distance of the page or frame from the left-hand page edge.</p> <p>y: Required. Specifies the distance of the page or frame from the top of the page.</p>	None.
<size>	<p>width: Required. Specifies the width of the page or frame, measured in the units defined by the unit attribute of the <page> tag.</p> <p>height: Required. Specifies the height of the page or frame, measured in the units defined by the unit attribute of the <page> tag.</p>	None.
* Required and repeatable. * Optional and repeatable. ? Optional and non-repeatable.		

Element	Attributes	Child Elements
<border>	<p>type: Specifies the style used to draw a frame border. Choose from <code>blank</code>, <code>broken</code>, <code>dashed</code>, <code>double</code>, <code>none</code>, <code>plain</code>, <code>regular</code>, or <code>single</code>. The default is <code>blank</code>.</p> <p>color: Specifies the border color using either hexadecimal notation or a color from <code>java.awt.Color</code>. The default is <code>black</code>.</p> <p>thickness: Optional. Specifies the border width in pixels. The default is 0.1.</p>	None.
<column>	<p>count: Required. Specifies the number of columns in the frame.</p> <p>spacing: Optional. Specifies the amount of space left between columns, measured in the units defined by the unit attribute of the <page> tag.</p>	None.
<flow-frame>	<p>name: Required. Specifies the name of a frame to be added to the sequence of frames to which the document will flow content.</p>	None.
<flow-page>	<p>name: Required. Specifies the name of the page to which the flow is to progress when a new page is begun.</p>	None.
<flow-section>	<p>name: Required. Specifies the name of the page to which the flow is to progress when a new section is begun.</p>	None.
<watermark>	<p>text: Required. Specifies the text of the watermark.</p> <p>style: Specifies the attributes to apply to the text.</p>	<p><location>?, <transform>*</p>
⁺ Required and repeatable. [*] Optional and repeatable. [?] Optional and non-repeatable.		

Element	Attributes	Child Elements
<transform>	<p>type: Specifies the type of transformation to apply to the watermark. Choose from none, translate, scale, rotate, rotate-with-translate. The default is none.</p> <p>translate-x: Specifies the x value for a translate transformation.</p> <p>translate-y: Specifies the y value for a translate transformation.</p> <p>scale-x: Specifies the x value for a scale transformation.</p> <p>scale-y: Specifies the y value for a scale transformation.</p> <p>rotation-angle: Specifies the angle of rotation for a rotate transformation.</p>	None.

+ Required and repeatable. * Optional and repeatable. ? Optional and non-repeatable.

7.2 XML-Based Page Templates

JClass ServerReport templates are written in the Extensible Markup Language (XML). The templates use a common Document Type Definition (DTD) that is built into the JClass ServerReport API. The DTD defines the tags and attributes used to specify the appearance of the page templates. The JClass ServerReport DTD (*/com/kg/jclass/xml-dtd/ServerReportPageTemplate.dtd*) is in the JClass ServerReport JAR file.

The following table lists and describes the default page templates available to you.

Default Template	Description
Blank_8p5x11	Creates a blank (no headers or footers) page of standard US Letter size.
Blank_8p5x14	Creates a blank (no headers or footers) page of standard US Legal size.
Blank_11x17	Creates a blank (no headers or footers) page of standard Tabloid size.
Blank_A3	Creates a blank (no headers or footers) page of standard ISO A3 size.
Blank_A4	Creates a blank (no headers or footers) page of standard ISO A4 size.

Default Template	Description
Blank_A5	Creates a blank (no headers or footers) page of standard ISO A5 size.

Note: All of the default templates are supported in both PDF and RTF output, as they are simple page templates.

7.3 The JCPage Object

The `JCPage` methods and attributes specify how an actual page should be laid out, what frames should exist on a page, and the frame order of the flow. `JCPage` objects are based on XML templates.

For each different page that your document will need, a different `JCPage` object will need to be defined.

7.4 The JCFrame Object

Use `JCFrame` methods when you want to render content that is not part of the main flow. For example, the information contained in header and footer frames is rendered separately from the contents of the body frame. Until new instructions are given, information printed into the header frame of a template page will recur.

Content is not added directly into the `JCFrame`'s render list; rather, content is appended to a current line until a complete line is built or the current line is flushed. The current line always begins as zero height – no assumption is made about the attributes of the elements that will subsequently be added to it. Thus, the current line will always fit immediately following the just-completed previous line, even if there is virtually no space available at the end of a frame. As content is added to the current line, its size will be adjusted to fit the new elements. If an element is added to the line which makes the line too tall for the available space, then the frame will attempt a `newColumn()` action. In a case where the `newColumn()` action succeeds, the elements of the current line are mapped to their new positions and the flow continues. In the remainder of cases where there is no following column, the frame throws an `EndOfFrameException`. For more information on exceptions, please see [Exceptions](#), in Chapter 11.

JCFrame methods cease rendering information when the content reaches the bottom of a frame, since they are not part of the flow. When the frame runs out of room, an EndOfFrameException is thrown.

JCFrame Method	Description
<code>newColumn (JTextStyle style)</code>	Generates a column break and advances the text to the next column in the specified frame. Throws an <code>EndOfFrameException</code> in the last (only) column of a frame.
<code>newLine (JTextStyle style)</code>	Ends the current line and transfers the flow to a new line. Throws an <code>EndOfFrameException</code> if there is not enough room to print the text.
<code>print (JTextStyle style, String text)</code>	Renders the specified content to this frame. Throws an <code>EndOfFrameException</code> if there is not enough room to print the text.

To retrieve a full page template frame, call:

```
template.stringToFrame("framename");
```

where `template` is the `JCPage` object `template`. To retrieve a particular frame from a simple page template, call one of the following methods.

Method	Retrieved Frame
<code>template.getHeaderFrame()</code>	The static header frame.
<code>template.getFooterFrame()</code>	The static footer frame.
<code>JCFlow.getCurrentFrame()</code>	The current flow frame (which is normally the body).

For more information on getting headers and footers, please refer to Section 7.1.1, [Simple Page Template Tags](#).

7.5 The Interactions of JCFlow and JCFrame Objects

The following table describes the methods used to control flow.

JCFlow Method	Description
<code>print ()</code>	Renders the specified content to the flow.
<code>newLine()</code>	Ends the current line and begins a new line.
<code>newParagraph()</code>	Begins a new paragraph.

JCFlow Method	Description
<code>newColumn()</code>	Advances the text flow to the top of the next column, in the next frame if necessary.
<code>newFrame()</code>	Advances the text flow to the next frame, generating a new page if necessary.
<code>newPage()</code>	When using a full page template, creates a new page based on the current page's <code>flow-page</code> , and directs the flow to the first frame of the new page's <code>flow-frame</code> . When using a simple page template, begins a new page exactly like the current page.
<code>newSection()</code>	Creates a new section based on the current section's <code>flow-section</code> and directs the flow to the first frame of the new page's <code>flow-frame</code> .

In cases where text is to flow from frame to frame, the page template specifies the basic flow of text within a document. At times, you may want to advance the flow before it reaches the end of the current frame or page. For example, on a title page, you may want to flow the text containing the document title into a frame you've named *Title*, then advance the flow to print the author's name into another frame you've named *Author*.

Because you need to use `JCFrame` methods when you want to render text and images independently from the flow, these methods are extremely important to the text flow of your layout. When you want to direct the flow through frames and pages, use `JCFlow` methods.

<flow-page> and <flow-section>

`JClass ServerReport` prints data to a page. The page is simply the unit where frames are placed, allowing for data to be printed. The `<flow-page>` element controls the order in which data will flow, identifying which page is next in the series.

A section is a sub-division of the document as a whole. For example, if the document being produced is a book, each chapter would be its own section. The `<flow-section>` element allows you to define which page template to use when a section break is called from the current page.

7.6 A Sample Template

The following examples are XML documents you could use to define an 8.5 x 11 (US Letter) page template that, unlike the standard template `Blank_8p5x11`, includes headers and footers. Following are two examples, one that uses a full page template, and one that uses a simple page template.

Simple page template:

```

<?xml version="1.0"?>
<!DOCTYPE page-template SYSTEM "ServerReportPageTemplate.dtd">
  <page-template title="8p5x11">
    <simple-page name="8p5x11-page" unit="inches">
      <size width="8.5" height="11"/>
      <margin top="0.25" bottom="0.0" left="1.0" right="1.0"/>
      <header/>
      <footer/>
      <body top=offset="0.75" bottom=offset="0.75">
        <column count="2"/>
      </body>
      <flow-section name="8p5x11-page" />
    </simple-page>
  </page-template>

```

Full page template:

```

<?xml version="1.0"?>
<!DOCTYPE page-template SYSTEM "ServerReportPageTemplate.dtd">
  <page-template title="8p5x11">
    <page name="8p5x11" unit="inches">
      <location x="0" y="0"/>
      <size width="8.5" height="11"/>
      <frame name="header" unit="inches" color="grey">
        <location x="1" y="0.25"/>
        <size width="6.5" height="0.75"/>
      </frame>
      <frame name="body" unit="inches">
        <location x="1" y="1"/>
        <size width="6.5" height="9"/>
        <column count="2"/>
      </frame>
      <frame name="footer" unit="inches" color="pink">
        <location x="1" y="10.25"/>
        <size width="6.5" height="0.75"/>
      </frame>
      <flow-frame name="body"/>
      <flow-page name="8p5x11"/>
      <flow-section name="8p5x11"/>
    </page>
  </page-template>

```

The Template DTD

The structure of JClass ServerReport templates is defined in an XML Document Type Definition (DTD) found in *ServerReportPageTemplate.dtd*. We reprint it here to give you an idea of the hierarchy of elements in an XML template. For a complete description of the elements used in a JClass ServerReport template, refer to Section 7.1.2, [Full Page Template Tags](#). Please note that the order of the attributes and child elements of each element is fixed; the order shown in the example below must be used.

```

<!-- DTD for JClass ServerReport Page Templates -->

<!ELEMENT page-template (simple-page+|page+)>
  <!ATTLIST page-template title CDATA #IMPLIED>

<!ELEMENT simple-page (size,margin?,gutter?,first-page-different?,

```

```

        header?,footer?,body,flow-section?))
<!ATTLIST simple-page name CDATA #REQUIRED
        unit (inches|points|cm|cms|centimeters|centimetres) "inches"
        orientation (automatic|portrait|landscape) "automatic"
        first (True|true|TRUE|False|false|FALSE) "false">

<!ELEMENT size EMPTY>
  <!ATTLIST size width CDATA #REQUIRED
        height CDATA #REQUIRED>

<!ELEMENT gutter EMPTY>
  <!ATTLIST gutter width CDATA #REQUIRED >

<!ELEMENT header EMPTY>

<!ELEMENT footer EMPTY>

<!ELEMENT first-page-different EMPTY>

<!ELEMENT body (border?, column?, margin?))>
  <!ATTLIST body top-offset CDATA #IMPLIED
        bottom-offset CDATA #IMPLIED>

<!ELEMENT flow-section EMPTY>
  <!ATTLIST flow-section name CDATA #REQUIRED>

<!ELEMENT border EMPTY>
  <!ATTLIST border type
        (blank|broken|dashed|double|none|plain|regular|single)
        "blank" color CDATA "black" thickness CDATA #IMPLIED>

<!ELEMENT column EMPTY>
  <!ATTLIST column count CDATA #REQUIRED
        spacing CDATA #IMPLIED>

<!ELEMENT page (location,size,frame+,flow-frame*,flow-page,flow-section)>
  <!ATTLIST page name CDATA #REQUIRED
        unit (inches|points|cm|cms|centimeters|centimetres) "inches"
        color CDATA #IMPLIED
        orientation (automatic|portrait|landscape) "automatic"
        first (True|true|TRUE|False|false|FALSE) "false">

<!ELEMENT frame (location,size,border?,column?,margin?))>
  <!ATTLIST frame name CDATA #REQUIRED
        unit (inches|points|cm|cms|centimeters|centimetres) "inches"
        color CDATA #IMPLIED contents-rotation (0|90|180|270) "0">

<!ELEMENT location EMPTY>
  <!ATTLIST location x CDATA #REQUIRED y CDATA #REQUIRED>

<!ELEMENT margin EMPTY>
  <!ATTLIST margin top CDATA #REQUIRED right CDATA #REQUIRED
        left CDATA #REQUIRED bottom CDATA #REQUIRED>

<!ELEMENT flow-frame EMPTY>
  <!ATTLIST flow-frame name CDATA #REQUIRED>

```



```
<!ELEMENT flow-page EMPTY>
<!ATTLIST flow-page name CDATA #REQUIRED>
```

7.7 Page Template Techniques

Headers and Footers

You create headers and footers in the page template by defining frames not connected to the document's main flow.

Because header and footer frames are not connected to the main flow of the document, but are defined separately, any text or images you render to those frames in a page template are replicated on every page that is based on that template.

The following XML template lays out a standard 8.5x11 page using a simple page template consisting of a header frame, a body frame, and a footer frame.

```
<simple-page name="BookLeft" unit="inches">
  <size width="8.5" height="11.0"/>
  <margin top="0.5" bottom="0.5" left="0.5" right="0.5" />
  <header />
  <footer />
  <body top-offset="0.5" bottom-offset="0.5">
    <margin top="0.25" bottom="0.25" left="0.0" right="0.0" />
  </body>
  <flow-section name="BookLeft" />
</simple-page>
```

This template lays out the same 8.5x11 page using a full page template.

```
<page name="BookLeft" unit="inches">
  <location x="0" y="0"/>
  <size width="8.5" height="11.0"/>
  <frame name="header">
    <location x="0.5" y="0.5"/>
    <size width="7.5" height="0.5"/>
  </frame>
  <frame name="body">
    <location x="0.5" y="1.25"/>
    <size width="7.5" height="8.5"/>
  </frame>
  <frame name="footer">
    <location x="0.5" y="10"/>
    <size width="7.5" height="0.5"/>
  </frame>
  <flow-frame name="body"/>
  <flow-page name="BookRight"/>
  <flow-section name="BookChapter"/>
</page>
```

The following layout is defined by the previous examples.

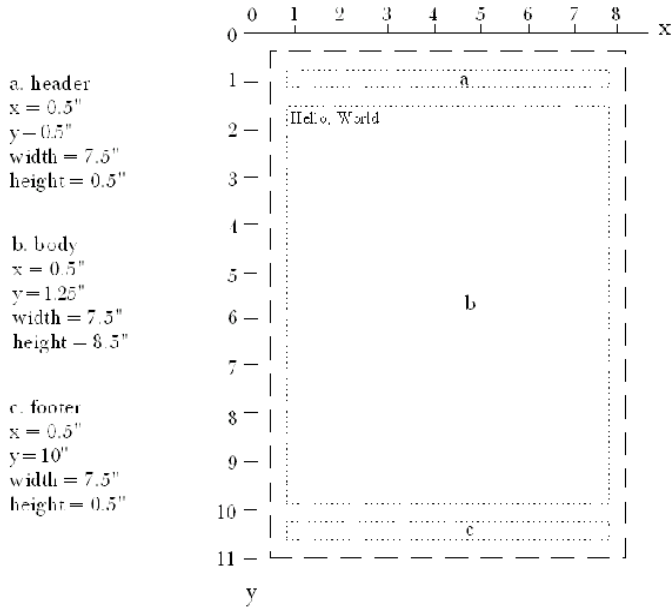


Figure 48 Header, Body, and Footer frames.

It is possible to layout several headers and footers behind the scene for simple page templates, depending on the values of the `JCDocument FacingPages` attribute and the `<first-page-different/>` tag. When `FacingPage` is true, left and right hand pages each have a separate header and footer; when a different first page has been specified, the first page will have a separate header and footer. To access these special headers and footers, either use the `<header-frame>` and `<footer-frame>` XML tags, or call `JCPage.getHeaderFrame()` or `JCPage.getFooterFrame()`.

Multiple Columns

To create multiple columns, add `column count` and `spacing` attributes to the definition of the body frame in the XML template. The following is a full page template example:

```
<frame name="body">
  <location x="0.5" y="1.25"/>
  <size width="7.5" height="8.5"/>
  <column count="2" spacing="0.5"/>
</frame>
```

The same can be generated using a simple page template, for example:

```
<body top-offset="0.5" bottom-offset="0.5">  
  <column count="2" spacing="0.5" />  
</body>
```

The `column count` and `spacing` parameters instruct JClass ServerReport to flow text through the body frame in two columns, separated by a gap of 0.5". Columns are always of equal width.

The changes to the template produce the following results.

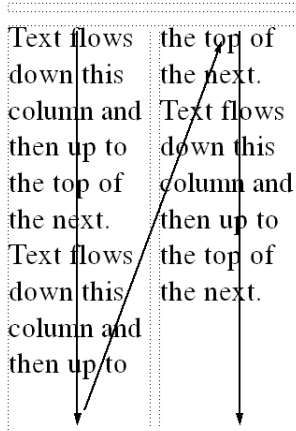


Figure 49 Text flowing through columns.

7.8 Rotating Frame Contents

Note: This feature is not supported in RTF output.

The contents of a frame can be rotated in 90 degree increments clockwise, with the help of the `<frame>` tag's `contents-rotation` attribute in page template XML, or `JCFrame`'s `setContentsRotation()` method in the API. Possible values are 0, 90, 180, and 270; 0 is the default value.

Note: The frame rotation must be specified before the document is created, in the frame definition for the page template.

When applying a rotation, only the frame's contents (for example, text, hyperlinks, images, tables, etc.) and columns are rotated; all of the other frame attributes (margins, height, and width) will behave normally, because the frame itself is not rotated.

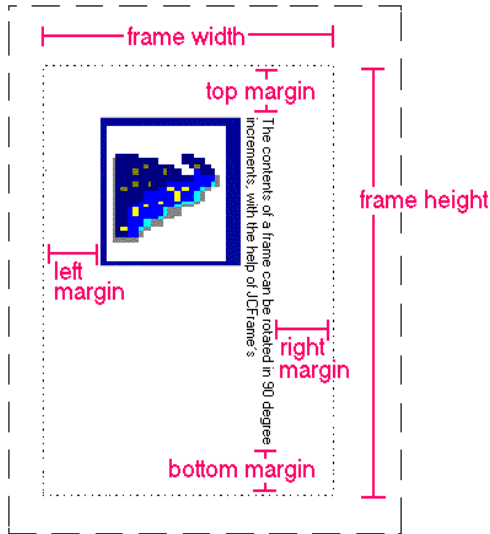


Figure 50 Frame attributes that do not rotate when frame contents are rotated.

For example:

```
<frame name="rotated-text" unit="inches" contents-rotation="90">
  <location x="1" y="1" />
  <size width="5" height="7" />
  <column count="2" />
  <margin top="0.5" left="0.1" right="0.1" bottom="0.5" />
</frame>
```

will produce the following:

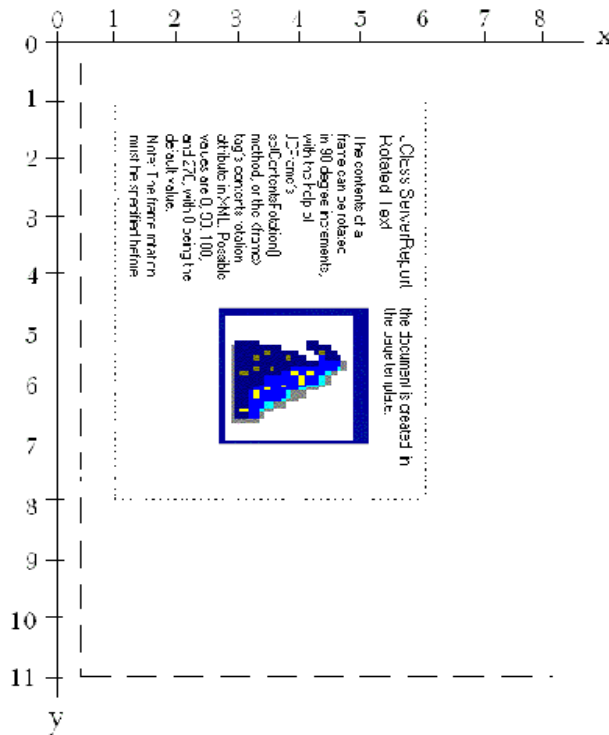


Figure 51 Rotated text in a ServerReport Frame.

7.9 Adding a Watermark

Note: This feature is not supported in RTF output.

You can add a text-based watermark to your page template. Watermarks are useful for identifying drafts or evaluation copies of a report. The watermark appears on each page that uses the template. The watermark is drawn after the page background, but before the frames are drawn.

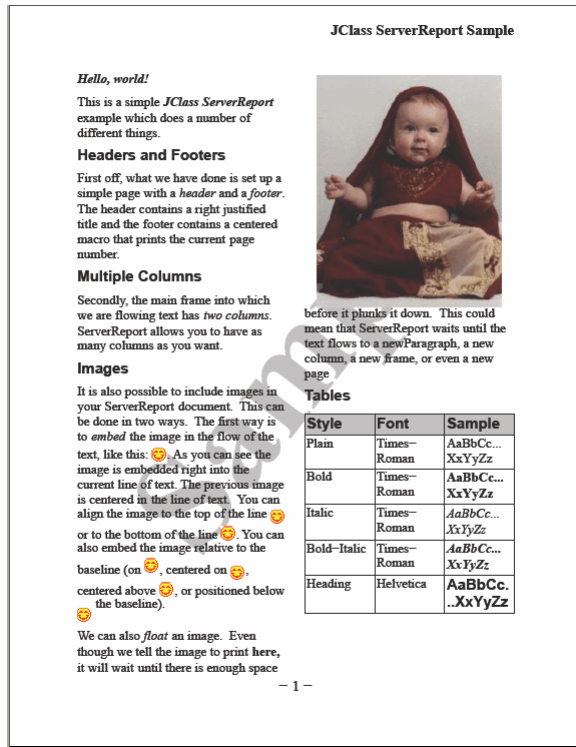


Figure 52 The Basic Sample Servlet example with a watermark

7.9.1 Creating the Watermark Text

To add a watermark, you set the `WatermarkText` property on the page template and specify the text to display.

```
// Specify the template and create the watermark text
pageTemplate = document.stringToTemplate("title");
pageTemplate.setWatermarkText("Draft");
```

7.9.2 Configuring the Text Properties of the Watermark

You can configure the text alignment, font style, and color by creating a `JCTextStyle` object and setting its color, alignment, and font properties. Other `JCTextStyle` properties are not supported for watermarks. Note that the valid values for the `Alignment` property are `JCTextStyle.ALIGNMENT_LEFT` (*default*), `JCTextStyle.ALIGNMENT_CENTER`, `JCTextStyle.ALIGNMENT_RIGHT`, and `JCTextStyle.ALIGNMENT_NONE` (used when you want to set the exact location of the watermark on the page). You then set the `JCTextStyle` object on the template using the `WatermarkStyle` property.

```
// Define the style elements for the watermark
JCTextStyle watermarkStyle = new JCTextStyle("watermark1", document);
watermarkStyle.setColor(Color.lightGray);
watermarkStyle.setAlignment(JCTextStyle.ALIGNMENT_CENTER);
watermarkStyle.setFont(createBoldFont(40));

// Specify the template and create the watermark text
pageTemplate = document.stringToTemplate("title");
pageTemplate.setWatermarkText("Draft");

// Apply the style to the watermark
pageTemplate.setWatermarkStyle(watermarkStyle);
```

7.9.3 Specifying the Location of the Watermark

By default, the watermark is drawn at 0,0, which is the top left corner of the page. If you want to specify where the watermark is located on the page, set the template's `WatermarkPoint` property and specify a `JCUnit.Point` object. `JCUnit.Point` takes an enumeration that specifies a unit of measure (such as `JCUnit.INCHES`) plus an x value and a y value.

When an alignment is also specified, the alignment overrides the x value of the `WatermarkPoint` property. To apply the x value, you need to set the `JCTextStyle` object's `Alignment` property to `JCTextStyle.ALIGNMENT_NONE`.

```
// Remove the alignment and specify an exact location
watermarkStyle.setAlignment(JCTextStyle.ALIGNMENT_NONE);
pageTemplate.setWatermarkPoint(new JCUnit.Point(JCUnit.INCHES, 9.0,
10.0));
```

7.9.4 Rotating, Scaling, or Translating the Watermark

If you want to rotate the watermark text or perform other transformations, you can provide a list of `JCTransform` objects using the template's `WatermarkTransformList` property. Each `JCTransform` object describes a simple affine transformation, such as a translation, rotation, rotation with translation, or scaling operation. These operations coincide with transformations defined by the `java.awt.geom.AffineTransform` class. Transformations are applied (in order) after the `WatermarkPoint` property and/or the alignment is applied.

To specify a transformation, create a `JCTransform` object and then use its methods to set the type of transformation. The following table summarizes the methods available. For details, look up `com.klg.jclass.sreport.JCTransform` in the *JClass ServerViews API Documentation*.

JCTransform Methods	Explanation
<code>setAsRotate(double rotationAngle)</code>	Rotates the text by the angle specified.
<code>setAsTranslate(JCUnit.Point translate)</code>	Translates the watermark text the distance specified in the supplied <code>JCUnit.Point</code> object.

JCTransform Methods	Explanation
setAsRotateWithTranslate(double rotationAngle, JUnit.Point translate)	Rotates and translates the watermark text using the specified values.
setAsScale(double scaleX, double scaleY)	Scales the watermark text using the specified x and y scale values.

The following code (from *JCLASS_SERVER_HOME/examples/sreport/SampleBase.java*) translates and then rotates the watermark.

```
// Create two transformations that translate and then rotate
// the watermark.
JCTransform transform = new JCTransform();
double translateX = 200.0;
double translateY = size.getHeight().getAs(JUnit.POINTS) - 200.0;
transform.setAsTranslate(new JUnit.Point(JUnit.POINTS, translateX,
                                         translateY));
templatePage.addToWatermarkTransformList(transform);
transform = new JCTransform();
transform.setAsRotate(-Math.atan2(11.0, 8.5));
templatePage.addToWatermarkTransformList(transform);
```

7.10 Applying Page Templates

After writing your own XML page template, you can apply it to your document by loading it either as an internal String or an external XML file. The following table describes the various JCPageTemplate methods you can use to apply templates.

Method	Explanation
importTemplates(JCDocument doc, File xmlfile)	Reads from java.io.File to import an XML template and apply it to the specified JCDocument.
importTemplates(JCDocument doc, Reader reader)	Reads from java.io.Reader to import an XML template and apply it to the specified JCDocument.
importTemplates(JCDocument doc, InputSource input)	Reads from org.xml.sax.InputSource to import an XML template and apply it to the specified JCDocument.
loadTemplates(File xmlfile)	Reads from java.io.File to load the XML template without applying it to a specific document.
loadTemplates(Reader reader)	Reads from java.io.Reader to load the XML template without applying it to a specific document.

Method	Explanation
<code>loadTemplates(InputStream input)</code>	Reads from <code>org.xml.sax.InputStream</code> to load the XML template without applying it to a specific document.

7.10.1 Loading External XML Files

Users of JClass ServerReport require a parser that is JAXP 1.1-compliant. For your convenience, JClass ServerReport provides both *jaxp.jar* and *crimson.jar* in the *JCLASS_SERVER_HOME/lib/* directory. You may substitute for *crimson.jar* any parser that is compliant with Sun's JAXP 1.1 specification. See [Sun's JAXP documentation](#) for more information.

The following example uses `java.io.File` to load the external XML file `8p5x11.xml` as a template.

```
JCDocument document = null;
try {
    document = new JCDocument(stream, JCPageTemplate.loadTemplates
        (new java.io.File("8p5x11.xml")));
}
catch (Exception e) {
    System.err.println("Error loading template = " + e);
    System.exit(1);
}
```

7.10.2 Loading XML Strings

The following example uses `java.io.StringReader` to load an XML template that is defined as a `String` (template) earlier in the program.

```
JCDocument document = null;
try {
    document = new JCDocument(stream, JCPageTemplate.loadTemplates
        (new StringReader(template)));
}
catch (Exception e) {
    System.err.println("Error loading template = " + e);
    System.exit(1);
}
```


Flowing Content into Your Document

Flow Frames Versus Static Frames ■ *The JCFLOW Object* ■ *Body Flow*
Front Matter Flow ■ *Flows Example*

Chapter 1 provided an overview of how JClass ServerReport flows content into a document. Chapters 2 through 7 discussed the different types of content you can add to your document and the page templates that are required.

In this chapter, we pull the information together and provide some examples of what a more complex program might look like. But first we start with a clarification of the differences between flow frames and static frames.

Terminology: The word “flow” is used in two different contexts in JClass ServerReport. Flow as a verb, as in “to flow content into a frame,” means to direct content into any frame. Flow used as a noun, as in “body flow,” refers specifically to a frame that has been designated as a flow-frame in order to hold the contents of the JCFLOW object.

8.1 Flow Frames Versus Static Frames

As mentioned in the preceding chapter, page templates can have two types of frames: flow frames and static frames. It is important to understand how each type of frame behaves before you begin flowing content into your document.

Content directed to a flow frame will flow sequentially from page to page, as in pages in a book. You can have facing pages or different page templates, and as long as each has a frame designated as a flow frame, the content will flow smoothly regardless of which template is used next. When multiple page templates are in use, each page template indicates which page template should be used next when the current page is full or a section break occurs.

Content placed into a static frame remains contained within the frame on a single page. For example, the header and footer frames are static frames, which means the content of the header and footer are simply repeated on each page. In fact, the content is part of the page template, which gets cloned each time a new page is created. Static frames can also be used for other purposes. For example, you may want to create a template for the title page that has static frames for the title, author, illustrator, publisher, and other information. Static frames in this context give you a great deal of control over the layout of discreet pieces of information. By default, a frame is static until designated as a flow frame. For example, in the following excerpt

from a full page template, header, body, and footer frames are defined. However, only the body frame is designated as a flow-frame.

```
<frame name="header" unit="inches" color="grey">
  <location x="1" y="0.25"/>
  <size width="6.5" height="0.75"/>
</frame>
<frame name="body" unit="inches">
  <location x="1" y="1"/>
  <size width="6.5" height="9"/>
  <column count="2"/>
</frame>
<frame name="footer" unit="inches" color="pink">
  <location x="1" y="10.25"/>
  <size width="6.5" height="0.75"/>
</frame>
<flow-frame name="body"/>
<flow-page name="8p5x11"/>
<flow-section name="8p5x11"/>
```

The last two lines specify which page template to use when either the page is full or there is a section break in the content. The flow-frame, flow-page, and flow-section elements are the only elements on a page template that control what happens when content is flowed to a flow frame.

When flowing content, JClass ServerReport creates JCFRAME objects for each frame defined in the page template. You can also create frames yourself using JCFRAME directly. For example, consider a text book that contains case studies. Each case study is self-contained and outside the main flow of the text, but may share the same page as related content within the main flow of text. In this case, you could create a JCFRAME to hold the case study and then embed or float the frame in the flow in much the same way you would handle an image. For more information, see [Adding Content Using Subframes](#), in Chapter 6. Alternatively, you could add a JCFRAME to a page's frame list and position it in a place where no other frame exists. In this way, it becomes a static frame on that particular page. For more information, see Chapter 7, [Defining Page Templates](#).

8.2 The JCFlow Object

In Chapter 1, [Learning JClass ServerReport Basics](#), you were introduced to the JCFlow object. JCFlow controls the flow of rendered content from frame to frame, page to page, and section to section. Most documents require that you define at least one JCFlow object. When the flow object is created, it is associated with the first flow frame of the first template in the document's template list. From there, it follows the flow-frame, flow-page, and flow-section elements as elements are added to the flow and events happen.

As discussed throughout this guide, it is the properties and methods in JCFlow that enable you to control the following flow-based activities:

- create a flow for the document

- define and print text
- define and embed/float images, icons, components, and `JCFrame` objects
- define hyperlinks on text or images
- begin a new line, paragraph, list, column, frame, page, or section
- track and set page numbering and section numbering
- define and print contents lists in the front matter flow
- change the order of page templates used
- define how to react to events using flow listeners

For more information, see the preceding chapters and look up `JCFlow` in the *API documentation*.

8.2.1 Constructors

At minimum, the `JCFlow` constructor takes one property:

```
JCFlow(JCDocument doc)
```

Other constructors include:

```
JCFlow(JCDocument doc, JCFlowListener fl)
JCFlow(JCDocument doc, JCFlowListener fl, List<JCPage> templates)
JCFlow(JCDocument doc, List<JCPage> templates)
```

where

- `JCDocument` is the parent document to which this flow belongs.
- `JCFlowListener` registers the specified listener for `JCFlow` events.
- `List<JCPage>` defines the page templates to use for the flow. If none are defined, a default page template is created and used for all pages.

8.2.2 Understanding How `JCFlow` Prints Content

The following process describes what happens when text is added to the flow. A similar process is followed for other types of content.

1. The application declares the addition of a text `String` to the flow, calling `JCFlow.print(String)`.
2. `JCFlow.print()` passes the `String` and the current `JCTextStyle` to the current frame using the corresponding `JCFrame.print(JCTextStyle, String)` method.
3. `JCFrame.print()` encapsulates the `String` object in a `StringRender` object, using the `JCTextStyle` to supply formatting information and font metrics information from the current `Graphics` (provided by the `JCPDFPrinter`).
4. `JCFrame.flowPrint()` is called with the new `StringRender` object. `flowPrint()` determines the amount of available space (subject to tab position and alignment,

indentation and margin) on the current line to see whether the String will fit. If the text is too long to fit, `flowPrint()` will attempt to split off part of the String to fit into the available space.

5. The text, or the portion of it that fits on the line, is added to the current line in memory, and any required adjustments are made to the height of the line and its baseline. (If the text is larger than previous elements on the line more space will be needed. Similarly, if the text is in superior subscript mode, then that may increase the height of the line.)
6. If the line has been made too high to fit in the amount of vertical space currently available, then the action is abandoned and the current line is stored in a `EndOfFrameException` object which is thrown to be caught by the `JCFlow` action.
7. If the line fits, but not all (or none) of the text was added to the line, the current line is pasted into the frame and a new line is begun. The remaining text is printed into the new line.
8. If an `EndOfFrameException` is thrown, it is caught in the `JCFlow.print()` method, which will then find the next flow frame and pass to it the current line of text which did not fit, followed by any other pending content.

Please note that the same sequence is followed for embedded objects, except that it is not possible to split them; thus, if embedded objects do not fit on the current line, they are handed in their entirety to the following line.

8.3 Body Flow

The body flow handles the headings, text, images, and so forth that make up your document.

Here are the essential things that you need to do for the body flow:

1. Specify page templates to use for the body flow by setting them on the document.
2. Define and specify a standard text style to use.
3. Create methods—which in turn call `JCFlow` methods—to do the actual work of flowing content into your document.
4. Instantiate the `JCFlow` object for your document.
5. Begin the flow.

6. Flow content (using previously defined methods).
7. End the flow.

The example at the end of this chapter highlights these steps.

8.4 Front Matter Flow

Note: This feature is supported only for PDF output.

Front matter contains all the material that comes before page 1 of your document, which may include the title page, a table of contents, other reference lists such as a list of figures, acknowledgements, and the preface. Usually, the front matter has its own page numbering sequence that is distinct from the rest of the book. Often the difference is highlighted by using Roman numerals for page numbers in the front matter, and then switching to Arabic numerals and restarting from 1 for the body of the book.

JClass ServerReport supports creating front matter through the `beginFrontMatter()` and `endFrontMatter()` methods found in `JCDocument`. You can add any type of content to the front matter flow that you can add to the body flow. You can also add contents list. For more information, see [Table of Contents and Other Reference Lists](#), in Chapter 4.

8.4.1 Understanding How Flow Affects Contents Lists

While JClass ServerReport processes the body flow, contents lists are formed by marking locations in the document where contents list entries are to point, associating the locations with text by creating `JCContentsListEntry` objects, and then adding these objects to a specified `JCContentsList` in the order that they are to appear in the contents list. Once the body flow ends and the front matter flow begins, the now completed contents lists can be printed. For more information, see Chapter 4, [Defining Linked Content](#).

The body flow and the front matter flow each have separate page lists. Each of these page lists are numbered separately starting at page 1 (this can be changed to a different number by setting the `StartingPageNumber` property on the document before the body flow or front matter flow is started). You cannot add a contents list to the body flow because the document cannot manage multiple page lists at the same time.

For the same reason, entries for contents list can only be placed in the body flow. Entries added to a contents list while in the front matter flow are ignored because page numbers for the front matter page list may not yet have been assigned. This means, for example, that if your preface is handled within the front matter flow, you cannot include entries for the preface within your table of contents.

8.4.2 Programming the Front Matter Flow

Here are the essential things that you need to do for the front matter flow:

1. Ensure the body flow is ended.
2. Specify page templates to use for the front matter flow, such as a title page template and a front matter text page template. If none are specified, the templates defined for the body flow are used.
3. Begin the front matter flow by calling the `beginFrontMatter()` method of `JCDocument`. This method returns a newly created `JCFlow` object to be used for the front matter flow. The `beginFrontMatter()` method takes the following arguments: a template list (if you created one), a flow listener, and a contents list (if you want the front matter to begin with a contents list).
4. Call `JCFlow` methods to do the actual work of flowing front matter content into your document.
5. End the front matter flow.

The example in the following section highlights these steps.

8.5 Flows Example

The following code is modified from *JCLASS_SERVER_HOME/examples/sreport/PolarBearsBase.java*. Note that the complete source code is not listed. For definitions of all the methods called, refer to the source code for the example.

Main Method

This is the main method for the program. Some of the key methods that are called are shown after this example. Note that the body flow has to end before the front matter flow begins, and that the bookmark tree needs to be created after the front matter flow ends.

```
/**
 * This demonstrates one use of JClass ServerReport. This program produces a
 * grade school report about Polar Bears. The body of the report
 * contains text and images.
 * @param os The output stream to write to
 * @param resolvingObject object used to help locate the data file
 * @param isDraft if true, create "Draft" watermark
 */
public PolarBearsBase(OutputStream os, Object resolvingObject,
                     boolean isDraft) {
    if (resolvingObject == null) {
        this.resolvingObject = this.getClass();
    }
    else {
        this.resolvingObject = resolvingObject;
    }
}
```



```

// Create the document and load the page templates.
document = new JCDocument(os, createTemplateList(textPage));
document.setCompressed(false);

// Add MetaData.
document.setTitle(TITLE);
document.setAuthor(AUTHOR);
document.setSubject(SUBJECT);
document.setCreator(CREATOR);
document.setKeywords(KEYWORDS);

// Create the standard style for this document.
standardStyle = (JCTextStyle)JCTextStyle.DEFAULT_TEXT.clone();
standardStyle.setNameAndScope("standard-doc", document);
standardStyle.setPointSize(12.0);
standardStyle.setParagraphSpacing(1.5);

// Create the list of templates for the body flow.
JCPage pageTemplate = document.stringToTemplate("text");
if (isDraft) {
    // Add the watermark to the text pages
    createWatermarkOnPage(pageTemplate);
}

// Define the style to use in the footer. The template does not
// have header.
JCTextStyle footerStyle = new JCTextStyle("footer", document);
footerStyle.setAlignment(JCTextStyle.ALIGNMENT_CENTER);
footerStyle.setFont(createBoldFont(14));
JCFrame footerFrame = pageTemplate.stringToFrame("footer");
try {
    // Center the page number in the frame and surround it
    // with dashes so that it has a "- <p> -" look to it.
    footerFrame.print(footerStyle, "- ");
    footerFrame.print(footerStyle, TextMacro.PAGE_NUMBER);
    footerFrame.print(footerStyle, " -");
}
catch (EndOfFrameException e) {
    e.printStackTrace();
}

// Create the list of page templates for front matter flow.
List<JCPage> templateList = createContentTemplateList(isDraft,
    "Table of Contents");
tableOfContents = new JCContentsList("toc", document, templateList);
templateList = createContentTemplateList(isDraft, "List of Figures");
listOfFigures = new JCContentsList("lof", document, templateList);

// Create a new flow to this document (This must be done after modifying
// the templates).
JCFlow flow = new JCFlow(document);

// Flow the content for the BODY FLOW.
writeReport(flow);

// End the body flow.

```

```

        flow.endFlow();

        // Prepare for the front matter flow. First create a template list with
        // a title page and text page templates.
        templateList = createTemplateList(titlePage, textPage);

        // Set Roman numeral page numbers on all but the title page.
        pageTemplate = getTemplateByName(templateList, "text");
        footerFrame = pageTemplate.stringToFrame("footer");
        printRomanNumberFooter(footerFrame);

        // Begin the FRONT MATTER FLOW.
        flow = document.beginFrontMatter(templateList, null);

        // Write the title page.
        writeTitlePage(flow);

        // Print out the table of contents.
        tableOfContents.setFlowPageTemplate("text");
        flow.print(tableOfContents);
        // Print out the list of figures.
        listOfFigures.setFlowPageTemplate("text");
        flow.print(listOfFigures);
        flow.newPage();

        // Write the acknowledgements page.
        writeAcknowledgements(flow);

        // End the front matter flow.
        document.endFrontMatter();

        // Create a bookmark tree (code omitted here)
        ...
    }

```

Methods Called During the Body Flow

The `writeReport(flow)` method that was called in the body flow in turn calls three methods: `doHeading()`, `doImage()`, and `doText()`. All the methods use JCFLOW methods to control what happens to the content during the flow. Notice that the `doHeading()` and `doImage()` methods add entries to contents lists.

```

/**
 * Write a heading, and add it to the Table of Contents.
 *
 * @param flow    the flow
 * @param text    heading text
 * @param tag     heading tag
 * @param location a unique location name for the heading
 */
protected void doHeading(JCFlow flow, String text, String tag,
                        String location) {
    // Set heading style and flow the heading
    // Note: It is important to follow the sequence:
    // (i) set style
    // (ii) new line (makes the new style active)

```

```

        // (iii) set location marker for content list
        // (iv) flow the content
        // Set the location marker immediately before the content that it marks
        flow.setCurrentTextStyle(JCTextStyle.HEADING3);
        flow.newLine();
        flow.markNextLocation(location);
        flow.print(text + ":");
        flow.newLine();
        // Set up the ToC entry
        JCContentsListEntry entry = new JCContentsListEntry(1, tag, text,
                                                             location);
        entry.setAutoNumbering(JCContentsListEntry.AUTO_NUMBERING_PARENT);
        entry.setHyperlinkUsed(true);
        tableOfContents.add(entry);
    }

    /**
     * Write an image, and add it to the List of Images.
     *
     * @param flow    the flow
     * @param image   the image
     * @param text    heading text
     * @param tag     heading tag
     * @param location a unique location name for the heading
     */
    protected void doImage(JCFlow flow, Image image, String text, String tag,
                           String location) {
        if (image != null) {
            // Set an italic style for the Figure caption
            JCTextStyle style = (JCTextStyle)standardStyle.clone();
            style.setAlignment(JCTextStyle.ALIGNMENT_CENTER);
            style.setFontStyle(java.awt.Font.ITALIC);
            // We'll use a half-line above and below the image
            style.setLineSpacing(0.5);
            flow.setCurrentTextStyle(style);
            flow.newLine();
            // Flow the centred image
            flow.markNextFloatLocation(location);
            flow.floatImage(image, JCDrawStyle.CENTER_IN_LINE);
            flow.newLine();
            // Flow the figure caption
            flow.print("Figure " + tag + ": " + text);
            flow.newLine();
            flow.newParagraph();
            // Set up the LoF entry
            JCContentsListEntry entry = new JCContentsListEntry(1, tag, text,
                                                             location);
            entry.setAutoNumbering(JCContentsListEntry.AUTO_NUMBERING_PARENT);
            entry.setHyperlinkUsed(true);
            listOfFigures.add(entry);
        }
    }
}

```

```

/**
 * Write a normal text paragraph.
 *
 * @param flow the flow
 * @param text the text in the paragraph to flow
 */
protected void doText(JCFlow flow, String text) {
    flow.setCurrentTextStyle(standardStyle);
    flow.newLine();
    flow.print(text);
    flow.newParagraph();
}

/**
 * Write the main report.
 *
 * @param flow the flow
 */
protected void writeReport(JCFlow flow) {
    flow.markNextLocation("h0");
    JCTextStyle tStyle = (JCTextStyle)JCTextStyle.HEADING1.clone();
    tStyle.setAlignment(JCTextStyle.ALIGNMENT_CENTER);
    flow.setCurrentTextStyle(tStyle);
    flow.print("Polar Bears");
    flow.newLine();
    flow.newLine();
    JCContentsListEntry entry = new JCContentsListEntry(0, "1",
        "Polar Bears", "h0");
    entry.setTagPrinted(false);
    entry.setHyperlinkUsed(true);
    tableOfContents.add(entry);

    //
    // Load in the images that we are going to use in this document
    // For simplicity in this demo we'll assume that they all load.
    // A robust application should handle image load failures.
    //
    ArrayList<Image> polarpics = new ArrayList<Image>();
    polarpics.add(loadPicture(IMAGE_PB_GLAM));
    polarpics.add(loadPicture(IMAGE_PB_PAIR));
    polarpics.add(loadPicture(IMAGE_PB_FUR));
    polarpics.add(loadPicture(IMAGE_PB_SWIM));

    // Main report (some content omitted)
    doHeading(flow, "Where Polar Bears are found", "A", "h1");
    doText(flow, "Polar Bears live in the Arctic, in the U.S.A, Canada, "
        + "Russia, Greenland and Norway. Polar Bears can live in the "
        + "ocean or on land in the Arctic. They can be found on pack "
        + "ice, coastal Islands, coastlines and Arctic waters.");
    doImage(flow, polarpics.get(1), "Two Canadian polar bears", "1", "f1");
    doHeading(flow, "How Polar Bears give birth", "B", "h2");
    doText(flow, "After the female Polar Bear reaches 5 years, she can have"
        + "cubs. They usually have 2 cubs which weigh 1 pound at "
        + "birth and are about 15 pounds when they leave the den.");
    ...
    doHeading(flow, "Conclusion", "I", "h9");

```

```

doText(flow, "God crafted and gifted Polar bears. God gave them a good "
        + "sense of smell. Their skin is black. They just look "
        + "white because of their fur.");
}

```

Methods Called During the Front Matter Flow

In addition to the contents lists—`tableOfContents` and `listOfFigures`—that were printed from the main class, two methods were called during the front matter flow: `writeTitlePage(flow)` and `writeAcknowledgements(flow)`. Notice that content for the title page is part of the flow with `newLines` added between content. Another way to accomplish the same thing would be to use a title page template with static frames.

```

/**
 * Writes the titlePage
 * @param flow the flow
 */
protected void writeTitlePage(JCFlow flow) {
    // Define the style for the title contents entries
    JCTextStyle style = (JCTextStyle)standardStyle.clone();
    style.setNameAndScope("standard-clone-title", document);
    style.setAlignment(JCTextStyle.ALIGNMENT_CENTER);
    style.setPointSize(14.0);
    style.setLineSpacing(1.8);
    flow.setCurrentTextStyle(style);

    // Loop through the title contents
    for (String titleContent : titleContents) {
        flow.print(titleContent);
        flow.newLine();
    }
}

/**
 * Produce the Acknowledgements page.
 * @param flow the flow
 */
private void writeAcknowledgements(JCFlow flow) {
    JCTextStyle heading = (JCTextStyle)JCTextStyle.HEADING3.clone();
    heading.setAlignment(JCTextStyle.ALIGNMENT_CENTER);
    flow.setCurrentTextStyle(heading);
    flow.print("Acknowledgements");
    flow.newLine(2);
    flow.setCurrentTextStyle(standardStyle);
    flow.print("Thanks to my Dad, without whom this ");
    flow.print(TextMacro.PAGE_TOTAL);
    flow.print(" page report would not have been possible.");
    flow.newLine(2);
    flow.print("Thanks as well to Stuart Hodgins, who provided the ");
    flow.print("photographs, and to the Toronto Zoo, ");
    flow.print("who provided the polar bears in the photographs!");
}

```

Conclusion

You should now have a better sense of how you can flow content into your own document. For a complete example, see *JCLASS_SERVER_HOME/examples/sreport/ PolarBearsBase.java*.

Rendering Your Document

JCDocument Object ■ *Listening for JClass ServerReport Events*

JClass ServerReport allows you to render your document using the `JCDocument` object. The following chapter discusses how to print the document, as well as how to set up listeners to indicate when significant activities have taken place.

9.1 JCDocument Object

JClass ServerReport prints using `JCDocument.print()`. The print command can be as simple as:

```
doc.print();
```

JClass ServerReport can also print a page range (this feature is not supported when creating RTF output). For example, to print pages 17 through 39, use:

```
doc.print(17, 39)
```

9.1.1 Printing to a File

You may already know that `java.io.FileOutputStream` lets you write data to a file.

`JCDocument` formats the output to enable your application to print directly to PDF or RTF files.

9.1.2 OutputPolicy and FlushPolicy

Once a page is created, you can designate whether it will be held until the entire document is finished before it is printed. This is set via the `OutputPolicy` property; please see [OutputPolicy](#).

As well, once pages are printed, you can designate whether pages are to be discarded once printed. You can do this by setting the `FlushPolicy` property. Please see [FlushPolicy](#).

OutputPolicy

Once a page is created, you can designate whether it will be held until the entire document is finished before it is printed. In the `JCDocument` class, the `OutputPolicy` property indicates whether rendered pages are to be held for printing. If set to `OUTPUT_POLICY_IMMEDIATE`

(default), then each page is outputted as it is completed (if all predecessors are complete). If set to `OUTPUT_POLICY_ON_REQUEST`, then completed pages are held in memory until the document is printed.

The `getOutputPolicy()` method gets the document's current policy on outputting completed (rendered) pages. The `setOutputPolicy()` method sets the document's behavior for outputting printed pages. Its parameter, `outputPolicy`, indicates the policy to apply to completed (rendered) pages.

FlushPolicy

Once pages are printed, you can designate whether to save or flush them. In the `JCDocument` class, the `FlushPolicy` property indicates whether pages are to be discarded once printed. If set to `FLUSH_POLICY_ON_OUTPUT` (default), then pages are flushed as they are completed and printed. If set to `FLUSH_POLICY_ALWAYS_SAVE`, then all pages are saved, not flushed, as they are completed and printed.

The `getFlushPolicy()` method gets the document's current policy on flushing completed (printed) pages. The `setFlushPolicy()` method sets the document's behavior for flushing printed pages. The `flushPolicy` parameter of `setFlushPolicy()` designates the policy to apply to completed (output) pages.

9.1.3 PDF Compression

By default, `ServerReport` compresses content streams contained within the PDF output it generates. Compression can be turned on and off with the `setCompressed()` method of the `JCDocument` class.

Note: This feature is not supported when creating RTF output.

9.1.4 Printing Large Documents

When printing documents of any size, but especially for larger documents, better performance is achieved by wrapping the output stream in a `BufferedOutputStream`:

```
BufferedOutputStream bos = new BufferedOutputStream(os, 2048);
JCDocument doc = new JCDocument(bos);
```

This may be especially important in application server or web server environments where exceptions may be thrown if multiple writes are not buffered.

9.1.5 Printing PDF documents from the command line (UNIX)

Besides printing directly from within Acrobat Adobe Reader by choosing **File > Print**, you can print PDF files from the command line. The syntax for printing from the command line is:

```
acroread -toPostScript <options> <pdf filename>
```

For example, to print the file *sample.pdf* to the default printer, type the following:

```
% cat sample.pdf | acroread -toPostScript | lp
```


You can also use `<options>` to control your print job from the command line. For detailed information, please review the Adobe Acrobat Reader manual.

9.1.6 The JCPDFPrinter Object

By default, all PDF documents generated by JClass ServerReport use a feature of PDF known as Automatic Stroke Adjustment. This feature attempts to provide a solution to the problem of maintaining consistent line widths when a high-resolution object, such as a PDF document, is rendered to a low-resolution device, such as a computer screen.

When this Automatic Stroke Adjustment feature is on, you may notice that some lines drawn in the PDF document are moved a pixel or two away from their correct locations. For example, in JClass ServerReport you may notice that table borders protrude slightly from the edges of the table.

JClass ServerReport provides a property (AutoStrokeAdjustment) in the JCPDFPrinter class through which this feature can be turned off.

To turn Automatic Stroke Adjustment off, simply call:

```
((JCPDFPrinter)document.getPrinter()).setAutoStrokeAdjustment(false);
```

When the Automatic Stroke Adjustment feature is turned off, lines of the same width may not be drawn with the same pixel thickness. Neither this nor the aforementioned protruding borders are caused by JClass ServerReport. These aberrations are side effects of the way Adobe Acrobat Viewer renders PDF documents. You should not see either condition when the document is rendered by a high-resolution device, such as a printer.

9.1.7 Defining RTF Behavior for Unsupported Features

Because of the limits of RTF functionality, not all of the JClass ServerReport features are supported when producing RTF output. For a complete list of unsupported features, please refer to Appendix B, “[RTF Limitations and Unsupported Features](#)”.

To determine JClass ServerReport’s behavior when it encounters one of the unsupported features, set the JCDocument’s `UnsupportedOperationAction` property to one of the following values.

Value	Function
<code>JCDocument.UNSUPPORTED_OP_DO_NOTHING</code>	If an unsupported operation is requested, JClass ServerReport will not do anything.
<code>JCDocument.UNSUPPORTED_OP_TRY_RELATED</code>	Default. If an unsupported operation is requested, JClass ServerReport will attempt to perform a related action. If no related action exists, JClass ServerReport will not do anything.

Value	Function
<code>JCDocument.UNSUPPORTED_OP_THROW_EXCEPTIONS</code>	If an unsupported operation is requested, JClass ServerReport will throw an exception.

9.1.8 RTF Output Compatibility

While the default RTF output is configured to be compatible with Microsoft Word 2002, it is possible to change it to be compatible with Microsoft Word '97. To do so, retrieve the current `RTFOutputProperties` object by calling `getOutputProperties()` on your instance of `JCDocument`. Then, call `setCompatibilityLevel(RTFOutputProperties.COMPATIBILITY_WORD_97)` on the `RTFOutputProperties` instance.

Note: Horizontal rules are not supported in Microsoft Word '97.

9.2 Listening for JClass ServerReport Events

If your application needs to be informed about such events as the beginning, completion, or ending of a frame or a page, you can implement the `JCFlowListener` interface and examine the event to take appropriate action. The `JCFlowListener` implementation can either be passed to the `JCFlow` constructor or added to an existing `JCFlow` via the `addFlowListener()` method.

JCFlowEvent

A `JCFlowEvent` occurs when a flow enters or exits a new frame or page as a result of document flow, and also when a frame or page is marked as complete by the resolution of embedded macros. The methods in `JCFlowEvent` are:

JCFlowEvent Method	Description
<code>getCurrentPageArea()</code>	The current <code>PageArea</code> on which the event occurred.
<code>getNextElementName()</code>	The name of the next <code>PageArea</code> to be processed.
<code>getNextPageArea()</code>	The next page relative to where the event occurred.
<code>getSource()</code>	The source of the event, the current <code>JCFlow</code> where the event occurred.

JCFlowListener

`JCFlowListener` methods each take a `JCFlowEvent` as their only parameter.

JCFlowListener Method	Description
<code>frameBegin()</code>	Invoked before the flow to a frame begins.

JCFlowListener Method	Description
<code>frameComplete()</code>	Invoked when the flow to a frame is complete, that is, when all macros in the frame have been evaluated.
<code>frameEnd()</code>	Invoked when the flow is transferred to another frame.
<code>pageBegin()</code>	Invoked before the flow to a page begins.
<code>pageComplete()</code>	Invoked when the flow to a page is complete, that is, when all macros on this page have been evaluated.
<code>pageEnd()</code>	Invoked when the flow is transferred to another page.

JCPrintEvent

A `JCPrintEvent` occurs when a document is opened or closed, or when a page begins or ends. The availability of a `JCPrintEvent` overcomes a limitation of AWT printing. You can now be notified when a document finishes printing.

JCPrintEvent Method	Description
<code>getPageNumber()</code>	Returns the page number or -1 if not applicable.
<code>getEventId()</code>	Returns the <code>eventId</code> . The returned value is one of <code>JCPrintEvent.BEGIN_PAGE</code> , <code>JCPrintEvent.END_PAGE</code> , <code>JCPrintEvent.OPEN_DOCUMENT</code> , or <code>JCPrintEvent.CLOSE_DOCUMENT</code> .

JCPrintListener

`JCPrintListener` methods each take a `JCPrintEvent` as their only parameter.

JCPrintListener Method	Description
<code>openDocument()</code>	Invoked before a document has been printed.
<code>closeDocument()</code>	Invoked after a document has been printed.
<code>beginPage()</code>	Invoked before a page has been printed.
<code>endPage()</code>	Invoked after a page has printed.

Setting Document-level PDF Features

Metadata ■ Security ■ Accessibility and Structured PDF Documents

You can add some document-level PDF features to your PDF output, specifically metadata, security, and accessibility.

10.1 Metadata

Metadata for a PDF document includes information about the document, such as the title, author, and keywords. Metadata helps readers and search engines find and identify PDF documents.

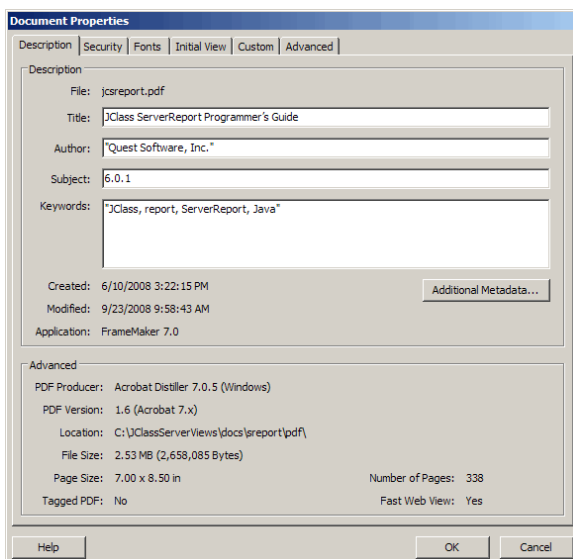


Figure 53 Metadata displayed in Adobe Acrobat Document Properties dialog

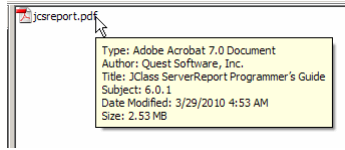


Figure 54 Metadata displayed in Windows Explorer

JClass ServerReport supports metadata in the `JCDocument` object. The following table lists the available properties and identifies a typical usage for each type of metadata.

Note: You do not need to follow the typical usage described here. If your organization has established standards for metadata, you should be guided by those standards.

Property	Description
Author	The person, people, or organization that claims authorship of the document, images, or other intellectual property contained within the PDF document.
Title	The name of the document as it appears within the document itself.
Subject	A concise description of what the document is about.
Keywords	A comma-separated list of words that a reader or a search engine can use to find your document.
Creator	The application in which the document was created. For example, “JClass ServerReport” would be the creator application for a PDF document created using JClass ServerReport.

To add metadata to your document, instantiate the `JCDocument` object and then set some or all of the metadata properties as Strings.

```
// Create the document and load the page templates
document = new JCDocument(os, createTemplateList(textPage));

// Add Metadata
document.setTitle("JClass ServerReport Programmer's Guide");
document.setAuthor("Quest Software, Inc.");
document.setSubject("6.0.1");
document.setCreator("JClass ServerReport");
document.setKeywords("JClass, report, ServerReport, Java");
```

10.2 Security

It is possible to impose security features on PDF files generated by JClass ServerReport. Security features are used to protect the PDF and to limit functionality. For example, the viewing of a PDF document may require a user to enter a password. Other functionality that may be restricted includes editing, copying of contents, or printing.

A file must be encrypted in order to impose such restrictions. A Java Cryptography Extension (JCE) compliant RC4 encryption provider must be installed to lock the PDF document from unauthorized access. Full encryption of the generated document is required in order to lock it, which is provided by the RC4 encryption provider.

10.2.1 The RC4 Encryption Provider

The locking of a PDF document is achieved through RC4 encryption. Any functional JCE compliant encryption provider that supports the RC4 cipher can be used to enable these features, assuming that it is properly installed. If none is installed, a `NoSuchAlgorithmException` is thrown if these methods are attempted, which results in stack trace output.

To find an encryption provider, you can search the Web with the keyword “encryption” or start with this list of providers: http://java.sun.com/products/jce/jce122_providers.html

To install an encryption provider:

1. Include the JAR that contains the relevant classes in the classpath.
2. Add the encryption provider package to the list of providers in the *java.security* file (located in the *jre\lib\security* directory).

For example, if using the RC4 encryption provider by BouncyCastle, the file may appear similar to the following:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
security.provider.3=com.sun.rsa.jca.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=org.bouncycastle.jce.provider.BouncyCastleProvider
```

Note: Security providers one through five are default providers in the Sun JDKs. This list may differ depending on your JDK and settings.

10.2.2 Setting Encryption and Password Security Options

The `JCPDFSecurity` object’s properties determine the security features that are imposed on the PDF. To enable encryption, set a `JCPDFSecurity` object with the `Encrypt` property `True`, on the `JCDocument`.

You can specify that the encrypted document be protected with a password. To protect a document with a password, set the `OwnerPassword` or `UserPassword` properties of the `JCPDFSecurity` object.

If a user attempts to open a document that is password-protected, the viewer application prompts for a password. If either the owner or user password is entered, the viewer application opens, decrypts, and displays the document. If an owner password is entered, the user has unlimited access, including the ability to change passwords. If a user password is entered, operations are restricted according to the permission flags. See the following section for a description of all the permissions that can be set.

Example

```
//Create the document
JCDocument document = new JCDocument(outputStream,
    JCDocument.BLANK_8p5X11);

//Set security parameters
if (JCPDFSecurity.isEncryptionProviderAvailable()) {
    JCPDFSecurity security = new JCPDFSecurity();
    security.setOwnerPassword("ownerpass");
    security.setUserPassword("userpass");
    security.setEncrypt(true);
    security.setAllowPrint(false);
    security.setAllowPrintAtFullQuality(false);
    security.setAllowEditContent(false);
    security.setAllowCopyContent(false);
    security.setAllowCopyForAccessibility(false);
    security.setAllowEditAnnotation(false);
    security.setAllowAssmeble(false);
    security.setEncryptionStrength(128);
    document.setSecurityParameters(security);
}
```

10.2.3 The JCPDFSecurity Object

The `JCPDFSecurity` object is responsible for PDF security features, and must be set before the document is printed. The following outlines the methods that can be applied to this object, and their influence on the output PDF file.

Property	Function
Encrypt	Enables encryption. Values are True or False; default is True.
OwnerPassword	Defines the owner password that is associated with the document
UserPassword	Defines the user password that is associated with the document.

Property	Function
<code>AllowPrint</code>	Determines whether or not the PDF can be printed. Values are True or False; default is True.
<code>AllowPrintAtFullQuality</code>	Determines whether or not the PDF can be printed at high quality. Values are True or False; default is True.
<code>AllowModifications</code>	Determines if the PDF can be modified. Values are True or False; default is True.
<code>AllowCopy</code>	Determines if the text and images contained in the PDF can be copied for purposes other than accessibility. Values are True or False; default is true.
<code>AllowCopyForAccessibility</code>	Determines if the text and images contained in the PDF can be extracted for accessibility purposes. Values are True or False; default is True.
<code>AllowAssemble</code>	Determines if the assemble feature can be applied to the PDF. Values are True or False; default is True.
<code>EncryptionStrength</code>	Defines the length of the encryption key. Values are 40 or 128; default is 40. Using another value results in an <code>IllegalArgumentException</code> .

10.3 Accessibility and Structured PDF Documents

Accessibility is about making your PDF document readable by as many people as possible via as many devices as possible. Many people rely on assistive technologies to be able to read documents online, such as a screen reader or magnification software. These types of technologies require an accessible PDF document to perform optimally. Even users of hand-held devices benefit from accessible PDF documents. For example, a smart phone or similar device can reflow the content of an accessible PDF document so that the line breaks are determined based on the width of the display area, rather than the original document layout, which reduces left/right scrolling and significantly improves the reading experience.

Accessibility for PDF documents requires that documents have an explicit structure. You create a structure by tagging the structural elements in your document, such as headings, paragraphs, sections, lists, tables, and figures. It is similar to a markup language; indeed many of the standard PDF tags correspond to HTML tags.

In JClass ServerReport, you create the tags required for a structured PDF document using methods in `JCFlow` and `JCFrame`. You then specify the output as a structured PDF document

using JCPDFPrinter. The following sections describe how to add the tags and output the structured document.

10.3.1 Adding Tags

As with any markup language, you need to mark the beginning and end of all structured elements within your document. The following table summarizes the methods to use and shows the PDF tag that results.

Structured Element	Begin Tag	End Tag	Tag
Section	<code>newSection()</code>	<code>endSection()</code>	Sect
Heading	<code>newHeading()</code>	<code>endHeading()</code>	H
Paragraph Text	<code>newParagraph()</code>	<code>endParagraph()</code>	P
Table	<code>newTable()</code>	<code>endTable()</code>	Table
List	<code>newList()</code>	<code>endList()</code>	L
List Item	<code>newListItem()</code>	<code>endListItem()</code>	LI
Hyperlink	<code>beginHyperlink()</code>	<code>endHyperlink()</code>	Link

For lists, list items, and hyperlinks, the methods shown are the same methods used to define lists and hyperlinks; the tags are created automatically. You do *not* need to add any additional method calls to your code to create the tags.

For images, the begin and end tags are automatically added to the document when the `embedImage()` or `floatImage()` methods of JCFLOW or JCFrame are called.

For all other elements, you need to call the methods shown in the table. To create tags, bracket the target `print()` method with the corresponding `new()` and `end()` methods. For example, the following code snippet show how to add a tag to a heading.

```
protected void doHeading(JCFlow flow, String text) {
    flow.setCurrentTextStyle(JCTextStyle.HEADING3);
    flow.newLine();

    // Mark the beginning of the heading
    flow.newHeading();

    // Print the heading
    flow.print(text);

    // Mark the end of the heading
    flow.endHeading();

    flow.newLine();
}
```

10.3.2 Tables in Structured PDF Documents

In JClass ServerReport, table cells can contain all kinds of content, including JCFRAME objects. However, the classes and methods that make it possible to support JCFRAME objects within cells also make it difficult to add structure to the content of those cells. Therefore, to add tables for use in a structured PDF document, you need to use different methods to add content to cells.

Recall that in Chapter 5, “[Defining Table-Based Content](#)”, the examples show using the `printToCell()` method to add content to cells. If you want to support a structured PDF document, you need to call the `getCell()` and `setCellValue()` methods instead. You will also need to ensure that your content fits within the width of the cell. There is no automatic control of text wrapping in these stripped down methods.

For example, you would change this:

```
table.printToCell(0, 0, style, "Labrador");
```

to this:

```
table.getCell(0, 0).setCellValue("Labrador");
```

The following code shows how to set up a table using the `getCell()` and `setCellValue()` methods. The last section shows how to implement the `newTable()` and `endTable()` methods for the structured PDF document.

```
// Create a table style with orange headers
JCTableStyle tableStyle = new JCTableStyle("newStyle", document);
tableStyle.getHeaderStyle().setBackground(Color.orange);

// Create a table, using the new table style
JCPageTable table = new JCPageTable(document, 3,
    new JCUnt.Measure(JCUnt.INCHES, 1.0),
    tableStyle);
table.setColumnWidth(0, new JCUnt.Measure(JCUnt.INCHES, 1.5));

// Create the header entries for the table
JCPageTable header = table.createHeaders();
header.getCell(0, 0).setCellValue("Dogs");
header.getCell(0, 1).setCellValue("Cats");
header.getCell(0, 2).setCellValue("Horses");

// Set the data in the body of the table, adding line breaks where
// necessary to ensure the data fits within the column width
table.getCell(0, 0).setCellValue("Labrador Retriever\nBasset Hound");
table.getCell(0, 1).setCellValue("Persian");
table.getCell(0, 2).setCellValue("Shetland");
table.getCell(1, 0).setCellValue("Collie");
table.getCell(1, 1).setCellValue("Siamese\nSavannah");
table.getCell(1, 2).setCellValue("Arabian\nLipizzan\nBlazer");

// Add the table to the structured PDF document
flow.newTable();
flow.print(table);
flow.endTable();
```

10.3.3 Creating a Structured PDF Document

After you have tagged all your structured elements, you need to tell JClass ServerReport that you want to output a structured PDF document. The following example shows how to set the `LogicallyStructured` property of `JCPDFPrinter` to `true` to create a structured document. If the `LogicallyStructured` property were omitted or set to `false`, the document would be outputted as an unstructured PDF document, even though the tags are still present.

```
JCPrinter printer = document.getPrinter();
if (printer instanceof JCPDFPrinter) {
    JCPDFPrinter pdfPrinter = (JCPDFPrinter)printer;
    pdfPrinter.setLogicallyStructured(true);
}
```

Adding Formulas to JClass ServerReport

[Introduction](#) ■ [util.formulae's Hierarchy](#) ■ [Expressions and Results](#) ■ [Math Values Operations](#) ■ [Expression Lists](#) ■ [Exceptions](#) ■ [Using Formulas in JClass ServerReport](#)

11.1 Introduction

The *formulae* package in *com.klg.jclass.util* has special capabilities for working with mathematical objects. Similar to the way that objects such as `java.lang.Double` wrap a primitive type, those in *com.klg.jclass.util.formulae* encapsulate mathematical expressions (operators) whose operands may be scalars, vectors (in the mathematical sense), and matrices. These objects may then be stored as the generalized values of cells in a JClass ServerReport `JCPageTable`, or in a JClass LiveTable (available as part of JClass DesktopViews – please see <http://www.quest.com> for more information), where they may be evaluated at run time to produce results based on the then-current data.

In addition, subclasses of `MathValue`, which are wrappers for generalized scalars, vectors, and matrices, provide several methods for converting an expression to a value and to a `String`, as well as other methods useful when dealing with these objects.

11.2 util.formulae's Hierarchy

The interfaces, abstract classes and derived classes, including possible exception classes, are shown in Figure 55.

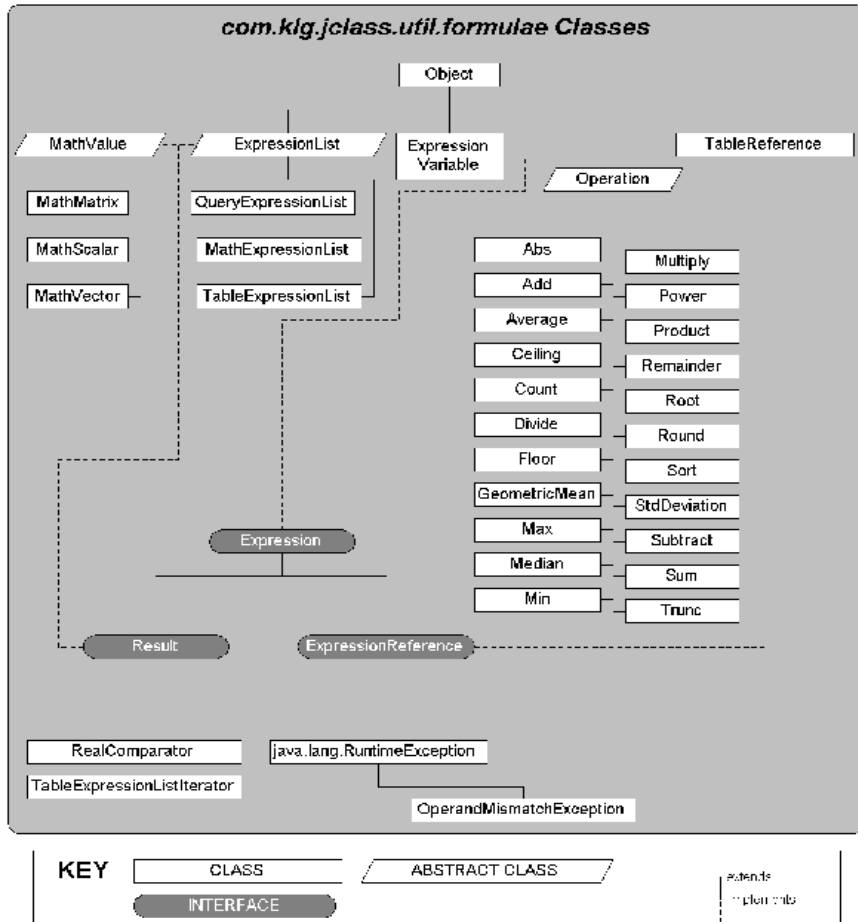


Figure 55 The inheritance hierarchy for `com.klg.jclass.util.formulae`.

The diverse set of mathematical operations permit you to compose complex mathematical formulas and provide references to them.

11.3 Expressions and Results

The top-level interface for the *com.klg.jclass.util.formulae* package is *Expression*, whose sole method is *evaluate()*. Any object that functions as an expression must have an *evaluate()* method that knows how to operate on data that might be a scalar, a vector, or a matrix. Applying the *evaluate()* method to an *Expression* produces a *Result*, which is a marker interface that identifies *Expression* types which are valid return types from the evaluation of other *Expressions*.

An *Expression* may be an *Operation*, as in

```
Expression f = new Add(op1, op2);
```

which, after evaluation, returns a *Result*.

11.4 Math Values

The abstract class *MathValue* forms the root for all derived constant-based result/data classes. It satisfies the *Expression* interface by defining an *evaluate()* method, which simply returns the *MathValue* as a *Result*. Its concrete subclasses are *MathMatrix*, *MathScalar*, and *MathVector*. Because *MathValue* has an *evaluate()* method it is an *Expression*. Thus, *MathValues* may be passed as *Expression* objects.

MathValue Methods

MathValue Method	Description
<code>evaluate()</code>	Satisfies the <i>Expression</i> interface by returning the stored value. No evaluation is required because no operation is implied.
<code>getDataFormat()</code>	Retrieves the <i>NumberFormat</i> associated with this data.
<code>matrixValue()</code>	Gets the contents of this <i>MathValue</i> as a matrix of <i>Numbers</i> .
<code>numberValue()</code>	Gets the contents of this <i>MathValue</i> as a <i>Number</i> .
<code>setDataFormat()</code>	Sets a <i>NumberFormat</i> to use on the contents of this <i>MathValue</i> .
<code>vectorValue()</code>	Gets the contents of this <i>MathValue</i> as a vector of <i>Numbers</i> .

Note: The subclasses of *MathValue* override all but the first method. Since, for example, *matrixValue()* is not appropriate to a *MathScalar*, it throws an *UnsupportedOperationException* if it is called. Other method-data type mismatches also throw *UnsupportedOperationExceptions*. The method tables for the subclasses indicate which methods are data type mismatches for the given class.

11.4.1 MathScalar

`MathScalar` is a scalar constant represented as a `MathValue`. By encapsulating it in this fashion it can support integer and real numbers, and it can be extended if necessary to support other types of scalar numbers. Its data field is a `realValue`, a `Number` that is output based on the current `dataFormat` kept in `MathValue`.

Example:

```
double s1 = 10.0; MathValue ss1 = new MathScalar(s1);
```

MathScalar Constructors

The no-argument constructor `MathScalar()` creates an instance that encapsulates the value 0.0, while the other three constructors take a `double`, an `int`, or a `java.lang.Number` argument.

MathScalar Methods

MathScalar Method	Description
<code>matrixValue()</code>	Throws an <code>UnsupportedOperationException</code> .
<code>numberValue()</code>	Gets the contents of this <code>MathValue</code> as a <code>Number</code> .
<code>toString()</code>	Returns a <code>String</code> representation of this value.
<code>vectorValue()</code>	Throws an <code>UnsupportedOperationException</code> .

11.4.2 MathVector

`MathVector` is a representation of the class of vectors in a linear algebra sense. They may also be used as operands in matrix multiplication. A `MathVector` encapsulates a list of values which may be integers, doubles, or more generally, objects of type `Number`. It has methods for retrieval or modification of a value at a particular index, and for outputting the list as a `String`. The operators discussed in the next section accept these objects as operands.

Example:

```
double[] ed = {2.71828, 3.1415927, 1.6020505};  
MathValue mv = new MathVector(ed);
```

MathVector Constructors

The constructors for `MathVector` parallel those for `MathScalar`, except they take arrays as parameters rather than single values.

MathVector Methods

MathVector Method	Description
getValueAt()	Retrieves the value at a particular index in the vector.
matrixValue()	Throws an UnsupportedOperationException.
numberValue()	Throws an UnsupportedOperationException.
setValueAt()	Sets the value at a particular index in the vector.
toString()	Outputs the value of this vector as a String.
vectorValue()	Gets the contents of this MathValue as a vector of Numbers.

11.4.3 MathMatrix

MathMatrix is a representation of the class of matrices, again in the sense of linear algebra. The package implements the basic addition and multiplication operations in matrix algebra, including multiplying a matrix by a vector. It has methods for retrieval or modification of a value at a particular pair of indices, and for outputting the matrix as a String. The operators discussed in the next section accept these objects as operands. For example:

```
double[][] m1 = {{1.1, 1.2, 1.3},
                 {2.1, 2.2, 2.3},
                 {3.1, 3.2, 3.3}};
MathValue mm = new MathMatrix(m1);
```

MathMatrix Constructors and Methods

The constructors for **MathMatrix** parallel those for **MathScalar**, except they take 2D arrays as parameters rather than single values.

MathMatrix Method	Description
getValueAt()	Retrieves the value at a particular row, column pair of index values in the matrix.
matrixValue()	Gets the contents of this MathValue as an array of Numbers.
numberValue()	Throws an UnsupportedOperationException.
setValueAt()	Sets the value at a particular row, column pair of index values in the matrix.
toString()	Outputs the value of this vector as a String.
vectorValue()	Throws an UnsupportedOperationException.

11.5 Operations

The abstract `Operation` class defines the basic elements of an operator. Binary operators have a left and right operand, which enables the correct ordering to be applied to matrix operations and any other non-commutative operators. Unary operators have a single operand. For example:

```
double[] ed = {2.71828, 3.1415927, 1.6020505};
double[] rd = {(Math.sqrt(5.0) + 1.0) / 2.0, 4.0, 32.0};

MathValue e = new MathVector(ed);
MathValue r = new MathVector(rd);

Expression add = new Add(e, r);
```

Operation Constructors

There is a no-argument constructor that creates a generic operator, and there are constructors for every unary and binary permutation of `Expressions` and `Numbers`. A sample constructor is: `Operation(Expression left, Expression right)`.

Operation Methods

Operation Method	Description
<code>clone()</code>	Returns a deep-copy clone of the operation and of all operands.
<code>evaluate()</code>	Returns a <code>Result</code> containing the evaluation of the expression.

11.5.1 The Defined Mathematical Operations

Unary Operators

Unary operators take one parameter, which is either an `Expression` or a `Number`. Because they are `Expressions` they all have an `evaluate()` method which returns a `Result`.

Operator	Description
<code>Abs</code>	The class for the absolute value operation. The operand may be a <code>Number</code> or an <code>Expression</code> , which may be a <code>MathScalar</code> or an <code>ExpressionList</code> , but not a vector or a matrix.
<code>Ceiling</code>	<code>Ceiling</code> is defined as the least integer greater than or equal to the operand, which may be a <code>MathValue</code> .
<code>Floor</code>	<code>Floor</code> is defined as the greatest integer less than or equal to the operand.
<code>Root</code>	Returns the positive square root of its operand.
<code>Round</code>	<code>Round</code> is defined as nearest integer to the operand. Rounding is done to an even number if the operand is exactly midway between two integers.

Operator	Description
Trunc	Takes the integer part of a number. Equivalent to rounding to the nearest integer closer to zero. Example: <code>trunc(-3.5) = -3</code> .

Binary Operators

Binary operators take two parameters, which are either Expressions or Numbers. Because they are Expressions they all have an `evaluate()` method which returns a Result,

Operator	Description
Add	Adds two Expressions. If the Expressions are vectors of the same length, pairwise addition is performed. Matrices may be added providing the two operands have the same number of rows and columns. Unary addition is possible, and returns the evaluated operand.
Average	Average (arithmetic mean) is defined as the sum of all elements divided by the number of elements. Its one-parameter constructor is an Expression, usually a list. Its two-parameter constructors are combinations of Expressions and Numbers.
Count	Count determines the total number of elements in its operands. Its one- and two-parameter constructors take one or two Expressions (usually a list or lists) and count their elements.
Divide	Division is the ratio of two operands. The left operand is the numerator and the right operand is the denominator.
GeometricMean	Geometric mean is defined as the n th root of the product of a set of n numbers. Its one-parameter constructor takes an Expression, usually a list. Its two-parameter constructors take combinations of Expressions and Numbers, multiplying all elements together and taking the n th root.
Max	Max is defined for a pair of elements or across a list. It selects the largest element. Its one-parameter constructor takes an Expression, usually a list. Its two-parameter constructors take combinations of Expressions and Numbers, examining all elements and selecting the largest.
Median	The Median of a list is the middle element of a sorted list, or the average of the two middle values if the list has an even number of elements. Its one- and two-parameter constructors take one or two Expressions.

Operator	Description
Min	Min is defined for a pair of elements or across a list. It selects the smallest element. Its one-parameter constructor takes an Expression, usually a list. Its two-parameter constructors take combinations of Expressions and Numbers, examining all elements and selecting the smallest.
Multiply	Multiplication is the product of a pair of elements. Its two-parameter constructors take combinations of Expressions and Numbers, examining all elements and selecting the smallest.
Power	The exponentiation (^) operation. Its two-parameter constructors take combinations of Expressions and Numbers. The left operand is the base and the right operand is the exponent.
Product	A product can be performed on a pair of elements or across a list. The product of an ExpressionList is the product of its individual members. Multiplication order is left-to-right, and first element of a list to last element. The result of a matrix multiplication may depend on the order of the operands.
Sort	This operation returns a sorted list of the given elements. Any secondary or nested lists are flattened.
StdDeviation	The sample standard deviation, given by $sd = \sqrt{(\sum (1 \text{ to } n)(\text{element} - \text{average})^2) / (n - 1)}$, where n is the number of samples and average is the sample average. It has one- and two-parameter constructors consisting of Expressions.
Subtract	The difference between two numbers. It has two-parameter constructors that take combinations of Expressions and Numbers.
Sum	A sum can be performed on a pair of elements or across a list. Its two-parameter constructors take combinations of Expressions and Numbers. Its one-parameter constructor usually takes an ExpressionList.

11.5.2 Reducing Operations to Values

Since Operations are Expressions, they all have an `evaluate()` method. Evaluation returns a Result, which may be converted to a String for printing. Here is an example:

```
double edd = 2.0;
double exp = 8.0;
MathValue eddy = new MathScalar(edd);
MathScalar expy = new MathScalar(exp);

double[] ed = {2.71828, 3.1415927, 1.6020505};
```

```

MathValue e = new MathVector(ed);

Expression pow = new Power(eddy, expy);
Expression powr = pow.evaluate();
// Either one of these has a toString() method
System.out.println("Power without evaluate(): " + pow);
System.out.println("Power with evaluate(): " + powr);

```

After which the following is written on the output:

```

Power without evaluate(): com.klg.jclass.util.formulae.Power@eb4f3b8c
Power with evaluate(): 256.0

```

You see that calling `evaluate()` is necessary to have a value returned by the implicit `toString()` call.

11.6 Expression Lists

Expression lists are handy containers that permit you to perform an operation on a group of values.

MathExpressionList

The example shown here uses the binary form of `Add` to find the grand total of all the elements in two `ExpressionLists`.

```

// Expression Lists
Expression[] exprs1 = {null, null, null, null, null, null, null,
                       null, null, null};
for (int i = 0; i < 10; i++){
    exprs1[i] = new MathScalar(95 + i);
}
ExpressionList explist1 = new MathExpressionList(exprs1);

Expression[] exprs2 = {null, null, null, null, null, null, null,
                       null, null, null};
for (int i = 0; i < 10; i++){
    exprs2[i] = new MathScalar(95 + i);
}
ExpressionList explist2 = new MathExpressionList(exprs2);

sssl = new Sum(explist1, explist2);
qqqq1 = sssl.evaluate();
System.out.println(
    "Summing ExpressionLists with evaluate(): " + qqqq1);

```

Here's the output:

```

Summing ExpressionLists with evaluate(): 1990

```

QueryExpressionList

A `QueryExpressionList` is designed as a wrapper for a set of `Expressions` stored in a JDBC-type `ResultSet`, that is, the result of a database query. Users of `JClass DataSource` may also use this facility. `JClass DataSource` is available as a part of the `JClass DesktopViews` suite.

TableExpressionList

Expression lists may be used to extend data from portions of a `JCPageTable` to produce summary reports. See Section 11.8, [Using Formulas in JClass ServerReport](#) for details.

11.7 Exceptions

OperandMismatchException

Various operations such as adding a number to a vector are not defined, whereas other operations such as multiplying a vector by a number can be interpreted as a scaling operation. At compile time numbers, vectors, and matrices can be declared as generic `Expressions`, making it impossible to predetermine which operations are not permitted. A run time check of the validity of an operation must be made. If a mathematical construct is evaluated and found to be illegal, the class throws an `OperandMismatchException`.

ClassCastException

There are cases where a run time class cast exception may occur. While most of these should be avoidable by selecting the correct class (such as using `Product` rather than `Multiply` when multiplying two vectors) the fact that both take `Expressions` as their parameters makes it difficult to avoid the possibility of an end user passing in an incorrect type if your application permits flexible user input. You may permit substitution of one arithmetic class for another, since they are all `Operations`. This also opens the door to class cast exceptions.

If the possibility exists for either of these exceptions, your code should attempt to handle it.

11.8 Using Formulas in JClass ServerReport

11.8.1 Performing a Mathematical Operation on a Range of Cells

Expression Lists and Expression References

Expression list objects hold a group of `Expressions`. `ExpressionList` is an abstract class whose methods permit the inclusion of additional elements to those already present, a method for removing elements or clearing all elements, for retrieving an element, and for comparing with another list. These operations are common to the concrete classes `MathExpressionList`, `QueryExpressionList`, and `TableExpressionList`.

Expression lists may be used as arguments for all mathematical operations. When given an expression list, evaluating a unary operator such as `Abs` returns a list containing the absolute

values of its input list. Binary operators may return a single result or a list. Given expression lists, the mathematical operators Abs, Add, Ceiling, Divide, Floor, Multiply, Power, Remainder, Root, Round, Sort, and Subtract return lists, while Average, Count, GeometricMean, Max, Median, Min, Product, and Sum all return a single result after evaluate() has been called on them.

Use TableExpressionList to perform an operation over a range of cells in a JCPageTable. The following code snippet shows that the required parameters are a table data model and a block of cells.

```
Expression expression = new TableExpressionList(
    pageTable.getTableData(),
    new MathScalar(startRow),      // first row
    new MathScalar(endRow),        // last row
    new MathScalar(startColumn),   // first column
    new MathScalar(endColumn)     // last column
);
Sum sum = new Sum(expression);
```

The next code fragment places the formula for the sum in a selected cell. The formula is evaluated and the value of the sum is written to the designated cell.

```
pageTable.getCell(i, j).setCellValue(sum);
```

The advantage of using TableExpressionLists is that the formulas containing them can be evaluated after all table data has been filled in.

Part II

Using JClass ServerReport with XML

JClass ServerReport XML Tutorial

*[Introduction](#) ■ [Lexicon](#) ■ [Overview of the XML changes](#) ■ [Starting the XML Document](#)
[Creating a Page Template](#) ■ [Creating Sections](#) ■ [Adding and Formatting Text](#)
[Adding Hyperlinks](#) ■ [Using Macros](#) ■ [Defining Draw Styles and Adding Tables](#)
[Triggering an External Java Class](#) ■ [ExternalTutorialHandler source code](#) ■ [Adding Front Matter](#)
[The Completed XML Document Code](#)*

Important: This tutorial is only suggested for users who understand XML programming concepts, and is not intended to teach you how to program with XML. Please make sure that you are familiar with XML and its programming concepts before attempting this tutorial.

If you are not familiar with XML, the JClass ServerReport Designer has been created to help you design your XML documents in a WYSIWYG fashion. For more information, see [JClass ServerReport Designer](#).

12.1 Introduction

This tutorial shows you how to start using JClass ServerReport with XML, without using the JClass ServerReport Designer to create a simple page template. It will encompass many different activities commonly performed when creating a JClass ServerReport document. To see the complete XML code, see Section 12.14, [The Completed XML Document Code](#). While proceeding through the tutorial, remember that you must always follow the proper XML structure, and use both start and end tags, while programming in XML. This tutorial, which is broken into nine major sections, will show you how to produce a PDF document that uses many of JClass ServerReport's XML capabilities. For more information on any of the XML tags used in this tutorial, please reference Chapter 13, [XML and JClass ServerReport](#).

Note: Comments that are in the XML tutorial will not be explained, or incorporated in the tutorial procedures.

12.1.1 Running the Tutorial

There are two different ways that you can choose to run the tutorial.

Standalone

In a command prompt, navigate to the `JCLASS_SERVER_HOME/examples/sreport/tutorial/xml` directory, and type:

```
java examples.sreport.tutorial.xml.RunXMLTutorial in.xml out.pdf
```

where `in.xml` is one of the tutorial xml files in that directory and `out.pdf` is the desired pdf file. To produce an RTF file, simply replace `out.pdf` with `out.rtf`.

Servlet

Add the `sreport-samples.war` file to your web server and pull up the `sreport-samples/index.html` file in your browser. Click on the *tutorials* link, which will bring you to a page that contains a link for each tutorial step.

This paragraph uses a 12-point TimesRoman font to render the text that it contains. This [link](#) points to the Quest web-site. This [link](#) points to the next section of this document.

This paragraph uses an 18-point TimesRoman font to render the text that it contains.

This paragraph uses an italicized 18-point TimesRoman font to render the text that it contains.

This paragraph uses a 12-point TimesRoman font and a bold tag to render the text that it contains

This paragraph uses a 12-point TimesRoman font and right alignment to render the text that it contains.

City	Population (in millions)
Toronto	
New York	
Chicago	
Vancouver	

Figure 56 Sample PDF output from the tutorial.

12.2 Lexicon

There are a few XML-specific terms that are commonly used throughout this tutorial:

- **Element:** An element is essentially XML data, that is bound by tags. For example:
`<page-template title="simple">`
- **Empty Element:** An empty element is an element that does not contain any other elements, and does not require a closing tag. For example:
`<flow-page name="simple"/>`
- **Nested Element:** A nested element is an element that is bounded by another element. In the following example, `size` is a nested element:
`<frame name="main" unit="inches">`
`<size width="6.5" height="9"/>`
`</frame>`
- **Attribute:** An XML attribute is the same as a Java attribute; it is directly associated with the element for which it exists, and is simply a named value or relationship to the entity. In the following example, `width`, and `height` are attributes:
`<size width="2.0" height="4.5"/>`

12.3 Overview of the XML changes

The following sections walk you through the creation of an XML document using JClass ServerReport. There are many steps involved in creating the final product; to run the tutorial, see [Running the Tutorial](#) in Section 12.1, [Introduction](#).

Below is a summary of the changes that you will be making throughout the course of this tutorial:

Section	Description	XML File
Section 12.4, Starting the XML Document	Start the tutorial by establishing the basic XML shell, to which you will add content and design elements.	<i>xml-tutorial-1.xml</i>
Section 12.5, Creating a Page Template	Create a page template, into which you will eventually feed text and images.	<i>xml-tutorial-1.xml</i> and <i>xml-tutorial-2.xml</i>
Section 12.6, Creating Sections	Learn how to create different sections for your PDF. Sections can be compared to chapters in a book.	<i>xml-tutorial-3.xml</i>
Section 12.7, Adding and Formatting Text	Design different text styles and apply them to text, which you will then feed in to your page and section templates.	<i>xml-tutorial-4.xml</i>

Section	Description	XML File
Section 12.8, Adding Hyperlinks	Make some of the text you just added into hyperlinks, which allows you to link your document to internal locations and external URLs.	<i>xml-tutorial-5.xml</i>
Section 12.9, Using Macros	Apply macros to your XML document to produce dynamic content.	<i>xml-tutorial-6.xml</i>
Section 12.10, Defining Draw Styles and Adding Tables	Define draw styles in your document and apply them to a table, which you will create.	<i>xml-tutorial-7.xml</i>
Section 12.11, Triggering an External Java Class	Learn how to use the <code><external-java-code></code> tag to trigger an external java class.	<i>xml-tutorial-8.xml</i>
Section 12.11.2, ExternalTutorialHandler source code	Add document-level metadata and some lists.	<i>xml-tutorial-9.xml</i>
Section 12.13, Adding Front Matter	Create a front matter flow with a title page, table of contents, and a foreword.	<i>xml-tutorial-9.xml</i>

Each section will begin with the code sample that you will be adding to your XML shell. Lines of code in *italic* represent the lines that already exist in the code that surrounds what you will be adding. To see the finished product's XML code, see Section 12.14, [The Completed XML Document Code](#).

12.4 Starting the XML Document

```

<?xml version="1.0"?>
<!DOCTYPE document SYSTEM "JCServerReport.dtd">
<document>

</document>

```

To start your XML document, you will need to write a declaration and reference the DTD that you are going to use. In this case, you will be declaring that you are using XML version 1.0, and that you are referring to the *JCServerReport.dtd*.

You will also identify the root node by adding start and end tags that refer to the type of node you are creating. In this case, you will be creating a document, and thus the root node is `document`.

This is a very basic shell for your JClass ServerReport XML document. Note that this does not contain any text, and only defines that XML will be used.

12.5 Creating a Page Template

Page templates are used to define the format of the pages where text will be printed. Page template information should be placed directly following the opening `<document>` tag, as it is the first basic unit of the XML file.

In this tutorial, you will have the opportunity to create two different page templates and experiment with the flow of text.

12.5.1 Page Template and Flow

This section will instruct you on the basic elements needed to create a page template, define different sections, design frames on the page, and add items to a flow.

Creating a Page Template

```
...
<!--this is the root of the document -->
<page-template title="simple">
<!-- define the page templates -->
  <simple-page name="simple" unit="inches">
    <size width="8.5" height="11"/>
  </simple-page>
</page-template>
...
```

1. Add a `page-template` element. Use the `title` attribute to give the page template a name. In this example, the name is `simple`.

Note: Though different elements in a template can share names, no two `page-template` elements can have the same name.

2. Define a new section by adding a nested `simple-page` element. Use the `name` attribute to give it a name (for example, `simple`), and the `unit` attribute to define what unit of measurement will be used when defining sizes and locations. In this example, we are using `inches`.

Note: Though different elements in a template can share names, no two `page` elements can have the same name.

3. Add a `size` nested element to determine the size of the page.
4. Add ending `simple-page` and `page-template` elements, in that order, to indicate that the page template is complete.

Adding Header, Footer and Body Frames

You now have a very basic page set up. Next, frames must be added to direct items in a flow in to it.

```
...
    <size width="8.5" height="11" />
    <header/>
    <footer/>
    <body top-offset="0.9" bottom-offset="0.9">
    </body>
...
```

1. Below the last nested element that defines the page (in this case, the `size` element), add a `header` element. Define its properties; in this case, it is an empty element and will use the default values.
2. Add a `footer` element and define its properties. In this case, it is an empty element and will use the default values.
3. Add a `body` element and define its properties. In this case, the `top-offset` and `bottom-offset` are set.

Note: Nested tags will be added to the `body` element in subsequent sections in this tutorial, and for that reason it should not be an empty element.

Defining the Flow

A flow must be defined in the page template in order to allow text and images to be flowed into the frames on the page.

```
...
    </body>
    <flow-section name="big-frame-page"/>
...
```

- Below the `body` element in the page template, add a `flow-section` element. In the `name` attribute, give the name of the section (or page) that the information should flow to when a new section is invoked.

Adding Text to the Flow

```
...
</page-template>

<flow>
This is a document.
</flow>
...
```

1. To add text to a flow, simply add a `flow` element below the `page-template` element.
2. Type the text that you would like to add below the `flow` element.

Note: The text should not be in the `flow` element, but between the `flow` tags.

3. Below the added text, add an ending `flow` element to indicate the end of the flowed text.

12.5.2 XML Document Shell

The following code sample is the minimum that is required to create an XML document with JClass ServerReport. At this stage, you have defined an XML document with a page, a frame, and flow.

```
<?xml version="1.0"?>
<!DOCTYPE document SYSTEM "JCServerReport.dtd">
<document>
  <!--this is the root of the document -->
  <page-template title="simple">
    <!-- define the page templates -->
    <simple-page name="simple" unit="inches">
      <size width="8.5" height="11"/>
      <header/>
      <footer/>
      <body top-offset="0.9" bottom-offset="0.9">
        </body>
        <flow-section name="big-frame-page"/>
      </simple-page>
    </page-template>

    <flow>
      This is a document.
    </flow>

  </document>
```

12.5.3 Advanced Page Template

Here you will learn how to create margins and columns.

Important: XML requires that the document tags follow a strict order for correct parsing. For example, `<border>` must be followed by `<column>`, and `<margin>` must follow `<column>`. If the order is not adhered to, the XML file cannot be parsed. For more information on adding border, please refer to [“Adding Borders”](#).

Defining Margins and Columns

```
...
  <body top-offset="0.9" bottom-offset="0.9">
    <column count="2"/>
    <margin top="0.25" left="0.1" right="0.1" bottom="0.25"/>
  ...
```

1. To create a body with multiple columns, simply add an empty `column` element, nested in the `body` element. The `count` attribute will determine the number of columns in the frame.
2. To define margins, simply add an empty `margin` element, nested in the `body` element. Next, define the width of the margin with `top`, `left`, `right`, and `bottom` attributes. These will use the same unit of measurement as the rest of the frame (which is defined in the `simple-page` element).

12.6 Creating Sections

When creating a larger document, it can be useful to create different sections.

12.6.1 Adding a New Section

```
...
</simple-page>
<simple-page name="big-frame-page" unit="inches">
  <size width="8.5" height="11"/>
  <body top-offset="0.0" bottom-offset="0.0">
    <margin top="0.25" left="1.0" right="1.0" bottom="0.25"/>
  </body>
  <flow-section name="simple"/>
</simple-page>
...
```

1. To create a new section, you need to add a new `simple-page` element, and set its unit of measurement, provide it with a distinct name, and define its nested elements and attributes.

In this case, we have added a page named `big-frame-page` and, with a `size` nested element.

2. In order to flow text into your new section, you must add a `body` element and define its nested elements and attributes.

In this case, we have defined its margins.

3. Next, define the name attribute for the `flow-section` element. This will determine the order in which elements will flow into the document.

In this case, we have set `flow-section` to `simple` to redirect the flow to the previously defined page when a section change occurs.

4. To flow text into the `big-frame-page` when a section change occurs, change the `simple` page's `flow-section` from `simple` to `big-flow-page`; otherwise, the text will continue to flow from the `simple` page to itself.
5. Add an ending `simple-page` element.

Adding Borders

```
...
<body top-offset="0.0" bottom-offset="0.0">
  <border type="single">
    <size width="8.5" height="11"/>
  </border>
</body>
...
```

- To add a border, simply add an empty `border` nested element in the frame element, and define the border type. In this case, add a `single` border around the new body frame.

12.6.2 Flowing Text into Sections

```
...
<flow>
  This is a document.
</flow>
```

```

<new-section/>
This is a new section.
<new section/>
This is back in the initial section.
...

```

1. To add text to the new section, simply add a `new-section` ending tag above the text that should be in the new section.
2. Each time you add an ending `new-section` tag, the following text in the flow will be directed to the section defined in the current page's `flow-section` element.

12.6.3 Flowing Text Into the Header

```

...
</page-template>
<header-frame page-name="simple">
Header
</header-frame>
...

```

1. To add text to the header, simply add a `header-frame` tag with a `page-name` attribute that specifies which page the header text should be written to.
2. Add the text that should be contained in the header, then add a closing `header-frame` tag.

12.6.4 Flowing Text Into the Footer

```

...
</header-frame>
<footer-frame page-name="simple">
Page <space/>
</footer-frame>
...

```

Note: In subsequent sections of this tutorial, the text in the footer will contain macros. For more information, see Section 12.9, [Using Macros](#).

1. To add text to the footer, simply add a `footer-frame` tag with a `page-name` attribute that specifies which page the footer text should be written to.
2. Add the text that should be contained in the footer, then add a closing `footer-frame` tag.

12.7 Adding and Formatting Text

In XML, JClass ServerReport uses text styles to define text formats, including the font, size, color, and other attributes. There are two steps to using formatted text: defining the text styles and applying them.

12.7.1 Defining different text styles

```

...
</page-template>

```

```

<text-styles>
  <text-style name="normal-times" font-family="Times-Roman"
    font-style="plain" point-size="12"
    paragraph-spacing="2.0"/>
  <text-style name="normal-times-right" font-family="Times-Roman"
    font-style="plain" point-size="12" align="right"
    paragraph-spacing="2.0"/>
  ...
</text-styles>
...

```

1. Below the page-template ending tag, add a text-styles tag. All text styles should be defined within the text-styles tags.
2. Add empty text-style tags for each text style that you would like to define. Each text-style requires a name attribute that is unique.
3. Define the different attributes for the text styles; for example, you can define font-family, font-style, point-size, align, and paragraph-spacing.
4. When you have finished adding the different text styles, add a closing text-styles tag.

12.7.2 Applying text styles

```

...
</page-template>
<header-frame page-name="simple">
  <use-text-style name="center-helvetica">
    Header
  </use-text-style>
</header-frame>
<footer-frame page-name="simple">
  <use-text-style name="center-helvetica">
    Page <space/>
  </use-text-style>
</footer-frame>
...

<flow>
  <current-text-style name="normal-times"/>
  This paragraph uses a 12-point TimesRoman font to render the text that it
  contains.
  <new-paragraph/>
  <current-text-style name="big-times"/>
  This paragraph uses an 18-point TimesRoman font to render the text that it
  contains.
  <new-paragraph/>
  <current-text-style name="big-times-italic"/>
  This paragraph uses an italicized 18-point TimesRoman font to render the
  text that it contains.
  <new-paragraph/><current-text-style name="normal-times"/>
  This paragraph uses a 12-point TimesRoman font and a bold tag to render the
  text that it contains.
  <new-paragraph/>
  <current-text-style name="normal-times-right"/>

```

This paragraph uses a 12-point TimesRoman font and right alignment to render the text that it contains.

...

There are two different elements that can be used to apply text styles: `use-text-style` and `current-text-style`.

Note: The empty `new-paragraph` tag causes a new paragraph to be generated in the document.

use-text-style

`use-text-style` should not be an empty element; it should surround the text that it is applied to.

1. Add an open `use-text-style` element before the text that should have this style applied to it. Define which text style to apply in the `name` attribute.
2. An ending tag should be added at the end of the text that should have this style applied to it.

current-text-style

`current-text-style` can be an empty element. It will be applied to any text that follows it in the current section, unless it has another text style applied to it.

- Add an empty `current-text-style` element before the text to which this style will be applied. Define which text style to apply in the `name` attribute.

12.7.3 Applying XML style tags

...

```
<new-paragraph/><current-text-style name="normal-times"/>  
<bold>This paragraph uses a 12-point TimesRoman font and a bold tag to  
render the text that it contains. </bold>  
<new-paragraph/>
```

...

- To incorporate further formatting, you can use other ServerReport XML tags in your document. In this case, use a `bold` tag to format the paragraph.

Note: Remember to close your XML tags; leaving tags open can result in formatting errors. For more information about available ServerReport XML tags, see Chapter 13, [XML and JClass ServerReport](#).

12.8 Adding Hyperlinks

JClass ServerReport has two different types of hyperlinks: external and internal. External links will point outside of the document, normally to a URL, whereas internal links will point to somewhere inside the document.

Note: For hyperlinks to be of any value, they must surround something that the end-user will be able to click. Therefore, the start and end tags should never be directly beside one another.

12.8.1 Creating external links

```
...
<flow>
<current-text-style name="normal-times"/>
This paragraph uses a 12-point TimesRoman font to render the text that it
contains. This
<hyperlink destination-name="http://www.quest.com"
  destination-type="external">
  link
</hyperlink>
points to the Quest web-site.
...
```

1. To add an external hyperlink, begin by adding a `hyperlink` element where you would like the link to begin. The `hyperlink` element should have two attributes: `destination-name`, which is the location, or URL, where the link points, and `destination-type`, which is `external`.
2. Add the text to be shown as a hyperlink. In this case, the word “link” will be the hyperlinked text.
3. Add an ending `hyperlink` tag to end the link.

12.8.2 Creating internal links

Creating an internal link is a two part process: first, you must define the location where the link will point, and second, you add the link.

```
...
<flow>
<current-text-style name="normal-times"/>
This paragraph uses a 12-point TimesRoman font to render the text that it
contains. This
<hyperlink destination-name="http://www.quest.com"
  destination-type="external">
  link
</hyperlink>
points to the Quest web-site. This
<hyperlink destination-name="other-section"
  destination-type="internal">
  link
</hyperlink>
points to the next section of this document.
...
<new-section/>
<current-text-style name="normal-helvetica"/>
<mark-location location-name="other-section" location-type="next"/>
This new section uses a Helvetica font.
...
```

1. To mark the location where the link will point, add a `mark-location` tag with a unique `location-name` attribute value and a valid `location-type` attribute value.

Valid location types are listed in [<mark-location> XML tag](#), in Chapter 4.

2. Add a `hyperlink` element where you would like the link to begin. The `hyperlink` element should have two attributes: `destination-name`, which is the `mark-location`'s `location-name` value, and `destination-type`, which is `internal`.
3. Add the text to be shown as a hyperlink. In this case, the word "link" will be the hyperlinked text.
4. Add an ending `hyperlink` tag to end the link.

12.9 Using Macros

JClass ServerReport comes with several macros that can be used for common functions. In this example, standard macros are applied to show the current page number and the total number of pages in the document.

12.9.1 Setting page numbers with macros

```
...
<footer-frame page-name="simple">
<use-text-style name="center-helvetica">
Page <space/><macro name="pageNum"/>
...
```

1. To call a macro, use the `macro` empty element, and provide the name of the macro in the `name` attribute.

In this case, the `pageNum` macro is used to display the page number of the current page.

12.10 Defining Draw Styles and Adding Tables

JClass Server Report allows you to define draw styles and add tables to a document. Though draw styles can be used for many purposes, they are often used in conjunction with tables as table borders. The following example will define a draw style for precisely that use.

12.10.1 Defining Draw Styles

```
...
</text-styles>

<draw-styles>
  <draw-style name="blue-line" color="blue" line-type="single"/>
</draw-styles>
...
```

1. To define a draw style, add a `draw-styles` element below the ending `text-style` tag.
2. Use `draw-style` tags to define different draw styles. Each `draw-style` tag requires its own unique name.

In this case, we have defined a `blue-line` style, which is a single blue line.

3. Use the different `draw-style` attributes to customize your draw style; for example, you can define `color` and `line-type`.
4. When you have finished defining your draw styles, add a closing `draw-styles` tag.

12.10.2 Creating a Table

Tables consist of several different elements: columns, rows, cells, and an optional header table. The number of columns is defined before content is flowed into the table, and is fixed; the number of rows is determined by the number of `<row>` elements added to the table.

Creating a table

```
...
<new-paragraph/>
<flow-table num-columns="2" border-style-name="blue-line">
  <column-info width="1.5" unit="inches"/>
  <column-info width="1.0" unit="inches"/>
...
</flow-table>
...
```

1. To create a table, simply add opening and closing `flow-table` elements, with attribute definitions in the opening element.

In this case, there are two columns (`num-columns="2"`), and the `blue-line` draw style is applied as a border style.

2. For each column in the table, a `column-info` nested element should be added immediately after the `flow-table` tag. The `column-info` tag will provide the size of each column.

In this case, all the columns use inches as their unit of measurement. The first column measures 1.5 inches and the second measures 1.0 inch.

Adding a header row

Adding a header row to a table provides you with header rows that will repeat at the top of the table each time it changes columns or pages.

```
...
  <column-info width="1.0" unit="inches"/>
  <header-table>
    <row>
      <cell text-style-name="center-helvetica">City</cell>
      <cell text-style-name="center-helvetica">Population (in
                                                millions)</cell>
    </row>
  </header-table>
...
```

1. To add a header row, add opening and closing `header-table` elements. The contents of the header row will be nested elements of `header-table`.
2. Add opening and closing `row` nested elements.

3. Add nested `cell` elements in the `row` element, and define their attributes. There should be one `cell` element for each column.

In this case, all of the header row's cells use the `center-helvetica` text style.

Adding a footer row

A footer is much like a header above.

```
<footer-table>
  <row>
    <cell text-style-name="center-helvetica">City</cell>
    <cell text-style-name="center-helvetica">Population (in
                                          millions)</cell>
  </row>
</footer-table>
```

Note: Only `JCPageTable` allows creating a footer by XML.

Adding content

Content is added to tables in row and cell nested elements. Please note that some of the cell content will be changed later on in the tutorial.

```
...
</header-table>
<row>
  <cell>Toronto</cell>
  <cell>2.2</cell>
</row>
<row>
  <cell>New York</cell>
  <cell>7.3</cell>
</row>
<row>
  <cell>Chicago</cell>
  <cell>2.8</cell>
</row>
<row>
  <cell>Vancouver</cell>
  <cell>1.8</cell>
</row>
</flow-table>
...
```

1. Add a row nested element inside the `flow-table`.

Note: If you have a header table, the row nested element should be directly below it.

2. In each row element, add a cell element for each column, followed by the cell content and an ending cell element.

Note: The column number is defined in the `flow-table`'s `num-columns` attribute.

3. Close the row element.
4. Continue adding rows and cells in this fashion until your table is complete.

12.11 Triggering an External Java Class

JClass ServerReport XML allows you to trigger external java classes to add information to the document that can not be easily specified within XML. In this example, you will be triggering the `ExternalTutorialHandler` class to populate table cells from an external data source.

Note: For this to function properly, the user object must be set on the document before `document.loadXML()` is called. In this case, use the following:

```
XMLLoadProperties xmlLoadProperties = document.getXMLLoadProperties();
xmlLoadProperties.setUserObject(new TutorialUserObject());
```

12.11.1 Using external Java code

```
...
    <row>
        <cell>Toronto</cell>
        <cell>
            <external-java-code
                class="examples.sreport.tutorial.xml.ExternalTutorialHandler">
                Toronto
            </external-java-code>
        </cell>
    </row>
    <row>
        <cell>New York</cell>
        <cell>
            <external-java-code
                class="examples.sreport.tutorial.xml.ExternalTutorialHandler">
                New York
            </external-java-code>
        </cell>
    </row>
    <row>
        <cell>Chicago</cell>
        <cell>
            <external-java-code
                class="examples.sreport.tutorial.xml.ExternalTutorialHandler">
                Chicago
            </external-java-code>
        </cell>
    </row>
    <row>
        <cell>Vancouver</cell>
```

```

        <cell>
            <external-java-code
                class="examples.sreport.tutorial.xml.ExternalTutorialHandler">
                Vancouver
            </external-java-code>
        </cell>
    </row>
    ...

```

1. In the second cell, add an `external-java-code` tag.
2. In the `external-java-code` element, reference the external class to trigger; for example:

```
class="examples.sreport.tutorial.xml.ExternalTutorialHandler"
```

3. In this example, the external class holds data, and will query the data with the `String` it is passed. Enter the `String` to query in the external class (Toronto, New York, Chicago, Vancouver).
4. Close the `external-java-code` element. Once the XML has been parsed, the data that resulted from the query will be displayed in the cell.

Remember that the user object must be set on the document before `document.loadXML()` is called.

12.11.2 ExternalTutorialHandler source code

The following is the source code for the `ExternalTutorialHandler`, referenced in Section 12.11.1, [Using external Java code](#), and the `TutorialUserObject`.

ExternalTutorialHandler

```

package examples.sreport.tutorial.xml;

import com.klg.jclass.sreport.*;

public class ExternalTutorialHandler implements ExternalCodeHandler {

    /* Default constructor. Needed for class creation via XML parser.
    */
    public ExternalTutorialHandler()
    {
    }

    /**
     * Implementation of ExternalCodeHandler interface. This method
     * makes changes to the document that could not be done inside the xml
     * itself. In this case, it gets the UserObject from the document's
     * XMLLoadProperties and uses this object to query the passed city
     * for its population.
     */
}

```

```

public void handle(JCDocument document,
                  JCFrame frame, JCTextStyle textStyle, String userData)
{
    TutorialUserObject tuo =
        (TutorialUserObject) document.getUserObject();

    double population = tuo.getDataByName(userData);
    if (population >= 0.0) {
        try {
            frame.print(textStyle, "" + population);
        } catch (EndOfFrameException eofe) {
            eofe.printStackTrace();
        }
    }
}
}

```

TutorialUserObject

```

/**
 * User Object class stored in ServerReport XMLLoadProperties for use in
 * ExternalCodeHandler implementation. In this implementation,
 * a city's population can be queried by the city's name.
 * In a real-world application, this class might be replaced with
 * a class that queries information from a database.
 */
public class TutorialUserObject {

    private String cities[] = {"Toronto", "Vancouver", "New York", "Chicago"};
    private double numbers[] = {2.2, 1.8, 7.3, 2.8};

    /* Default constructor.
    */
    public TutorialUserObject()
    {
    }

    /**
     * Look up data by name.
     */
    public double getDataByName(String name)
    {
        for (int i = 0; i < cities.length; i++) {
            if (cities[i].equals(name)) {
                return numbers[i];
            }
        }
        return -1.0;
    }
}

```

12.12 Adding Metadata and Lists

The last part of the tutorial adds metadata and lists to the document.

12.12.1 Adding Metadata

Metadata is included in the document's properties, immediately after the document element.

```
<document>
  <document-properties>
    <meta-data title="XML Tutorial 9"
      author="The JClass Team"
      subject="Adding Front Matter to a Document"
      creator="XML Tutorial Handler"
      keywords="Front Matter, Contents List,
        Table of Contents, Foreword, Lists"/>
  </document-properties>
  <!-- this is the root of the document -->
  ...
```

12.12.2 Adding Lists

This section adds ordered, unordered, and nested lists to the flow. The `list` element can only contain `list-items` or nested `list` elements. List items can be made up of text, images, or hyperlinks.

For this tutorial, we first define some symbol styles to use as bullets in the unordered lists. Symbol styles are added in the same place as text styles and draw styles.

```
...
  </draw-style>
</draw-styles>

<symbol-styles>
  <symbol-style name="dot" color="black" shape="dot" size="4"/>
  <symbol-style name="square" color="red" shape="square" size="5"/>
</symbol-styles>
...
```

We then add a series of lists to the second section of the document. Notice that in the second list, one of the symbol styles is used for `tag-symbol-style-name`. In the ordered list with a sublist, two different numbering styles are used for the number tag.

```
...
  This new section, which displays some lists, uses a Helvetica font.
  <new-paragraph/>

  Here is your basic list with default values:
  <list>
    <list-item>item 1</list-item>
    <list-item>item 2</list-item>
    <list-item>item 3</list-item>
  </list>
  <new-line/>
```

Here is a list with symbols for bullets:

```
<list type="unordered"
      bullet-style="symbol"
      tag-symbol-style-name="dot">
  <list-item>item 1</list-item>
  <list-item>item 2</list-item>
  <list-item>item 3</list-item>
</list>
<new-line/>
```

Here is your basic ordered list:

```
<list type="ordered" tag-tab="0.35">
  <list-item>item 1</list-item>
  <list-item>item 2</list-item>
  <list-item>item 3</list-item>
</list>
<new-line/>
```

Here is an ordered list, with a sublist that is also ordered:

```
<list type="ordered"
      tag-alignment="left"
      numbering-style="alpha-lower"
      tag-tab="0.35">
  <list-item>item 1</list-item>
  <list type="ordered"
        tag-alignment="center"
        tag-string=""
        numbering-style="roman-lower"
        tag-tab="0.35"
        text-tab="0.15">
    <list-item>item 1</list-item>
    <list-item>item 2</list-item>
    <list-item>item 3</list-item>
  </list>
  <list-item>item 2</list-item>
  <list-item>item 3</list-item>
</list>
<new-line/>
```

You can also mix list types with nested lists:

```
<list type="unordered"
      bullet-style="symbol"
      tag-symbol-style-name="square"
      tag-tab="0.30">
  <list-item>item 1</list-item>
  <list-item>item 2</list-item>
  <list-item>item 3</list-item>
  <list type="ordered"
        numbering-style="alpha-upper"
        tag-string=""
        tag-tab="0.35">
    <list-item>item 1</list-item>
    <list-item>item 2</list-item>
  </list type="unordered">
```

```

tag-string="==">
tag-text-style-name="normal-times">
<list-item>item 1</list-item>
<list-item>item 2</list-item>
<list-item>item 3</list-item>
</list>
<list-item>item 3</list-item>
</list>
</list>
...

```

Figure 57 shows how the lists look in the PDF document.

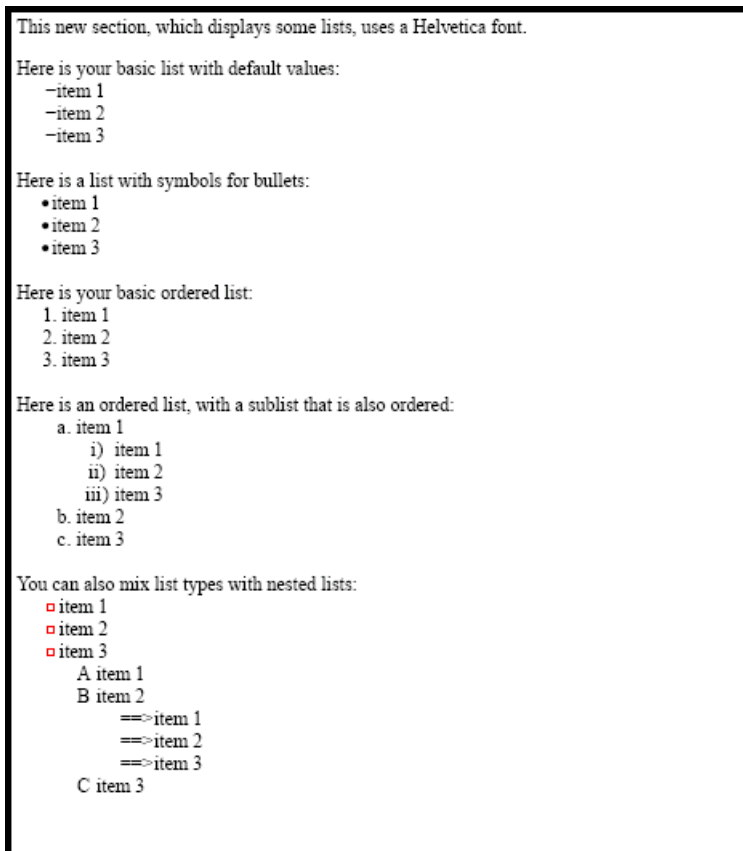


Figure 57 Lists

12.13 Adding Front Matter

Front matter can include content such as the title page, table of contents, other reference lists, acknowledgements, forward, and preface.

To create a front matter flow, take the following steps:

1. Define the page templates.
2. Start the front matter, specifying a page template to use.
3. Start the flow.
4. Flow your content.
5. End the flow.
6. End the front matter.

In this tutorial, we will create three pages in the front matter flow: a title page, table of contents, and a forward.

12.13.1 Defining the Page Templates for the Front Matter

We start by defining two page templates for the front matter. The front matter page templates are defined immediately after the body flow page templates. Notice that the `front-matter` title page flows to the `text` page when either a new page or a new section begins, and the `text` page flows only to itself.

```
...
    </simple-page>
  </page-template>

  <page-template title="front-matter">
    <!-- define the page templates -->
    <page name="title" unit="inches" first="true">
      <location x="0" y="0"/>
      <size width="8.5" height="11"/>
      <frame name="titleFrame" unit="inches">
        <location x="2.5" y="4.0"/>
        <size width="3.5" height="3.0"/>
      </frame>
      <flow-frame name="titleFrame"/>
      <flow-page name="text"/>
      <flow-section name="text"/>
    </page>
    <page name="text" unit="inches">
      <location x="0" y="0"/>
      <size width="8.5" height="11"/>
      <frame name="body" unit="inches">
        <location x="1.0" y="1.0"/>
        <size width="6.5" height="9.0"/>
      </frame>
      <frame name="footer" unit="inches">
        <location x="1.0" y="10.25"/>
```



```

        <size width="6.5" height="0.5"/>
    </frame>
    <flow-frame name="body"/>
    <flow-page name="text"/>
    <flow-section name="text"/>
</page>
</page-template>
...

```

12.13.2 Starting and Ending the Front Matter Flow

Once we have page templates defined, we can use them in a front matter flow. We set the `front-matter` element first and then start a new flow. After flowing the content (described later), we end the flow and the front-matter element.

Note: The body flow must end before the front matter flow begins.

```

...
</flow>
<!-- Body flow is ended, we can now start the front matter flow. >
<front-matter template-list-name="front-matter">
    <flow>
        <!-- Flow front matter content>
    </flow>
</front-matter>
...

```

The following sections show how to flow a title page, table of contents, and a foreword in the front matter flow.

12.13.3 Adding a Title Page

The text for the title page is flowed using the `front-matter` page template.

```

...
<front-matter template-list-name="front-matter">
    <flow>
        <current-text-style name="center-title"/>
        XML Tutorial 9<new-line/>
        by<new-line/>
        The JClass Team
        ...
    </flow>
</front-matter>
...

```

1. Set a text style if desired.
2. Flow the title text.

12.13.4 Adding a Table of Contents

After we flow the title page, we flow a table of contents. Figure 58 shows what the table of contents looks like in the PDF document.

Table of Contents		
A	First section	1
A.1	First paragraph	1
A.2	Second paragraph	1
A.3	Third paragraph	1
A.4	Fourth paragraph	1
A.5	Fifth paragraph	1
A.6	The table	1
B	New section	2
B.1	Basic Unordered List	2
B.2	Unordered List with Symbols	2
B.3	Basic Ordered List	2
B.4	Nested List	2
B.5	Mixed Nested List	2
C	Back to initial page type	3

Figure 58 Table of contents

Notice that there are three sections: A, B and C. The entries within each section are numbered using the section letter followed by a period and a number. The text for each entry is a hyperlink.

To be able to output a table of contents, you need to take the following steps:

1. Define the text styles to use in the table of contents.
2. Define a contents list to hold the table of contents.
3. Mark the locations of the sections, headings, or other content that you want included in the table of contents, and then create content entries that refer to those marked locations.
4. Output the contents list

The following sections describe these steps.

Defining Text Styles for the Table of Contents

We define text styles for the table of contents in the same place we define all other text styles.

```
...  
  <text-styles>  
  ...
```

```

        <text-style name="toc-style-0" font-family="Times-Roman"
            font-style="plain" point-size="14" indent-left="0.5">
            <tab-stop position="1.0"/>
            <tab-stop position="5.0"/>
        </text-style>
        <text-style name="toc-style-1" font-family="Times-Roman"
            font-style="plain" point-size="12" indent-left="1.0">
            <tab-stop position="1.5"/>
            <tab-stop position="5.0"/>
        </text-style>
    </text-styles>
    ...

```

Defining the Contents List

Before we begin the flow, we define the “Table of Contents” contents list. The `contents-list` element is added after the page templates and style definitions but before the frame definitions.

```

    ...
    </text-styles>

    <contents-list name="Table of Contents"
        flow-page-template="text">
        <default-text-style name="toc-style-0" level="0"/>
        <default-text-style name="toc-style-1" level="1"/>
    </contents-list>

    <header-frame page-template-name="simple" page-name="simple">
    ...

```

Marking Locations and Creating Entries

For each piece of content that you want to include in the table of contents, set a `mark-location` element before the content itself. In the following example, a paragraph is marked with the name “second-paragraph”. Then the `contents-list-entry` details are specified. Here, the text “Second Paragraph” is added as a level 1 heading to the Table of Contents. The entry is numbered using the parent autonumbering style, which takes the number of the parent plus a number, which for this particular example results in A.2 (A for the section the entry belongs to and 2 because this is the second entry). In this case, the entry is treated as a hyperlink.

```

    ...
    <flow>
    ...
    <new-paragraph/>
    <current-text-style name="big-times"/>
    <mark-location location-name="second-paragraph"
        location-type="next"/>
    <contents-list-entry contents-list-name="Table of Contents"
        level="1"
        text="Second paragraph"
        auto-numbering="parent"
        location-name="second-paragraph"
        hyperlink-used="true"/>
    This paragraph uses an 18-point TimesRoman font to render the
    ...

```

If you want to see all the locations used in the tutorial's table of contents, see Section 12.14, [The Completed XML Document Code](#).

Outputting the Contents List

To output the contents list, you use the `output-contents-list` element within the front matter flow.

Note: You cannot add contents lists to the body flow. For more information, see Chapter 8, [Flowing Content into Your Document](#).

```
...
  <front-matter template-list-name="front-matter">
    <flow>
      ...
      <output-contents-list name="Table of Contents"/>
      ...
    </flow>
  </front-matter>
...
```

12.13.5 Adding a Foreword

The foreword, or any text/image based content, can be add to the front matter flow. In this case, the foreword follows the table of contents and starts on a new page.

```
...
  <front-matter template-list-name="front-matter">
    <flow>
      ...
      <new-page/>
      Foreword<new-line/>
      <current-text-style name="normal-times"/>
      This foreword is more than four words long. Notice that
      this page is part of the front matter whose pages are numbered
      separately from the document.
    </flow>
  </front-matter>
...
```

12.14 The Completed XML Document Code

The following is the completed code for the tutorial example. If your own XML file is not working properly, please check it against the following to ensure that your code is correct.

To see the XML output at any of the stages leading up to this point, you can access the tutorial at different stages by running it. For more information about running the tutorial, see [Running the Tutorial](#) in Section 12.1, [Introduction](#).

```
<?xml version="1.0"?>
<!DOCTYPE document SYSTEM "JCServerReport.dtd">
<document>
  <document-properties>
    <meta-data title="XML Tutorial 9"
      author="The JClass Team"
      subject="Adding Front Matter to a Document"
      creator="XML Tutorial Handler"
      keywords="Front Matter, Contents List,
        Table of Contents, Foreword, Lists"/>
  </document-properties>
  <!-- this is the root of the document -->
  <page-template title="simple">
    <!-- define the page templates -->
    <simple-page name="simple" unit="inches">
      <size width="8.5" height="11"/>
      <margin top="0.25" bottom="0.25" left="1.0" right="1.0"/>
      <header/>
      <footer/>
      <body top-offset="0.9" bottom-offset="0.9">
        <column count="2"/>
        <margin top="0.1" left="0.1" right="0.1" bottom="0.1"/>
      </body>
      <flow-section name="big-frame-page"/>
    </simple-page>
    <simple-page name="big-frame-page" unit="inches">
      <size width="8.5" height="11"/>
      <margin top="0.25" bottom="0.25" left="1.0" right="1.0"/>
      <body top-offset="0.0" bottom-offset="0.0">
        <border type="single"/>
        <margin top="0.1" left="0.1" right="0.1" bottom="0.1"/>
      </body>
      <flow-section name="simple"/>
    </simple-page>
  </page-template>

  <page-template title="front-matter">
    <!-- define the page templates -->
    <page name="title" unit="inches" first="true">
      <location x="0" y="0"/>
      <size width="8.5" height="11"/>
      <frame name="titleFrame" unit="inches">
        <location x="2.5" y="4.0"/>
        <size width="3.5" height="3.0"/>
      </frame>
    </page>
  </page-template>
</document>
```

```

        <flow-frame name="titleFrame"/>
        <flow-page name="text"/>
        <flow-section name="text"/>
    </page>
    <page name="text" unit="inches">
        <location x="0" y="0"/>
        <size width="8.5" height="11"/>
        <frame name="body" unit="inches">
            <location x="1.0" y="1.0"/>
            <size width="6.5" height="9.0"/>
        </frame>
        <frame name="footer" unit="inches">
            <location x="1.0" y="10.25"/>
            <size width="6.5" height="0.5"/>
        </frame>
        <flow-frame name="body"/>
        <flow-page name="text"/>
        <flow-section name="text"/>
    </page>
</page-template>

<text-styles>
    <text-style name="normal-times" font-family="Times-Roman"
        font-style="plain" point-size="12"
        paragraph-spacing="2.0"/>
    <text-style name="normal-times-right" font-family="Times-Roman"
        font-style="plain" point-size="12" align="right"
        paragraph-spacing="2.0"/>
    <text-style name="big-times" font-family="Times-Roman"
        font-style="plain" point-size="18"
        paragraph-spacing="2.0"/>
    <text-style name="big-times-italic" font-family="Times-Roman"
        font-style="italic" point-size="18"
        paragraph-spacing="2.0"/>
    <text-style name="normal-helvetica" font-family="Helvetica"
        font-style="plain" point-size="12"
        paragraph-spacing="2.0"/>
    <text-style name="center-helvetica" font-family="Helvetica"
        font-style="plain" point-size="12" align="center"
        paragraph-spacing="2.0"/>
    <text-style name="center-title" font-family="Times-Roman"
        font-style="plain" point-size="14" align="center"
        line-spacing="1.8"/>
    <text-style name="toc-style-0" font-family="Times-Roman"
        font-style="plain" point-size="14" indent-left="0.5">
        <tab-stop position="1.0"/>
        <tab-stop position="5.0"/>
    </text-style>
    <text-style name="toc-style-1" font-family="Times-Roman"
        font-style="plain" point-size="12" indent-left="1.0">
        <tab-stop position="1.5"/>
        <tab-stop position="5.0"/>
    </text-style>
</text-styles>

```

```

<draw-styles>
  <draw-style name="blue-line" color="blue" line-type="single"/>
</draw-styles>

<symbol-styles>
  <symbol-style name="dot" color="black" shape="dot" size="4"/>
  <symbol-style name="square" color="red" shape="square" size="5"/>
</symbol-styles>
<contents-list name="Table of Contents"
  flow-page-template="text">
  <default-text-style name="toc-style-0" level="0"/>
  <default-text-style name="toc-style-1" level="1"/>
</contents-list>

<header-frame page-template-name="simple" page-name="simple">
  <use-text-style name="center-helvetica">
    Header
  </use-text-style>
</header-frame>

<footer-frame page-template-name="simple" page-name="simple">
  <use-text-style name="center-helvetica">
    Page
    <space/>
    <macro name="pageNum"/>
  </use-text-style>
</footer-frame>

<doc-frame page-template-name="front-matter" page-name="text"
  name="footer">
  <use-text-style name="center-helvetica">
    <macro name="romanPageNum"/>
  </use-text-style>
</doc-frame>

<flow>
  <current-text-style name="normal-times"/>
  <mark-location location-name="first-section"
    location-type="current"/>
  <contents-list-entry contents-list-name="Table of Contents"
    level="0"
    tag="A"
    text="First section"
    location-name="first-section"
    hyperlink-used="true"/>
  <mark-location location-name="first-paragraph"
    location-type="next"/>
  <contents-list-entry contents-list-name="Table of Contents"
    level="1"
    text="First paragraph"
    auto-numbering="parent"
    location-name="first-paragraph"
    hyperlink-used="true"/>

```

```

This paragraph uses a 12-point TimesRoman font to render the
text that it contains. This
<hyperlink destination-name="http://www.quest.com"
            destination-type="external">
    link
</hyperlink>
points to the Quest web-site. This
<hyperlink destination-name="other-section"
            destination-type="internal">
    link
</hyperlink>
points to the next section of this document.
<new-paragraph/>
<current-text-style name="big-times"/>
<mark-location location-name="second-paragraph"
                location-type="next"/>
<contents-list-entry contents-list-name="Table of Contents"
                    level="1"
                    text="Second paragraph"
                    auto-numbering="parent"
                    location-name="second-paragraph"
                    hyperlink-used="true"/>
This paragraph uses an 18-point TimesRoman font to render the
text that it contains.
<new-paragraph/>
<current-text-style name="big-times-italic"/>
<mark-location location-name="third-paragraph"
                location-type="next"/>
<contents-list-entry contents-list-name="Table of Contents"
                    level="1"
                    text="Third paragraph"
                    auto-numbering="parent"
                    location-name="third-paragraph"
                    hyperlink-used="true"/>
This paragraph uses an italicized 18-point TimesRoman font to render
the text that it contains.
<new-paragraph/>
<current-text-style name="normal-times"/>
<mark-location location-name="fourth-paragraph"
                location-type="next"/>
<contents-list-entry contents-list-name="Table of Contents"
                    level="1"
                    text="Fourth paragraph"
                    auto-numbering="parent"
                    location-name="fourth-paragraph"
                    hyperlink-used="true"/>
<bold>This paragraph uses a 12-point TimesRoman font and a bold tag
to render the text that it contains
</bold>
<new-paragraph/>
<current-text-style name="normal-times-right"/>
<mark-location location-name="fifth-paragraph"
                location-type="next"/>

```



```

<contents-list-entry contents-list-name="Table of Contents"
    level="1"
    text="Fifth paragraph"
    auto-numbering="parent"
    location-name="fifth-paragraph"
    hyperlink-used="true"/>

```

This paragraph uses a 12-point TimesRoman font and right alignment to render the text that it contains.

```

<new-paragraph/>

```

```

<mark-location location-name="flow-table" location-type="next"/>

```

```

<contents-list-entry contents-list-name="Table of Contents"
    level="1"
    text="The table"
    auto-numbering="parent"
    location-name="flow-table"
    hyperlink-used="true"/>

```

```

<flow-table num-columns="2" border-style-name="blue-line">

```

```

  <column-info width="1.5" unit="inches"/>

```

```

  <column-info width="1.0" unit="inches"/>

```

```

  <header-table>

```

```

    <row>

```

```

      <cell text-style-name="center-helvetica">City</cell>

```

```

      <cell text-style-name="center-helvetica">Population (in
        millions)

```

```

      </cell>

```

```

    </row>

```

```

  </header-table>

```

```

  <row>

```

```

    <cell>Toronto</cell>

```

```

    <cell>2.2</cell>

```

```

  </row>

```

```

  <row>

```

```

    <cell>New York</cell>

```

```

    <cell>7.3</cell>

```

```

  </row>

```

```

  <row>

```

```

    <cell>Chicago</cell>

```

```

    <cell>2.8</cell>

```

```

  </row>

```

```

  <row>

```

```

    <cell>Vancouver</cell>

```

```

    <cell>1.8</cell>

```

```

  </row>

```

```

</flow-table>

```

```

<new-section/>

```

```

<current-text-style name="normal-helvetica"/>

```

```

<mark-location location-name="new-section" location-type="next"/>

```

```

<contents-list-entry contents-list-name="Table of Contents"
    level="0"
    tag="B"

```

```

    text="New section"

```

```

    location-name="new-section"

```

```

    hyperlink-used="true"/>

```

This new section, which displays some lists, uses a Helvetica font.

```

<mark-location location-name="first-list" location-type="next"/>
<contents-list-entry contents-list-name="Table of Contents"
    level="1"
    text="Basic Unordered List"
    location-name="first-list"
    auto-numbering="parent"
    hyperlink-used="true"/>

<new-paragraph/>
Here is your basic list with default values:
<list>
    <list-item>item 1</list-item>
    <list-item>item 2</list-item>
    <list-item>item 3</list-item>
</list>

<mark-location location-name="second-list" location-type="next"/>
<contents-list-entry contents-list-name="Table of Contents"
    level="1"
    text="Unordered List with Symbols"
    location-name="second-list"
    auto-numbering="parent"
    hyperlink-used="true"/>

<new-line/>
Here is a list with symbols for bullets:
<list type="unordered"
    bullet-style="symbol"
    tag-symbol-style-name="dot">
    <list-item>item 1</list-item>
    <list-item>item 2</list-item>
    <list-item>item 3</list-item>
</list>

<mark-location location-name="third-list" location-type="next"/>
<contents-list-entry contents-list-name="Table of Contents"
    level="1"
    text="Basic Ordered List"
    location-name="third-list"
    auto-numbering="parent"
    hyperlink-used="true"/>

<new-line/>
Here is your basic ordered list:
<list type="ordered" tag-tab="0.35">
    <list-item>item 1</list-item>
    <list-item>item 2</list-item>
    <list-item>item 3</list-item>
</list>

<mark-location location-name="fourth-list" location-type="next"/>
<contents-list-entry contents-list-name="Table of Contents"
    level="1"
    text="Nested List"
    location-name="fourth-list"
    auto-numbering="parent"
    hyperlink-used="true"/>

<new-line/>
Here is an ordered list, with a sublist that is also ordered:

```

```

<list type="ordered"
      tag-alignment="left"
      numbering-style="alpha-lower"
      tag-tab="0.35">
  <list-item>item 1</list-item>
  <list type="ordered"
        tag-alignment="center"
        tag-string=""
        numbering-style="roman-lower"
        tag-tab="0.35"
        text-tab="0.15">
    <list-item>item 1</list-item>
    <list-item>item 2</list-item>
    <list-item>item 3</list-item>
  </list>
  <list-item>item 2</list-item>
  <list-item>item 3</list-item>
</list>

<mark-location location-name="fifth-list" location-type="next"/>
<contents-list-entry contents-list-name="Table of Contents"
                    level="1"
                    text="Mixed Nested List"
                    location-name="fifth-list"
                    auto-numbering="parent"
                    hyperlink-used="true"/>

<new-line/>
You can also mix list types with nested lists:
<list type="unordered"
      bullet-style="symbol"
      tag-symbol-style-name="square"
      tag-tab="0.30">
  <list-item>item 1</list-item>
  <list-item>item 2</list-item>
  <list-item>item 3</list-item>
  <list type="ordered"
        numbering-style="alpha-upper"
        tag-string=""
        tag-tab="0.35">
    <list-item>item 1</list-item>
    <list-item>item 2</list-item>
    <list type="unordered"
          tag-string="=>"
          tag-text-style-name="normal-times">
      <list-item>item 1</list-item>
      <list-item>item 2</list-item>
      <list-item>item 3</list-item>
    </list>
    <list-item>item 3</list-item>
  </list>
</list>

<new-section/>
<current-text-style name="normal-helvetica"/>
<mark-location location-name="back-in-initial-page-type"
              location-type="next"/>

```

```

        <contents-list-entry contents-list-name="Table of Contents"
            level="0"
            tag="C"
            text="Back to initial page type"
            location-name="back-in-initial-page-type"
            hyperlink-used="true"/>
        <current-text-style name="normal-times"/>
        This is a new section which switched back to the initial
        page type and uses a TimesRoman font.
    </flow>

    <front-matter template-list-name="front-matter">
        <flow>
            <current-text-style name="center-title"/>
            XML Tutorial 9<new-line/>
            by<new-line/>
            The JClass Team
            <output-contents-list name="Table of Contents"/>
            <new-page/>
            Foreword<new-line/>
            <current-text-style name="normal-times"/>
            This foreword is more than four words long. Notice that
            this page is part of the front matter whose pages are numbered
            separately from the document.
        </flow>
    </front-matter>

</document>

```

XML and JClass ServerReport

[Specifying Other XML Files](#) ■ [Using the external-java-code Tag](#)
[Using JClass ServerReport to Convert XML to PDF or RTF](#)
[Using JClass ServerReport to Convert from XML to HTML](#)

DTD files included with JClass ServerReport that are relevant to XML output are outlined in the table below. All of these DTDs are included in the JClass ServerReport JAR file (*jcsreport.jar*), located in the *JCLASS_SERVER_HOME/lib/* directory. The DTDs are installed into the *JCLASS_SERVER_HOME/xml-dtd/* directory for your reference.

DTD filename	Description
<i>JCServerReport.dtd</i>	The main consolidating DTD for JClass ServerReport; contains all other JClass ServerReport DTDs
<i>ServerReportFlow.dtd</i>	Describes the <code>flow</code> element
<i>ServerReportDocFrame.dtd</i>	Describes the <code>doc-frame</code> element
<i>ServerReportPageTemplate.dtd</i>	Outlines the structure of JClass ServerReport page templates
<i>ServerReportDrawStyle.dtd</i>	Describes the template drawing elements for JClass ServerReport
<i>ServerReportSymbolStyle.dtd</i>	Describes the template symbol style objects for JClass ServerReport
<i>ServerReportTableStyle.dtd</i>	Describes the template table style objects for JClass ServerReport
<i>ServerReportTextStyle.dtd</i>	Specifies the template text style elements for JClass ServerReport

Note: For a full list of the DTD tags, see [JClass ServerReport DTD Tags](#).

13.1 Specifying Other XML Files

In addition to specifying an entire document in XML, you can also specify these elements in an XML file for use in Java applications.

- `page-template`
- `text-styles`
- `draw-styles`
- `symbol-styles`
- `table-styles`

Note that you do not need to use a file; any object that can be made into an `InputStream` can be used. An `InputStream` can be created, for example, by passing an instance of any `java.io.InputStream` or `java.io.Reader` into the `InputStream` constructor. A `String`, for example, can be used via a `StringReader`.

Document Styles

If styles are saved to a file without flow or content information, the file is referred to as a document style file. Document style files do not contain start or ending `<document>` tags, and are loaded using the `JCDocument.loadDocumentStyles(InputStream source)` and `JCDocument.loadDocumentStyles(InputStream stream)` methods. A document styles file can contain `page-template`, `text-styles`, `draw-styles`, `table-styles`, or any combination thereof. The effect of loading the file is the same as the effect of loading individual file types, as described in Section 13.1.1, [Applying a page-template From XML](#).

13.1.1 Applying a page-template From XML

You can specify a `page-template` in an XML file for use in your Java applications. This means that if you have written your own XML `page-template`, you can apply it to your document by loading it as either an internal `String` or an external XML file. There are many `JCPageTemplate` methods you can use to apply templates.

Servlet Example

`JClass ServerReport` provides a servlet example – *PageTemplateServlet.java* – that demonstrates loading `page-templates` from an XML file and including these in a document (found in `JCLASS_SERVER_HOME/examples/sreport/xml/`).

Code Example

This code example depicts using `page-templates` from an XML file in a Java application. As noted, you do not need to use a file; any object that can be made into an `InputStream` can be used. A `String`, for example, can be used. Consider:

```
String template = "<?xml version=\"1.0\"?>" +
"<!DOCTYPE page-template SYSTEM \"ServerReportPageTemplate.dtd\">\n" +
"<page-template title=\"8p5x11\">\n" +
"  <page name=\"8p5x11\" unit=\"inches\"> +
"    <location x=\"0\" y=\"0\"/>" +
```

```

"        <size width=\"8.5\" height=\"11\"/>" +
"        <frame name=\"header\" unit=\"inches\">" +
"            <location x=\"1\" y=\"0.25\"/>" +
"            <size width=\"6.5\" height=\"0.75\"/>" +
"        </frame>" +
"        <frame name=\"body\" unit=\"inches\">" +
"            <location x=\"1\" y=\"1\"/>" +
"            <size width=\"6.5\" height=\"9\"/>" +
"            <column count=\"2\"/>" +
"        </frame>" +
"        <frame name=\"footer\" unit=\"inches\">" +
"            <location x=\"1\" y=\"10.25\"/>" +
"            <size width=\"6.5\" height=\"0.75\"/>" +
"        </frame>" +
"        <flow-frame name=\"body\"/>" +
"        <flow-page name=\"8p5x11\"/>" +
"        <flow-section name=\"8p5x11\"/>" +
"    </page>" +
"</page-template>\n";

```

The String then can be used with the following call:

```

JCPageTemplate.loadTemplates(new InputSource(new
    StringReader(template)));

```

13.1.2 Applying text-styles From XML

You can specify text-styles in an XML file for use in Java applications.

The `JCTextStyle` class specifies all attributes controlling the appearance of text in `JClass ServerReport`. The `loadTextStyles()` methods are core methods in `JCTextStyle` and take as parameters either an `InputStream` or an `InputSource` that refers to XML data.

The following steps provide an overview of how to specify `JCTextStyles` in XML form.

1. Create an XML file containing your text-style definitions.
2. Place this XML file in a location accessible to your application.
3. Call `JCTextStyle.loadTextStyles()`.

Servlet Example

`JClass ServerReport` provides a servlet example – *TextStyleServlet.java* – that demonstrates loading text-styles from an XML file and including these text-styles in a document (found in `JCLASS_SERVER_HOME/examples/sreport/xml/`) and in the WAR file.

Code Example

The following *text.xml* file can be included in your Java application:

```

<!-- text.xml -->
<?xml version="1.0"?>
<!DOCTYPE text-styles SYSTEM "ServerReportTextStyle.dtd">
<text-styles>
    <text-style name="times" font-family="Times-roman">
        <tab-stop position="0.5" />
        <tab-stop position="1.0" />
    </text-style>

```

```

        </text-style>
        <text-style name="times-red" color="red"
            font-family="Times-roman">
            <tab-stop position="4.0" fill="underline" />
        </text-style>
        <text-style name="big" point-size="14" align="center" />
        <text-style name="big-left" point-size="14" align="left" />
        <text-style name="big-scary" point-size="16" align="center"
            font-style="bold" />
    </text-styles>

```

In order to use these text-styles, include the following code:

```

JCTextStyle.loadTextStyles(
    new java.io.FileInputStream(new java.io.File("text.xml")));

```

13.1.3 Applying draw-styles From XML

You can specify draw-styles in an XML file for use in your Java applications.

The `JCDrawStyle` class creates styles for drawn objects, and specifies line and fill attributes (for example, color and width). The `loadDrawStyles()` methods are core methods in `JCDrawStyle` and take as parameters either an `InputStream` or an `InputSource` that refers to XML data.

To specify `JCDrawStyles` in XML form, follow these steps.

1. Create an XML file containing your draw-styles.
2. Place this XML file in a location accessible to your application.
3. Call `JCDrawStyle.loadDrawStyles()`.

Servlet Example

`JClass ServerReport` provides a servlet example – *DrawStyleServlet.java* – that demonstrates loading draw-styles from an XML file and including these draw-styles in a document (found in `JCLASS_SERVER_HOME/examples/sreport/xml/`).

Code Example

The following *draw.xml* file can be included in your Java application:

```

<!-- draw.xml -->
<?xml version="1.0"?>
<!DOCTYPE draw-styles SYSTEM "ServerReportDrawStyle.dtd">
<draw-styles>
    <draw-style name="tacky" color="red" line-type="double" />
    <draw-style name="restrained" color="black" />
</draw-styles>

```

You then can load *draw.xml* by calling:

```

JCDrawStyle.loadDrawStyles(
    new java.io.FileInputStream(new java.io.File("draw.xml")));

```


13.1.4 Applying symbol-styles From XML

You can specify symbol-styles in an XML file for use in your Java applications.

The `JCSymbolStyle` class creates bullets for use in unordered lists, specifying shape, color, and size attributes. The `loadSymbolStyles()` methods take as parameters either an `InputStream` or an `InputSource` that refers to XML data.

To specify `JCSymbolStyles` in XML form, follow these steps.

1. Create an XML file containing your symbol-styles.
2. Place this XML file in a location accessible to your application.
3. Call `JCSymbolStyle.loadDrawStyles()`.

Servlet Example

JClass ServerReport provides a servlet example – *SymbolStyleServlet.java* – that demonstrates loading symbol-styles from an XML file and including these symbol-styles in a document (found in *JCLASS_SERVER_HOME/examples/sreport/xml/*).

Code Example

The following *symbol.xml* file can be included in your Java application:

```
<!-- symbol.xml -->
<?xml version="1.0"?>
<!DOCTYPE symbol-styles SYSTEM "ServerReportSymbolStyle.dtd">
<symbol-styles>
    <symbol-style name="dot" color="black" shape="dot" size="4"/>
    <symbol-style name="square" color="red" shape="square" size="5"/>
</symbol-styles>
```

You can then load *symbol.xml* by calling:

```
JCSymbolStyle.loadSymbolStyles(
    new java.io.FileInputStream(new java.io.File("symbol.xml")));
```

13.1.5 Applying table-styles From XML

You can specify table-styles in an XML file for use in your Java applications. The `JCTableStyle` class creates styles used to draw `JCPageTable` and `JCFlowTable` objects. The `loadTableStyles()` methods are core methods in `JCTableStyle` and take as parameters either an `InputStream` or `InputSource` that refers to XML data.

To specify `JCTableStyles` in XML form, follow these steps.

1. Create an XML file containing your table-styles.
2. Place this XML file in a location accessible to your application.
3. Call `JCTableStyle.loadTableStyles()`.

Note: Because `JCTableStyle` objects refer to `JCDrawStyle` objects, you must either use stock draw styles in your table style definition (please see [Stock Draw Styles](#)) or define your own

draw styles before loading your table styles. If you choose to define draw styles, you may do so in the code with `JCDrawStyle` classes or you may load them from an XML file.

Servlet Example

`JClass ServerReport` provides a servlet example – *TableStyleServlet.java* – that demonstrates loading table-styles from an XML file and including these in a document (found in *JCLASS_SERVER_HOME/examples/sreport/xml/*).

Code Example

The following *table.xml* file using stock draw styles can be included in your Java application:

```
<?xml version="1.0"?>
<!DOCTYPE table-styles SYSTEM "ServerReportTableStyle.dtd">
<table-styles>
  <table-style name="dashed-table"
    top-border="default line"
    bottom-border="default line"
    left-border="default line"
    right-border="default line"
    row-border="dashed line"
    header-border="default 2pt line"
    column-border="dashed line"
    show-first-row-top-border="true"
    first-column-special="false"
    background="white">
  </table-style>
</table-styles>
```

You then can load *table.xml* by calling:

```
JCTableStyle.loadTableStyles(
    new java.io.FileInputStream(new java.io.File("table.xml")));
```

13.1.6 Applying Scope to XML Styles

For styles that have been created in XML separate from the document, or that have been created through XML that is linked to the document (either through the document's `loadXML()` or `loadDocumentStyle()` methods), simply set the `StyleScope` property for the `XMLLoadProperties` class.

If styles that have been created through XML that is linked to the document have a null value of `StyleScope`, this implies that the style has document-level scope.

13.2 Using the external-java-code Tag

The `external-java-code` tag permits a Java class to be executed during XML file parsing. The specified class can be passed a String of parameters.

For instance, the following XML snippet calls a class (`Foo.class`) that implements the `ExternalCodeHandler` interface.

```
<external-java-code class="Foo">
    cars, 2, 5
</external-java-code>
```

For detailed information, please see the External Java Servlet example, automatically installed in *JCLASS_SERVER_HOME/examples/sreport/xml/* or in the WAR file.

13.2.1 ExternalCodeHandler Interface

The `ExternalCodeHandler` interface must be implemented in order to use the `external-java-code` tag in JClass ServerReport XML input. You will need to create a class that implements this interface. This class must have an empty constructor, that is, a constructor that takes no arguments, so that an instance can be created by the XML parser.

The `ExternalCodeHandler` interface has one method: `void handle(JCDocument doc, JCFrame frame, JCTextStyle textStyle, String userData)`.

JClass ServerReport will supply the document, current frame if any, current text style, and the data between the beginning and end tags. You will need to parse the data String, if required, and perform any other handling you wish.

To facilitate communication between your servlet or application and your external code handler class, an object may be set in the `UserObject` property of the `JCDocument` class. This user-specified object may be retrieved inside your external code handler by calling `JCDocument.getUserObject()`.

13.3 Using JClass ServerReport to Convert XML to PDF or RTF

To get you started, JClass ServerReport includes example servlets (found in *JCLASS_SERVER_HOME/examples/sreport/xml/*; also included in the samples WAR file). These servlets return the set XML source in PDF or RTF format.

For instance, to demonstrate loading an entire document from an XML file, please see the *DocumentServlet.java* servlet example, found in *JCLASS_SERVER_HOME/examples/sreport/xml/*. Another example is found in the command line example *XMLData.java*, found in *JCLASS_SERVER_HOME/examples/sreport/main/*.

13.3.1 Overview of XML Parsing

The following graphic provides an overview of the steps required to convert your JClass ServerReport document XML into PDF or RTF output.



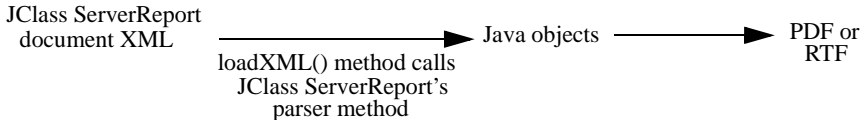
13.3.2 Requirements of Your XML Parser

In order to use JClass ServerReport to convert XML, you will need an XML parser that is JAXP 1.1-compliant and SAX Level 2-compatible. For your convenience, JClass ServerReport includes both *jaxp.jar* and *crimson.jar* in the *JCLASS_SERVER_HOME/lib/* directory. You may substitute for *crimson.jar* any parser that is compliant with Sun's JAXP 1.1 specification. See [Sun's JAXP documentation](#) for more information.

13.3.3 Creating a Document with XML

The `JCDocument` class contains a list of pages that are governed by a flow. For XML parsing, the `JCDocument` class contains the `loadXML()` method, which calls the parser methods to delineate/parse the XML, create the appropriate objects, and write to the current document. You are able to define all document formatting and data in one XML stream.

This means that the basic graphic in Section 13.3.1, [Overview of XML Parsing](#) can be refined to:



Note: If you are loading `JCTableStyles`, you must use stock draw styles or define them yourself. If you choose to define them, you may do so in the code with `JCDrawStyle` classes or load from an XML file (they must be loaded before the `JCTableStyles`).

Stock Draw Styles

The stock draw styles in `JCDrawStyle` comprise the following (please review [JCDrawStyle](#) in the [API](#) for full details):

Style	Name property	Description
BLANK	default blank	no line
LINE	default line	thin black lines
LINE_1POINT	default 1pt line	medium (1 point) black lines
LINE_2POINT	default 2pt line	thick (2 point) black lines

Style	Name property	Description
LINE_DASHED	dashed line	thin dashed line
LINE_DOUBLE	default double line	thin black double lines

13.3.4 Loading Document XML Files

`loadXML()` is a convenience method in `JCDocument` that loads XML data that conforms to the *JCServerReport.dtd* specification. This method can take as its parameters an `InputStream` that contains the XML information or as an `InputSource`, which can be used to read XML data from a variety of sources including a `String`.

The following example shows how to load XML data from an `InputStream`.

```
XMLLoadProperties loadProperties = new XMLLoadProperties();
loadProperties.setResolvingServletContext(servletContext);

JCDocument document = null;
try {
    document = new JCDocument(stream);
    document.setXMLLoadProperties(loadProperties);
    document.loadXML(
        new FileInputStream(new File("data.xml")));
}
catch (Exception e) {
    System.err.println("Error loading document data = " + e);
    System.exit(1);
}
```

The `loadXML()` method in this case takes as its parameter the `InputStream` containing the XML information (in this case, a file called `data.xml`). The `XMLLoadProperties` property may be used to specify options related to interpreting XML input.

For a demonstration of loading your document from an XML file, please see the XML Data example, automatically installed in `JCLASS_SERVER_HOME/examples/sreport/main/`

XMLLoadProperties object

`XMLLoadProperties` is a class that contains properties, set by a user, that may be needed during XML parsing.

Property	Description
<code>ResolvingServletContext</code>	An instance of <code>ServletContext</code> that will be used to resolve files from within a servlet.
<code>ResolvingClass</code>	An instance of <code>Class</code> that will be used to resolve files from the <code>CLASSPATH</code> .
<code>RelativeURLPrefix</code>	A prefix prepended to relative URLs specified in XML tags (for example, <code><embed-image></code> and <code><embed-chart></code>).

Property	Description
IgnoreExternalResource Exceptions	Ignores exceptions thrown when reading files external to the primary XML file, such as external text styles. Default is false.

13.3.5 Loading Document XML Strings

The following example uses `java.io.StringReader` and `org.xml.sax.InputSource` classes to load an XML document that is defined as a `String` (`documentData`) earlier in the program.

```
XMLLoadProperties loadProperties = new XMLLoadProperties();
loadProperties.setResolvingServletContext(servletContext);

JCDocument document = null;
try {
    document = new JCDocument(stream);
    document.loadXML(
        new InputSource(new StringReader(documentData)), loadProperties);
}
catch (Exception e) {
    System.err.println("Error loading document data = " + e);
    System.exit(1);
}
```

13.3.6 Configuring an XML Input Source To Create a PDF or RTF

In order to create XML input source that configures and creates a PDF or RTF document, follow these steps.

1. Write your XML.
2. Create an output stream.
3. Create a `JCDocument` object in your code with your output stream.
4. Call `JClass ServerReport`'s `loadXML()` method, which calls the parser methods to delineate/parse the XML, create the appropriate objects, and write to the current document.
5. Call `document.print();`.

13.4 Using JClass ServerReport to Convert from XML to HTML

You may want to convert your XML document to HTML. Although `JClass ServerReport` is not required for this process, for your convenience we outline here the steps required.

From your XML source you can use Extensible Style Sheets Language (XSL) to translate your XML to HTML format. For an overview, please see the Phone Bill demo, automatically installed in `JCLASS_SERVER_HOME/demos/sreport/phone/`.

13.4.1 Extensible Style Sheets Language (XSL) Background

XSL is one of the two ways to design XML documents to be displayed in browsers (the other is via cascading style sheets [CSS]). For instance, using XSL, you can indicate how a tag that you created should be displayed.

XSL comprises two parts:

- transformation language
- formatting language

Basically, the transformation language lets you convert documents into different forms, while the formatting language lets you format the documents. Because the transformation process allows you to add the tags that the formatting process requires, one generally first transforms the document, then formats it.

You can use XSL to transform your own XML into a PDF – via JClass ServerReport document XML – or into HTML.



The next section, [XSL Transformation Language \(XSLT\)](#), goes into more detail about the transformation language. The following section, [XSL Formatting Language](#), delves into XSL formatting objects.

XSL Transformation Language (XSLT)

XSLT is a language for transforming XML documents into other documents, for instance, an HTML document.

In brief, one uses XSLT to alter and manipulate documents, to effect the changes you want. XSLT adds formatting/styling information to an XML source document by transforming it into a document comprising XSL formatting objects (or into other formats such as HTML, XHTML, or SVG). This transformation is accomplished via a style sheet, with syntax in well-formed XML.

Note that in order to use XSLT, you will need an XSLT processor, such as *xalan.jar* (from the Apache Software Foundation), in your CLASSPATH. *xalan.jar* is provided for your convenience in `JCLASS_SERVER_HOME/jakarta-tomcat/lib/`.

The W3C recommendations for XSLT are provided at <http://www.w3.org/TR/>. This Web site has a plethora of information about XSLT and is an excellent resource.

Creating XSLT Transformations

To create XSLT transformations, you need:

- the document to transform;
- the style sheet (must be a properly formatted XML document) that outlines the transformations.

For instance, when converting an XML document into HTML, you would need to add HTML's `<P>` tags where appropriate.

As an example, here is an XSLT style sheet for transforming generic data-oriented XML into HTML from the Phone Bill demo, automatically installed in `JCLASS_SERVER_HOME/demos/sreport/phone/`.

Note that to run this example, you will need an XSLT processor, such as *xalan.jar* (from the Apache Software Foundation), in your CLASSPATH. *xalan.jar* is provided in *JCLASS_SERVER_HOME/jakarta-tomcat/lib/*. If you are using the version of Tomcat that is shipped with JClass ServerReport, you are all ready to go.

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="calls">

<html><body>
<CENTER>
<h1>Jingle Bells Telephone Co.</h1>
<H2>Summary of Charges</H2></CENTER>
<center><h3>for Washington Irving
</h3></center><table width="100%" border="1"><tr><th>Date</th>
<th>Number</th><th>Location</th><th>Time</th><th>Duration</th>
<th>Rate</th><th>Cost</th></tr>
<xsl:apply-templates select="call" />
<tr><td colspan="6"><b>Total:</b></td>
<td align="right">${xsl:value-of
    select="format-number(sum(//cost),'###0.00')"} />
</td></tr>
</table></body></html>
</xsl:template>

<xsl:template match="call">
<tr>
<td>
<xsl:value-of select="date" />
</td><td>
<xsl:value-of select="number" />
</td><td>
<xsl:value-of select="location"/>
</td><td align="right">
<xsl:value-of select="time" />
</td><td align="right">
<xsl:value-of select="duration" />
</td><td align="right">
<xsl:value-of select="rate" />

```



```
        </td><td align="right">
        <xsl:value-of select="cost"/>
        </td>
    </tr>
</xsl:template>
</xsl:stylesheet>
```

XSL Formatting Language

Recall that XSL comprises a transformation language as well as a formatting language, and that this formatting language lets one design the documents.

The W3C has defined XSL's formatting objects; these include objects such as `root`, `character`, and so forth. Each object has its own XML tag, and the properties that each supports are attributes of the tag. One can use these predefined objects to specify explicitly the exact formatting for a document.

Details about formatting properties are outlined at <http://www.w3.org/TR/xsl/>.

Online XSL Formatting Object Resources

There are myriad online resources for learning about and using XSL formatting objects. Here are a few that may assist you.

<http://www.ibiblio.org/xml/books/bible2/chapters/ch18.html>

<http://xml.coverpages.org/xsl.html>

<http://www.renderx.com/~grig/xsl2k/>

JClass ServerReport DTD Tags

Structured Versus Linear Usage ■ *Document Tags* ■ *Page Template* ■ *Text Styles*
Draw Styles ■ *Symbol Styles* ■ *Table Styles*

The following is a list of the different tags available in JClass ServerReport through the DTD. Each tag will have information about its purpose, its sub-elements and attributes.

Note that in the following tables, color String is a String representing a named color, or a String of the format “r-g-b” where r, g, and b are integer values from 0 to 255 representing the red, green, and blue components of the color.

14.1 Structured Versus Linear Usage

Portions of JClass ServerReport’s XML functionality come in two varieties:

- structured – use XML to model structured document data which JClass ServerReport parses to create documents
- linear – use XML tags to specify a linear series of commands, similar to coding an application that calls JClass ServerReport methods

14.1.1 Structured

In the structured approach, you can use JClass ServerReport’s XML functionality as a convenient mark-up language before the XML file is parsed to create a document. You would utilize this approach when retrieving document data from a database or transforming data from an XML file that is in a different structured format before creating a PDF or RTF document. This approach would also be used if you wished to convert an XML document to another mark-up language, such as HTML, rather than to PDF.

14.1.2 Linear

In the linear approach, you can use JClass ServerReport’s XML functionality to process commands defined in an XML file and create a PDF or RTF document. You would use this approach when structure is not important in your input data, such as the case of reading data from an unformatted file or input stream. In the linear case, you would use tags such as <new-

paragraph/>, <new-section/>, and <current-text-style/> instead of <paragraph>, <section>, and <use-text-style>.

Note: Although it is possible to mix both Structural and Linear tags in the same XML document, this is not advisable because results may be unpredictable.

14.2 Document Tags

Document Tags are contained in *JCServerReport.dtd*.

Document Body Tags

The following table contains a set of elements that can be subelements of another element. The elements that can use these subelements refer back to this list.

■ <bold>	■ <new-column>
■ <current-text-style>	■ <new-line>
■ <embed-image>	■ <new-page>
■ <external-java-code>	■ <new-paragraph>
■ <float-image>	■ <new-section>
■ <flow-table>	■ <overflow-rules>
■ <horizontal-rule>	■ <page-table>
■ <hyperlink>	■ <paragraph>
■ <italic>	■ <section>
■ <list>	■ <space>
■ <list-item>	■ <tab>
■ <macro>	■ <unsupported-operation>
■ <mark-location>	■ <use-text-style>

14.2.1 alt-flow

Purpose:

Specifies content to be added to the flow in the event that the containing tag fails to perform its function.

**Equivalent in
JClass ServerReport:** n/a

Contained in: <embed-image>, <float-image>, <embed-chart>,
<float-chart>, <embed-media-clip>, <float-media-clip>

Sub-Elements: text or any [Document Body Tags](#)

Attributes: none

14.2.2 bold

Purpose:

Modifies the current text style to be bold; if the current style is italic, then the style will be bold-italic – contains text and tags that will use the modified text style.

**Equivalent in
JClass ServerReport:** JCTextStyle.setFontStyle(Font.BOLD)

Contained in: <flow>, <paragraph>, <section>, <doc-frame>, <header-
frame>, <footer-frame>, <cell>

Sub-Elements: text or any [Document Body Tags](#)

Attributes: none

14.2.3 bookmark

Purpose:

Defines the individual links in a bookmark tree.

Equivalent in JClass ServerReport: `DefaultMutableTreeNode, BookmarkNode`

Contained in: `<bookmark-tree>, <bookmark>`

Sub-Elements: `<bookmark>`

Attributes:

Name	Definition	Possible Values	Equivalent Method
bookmark-text	Provides the text to be displayed and associated with the link in the Acrobat Bookmarks tab. Required.	String	n/a
destination-name	Defines the location that the bookmark points to. Required.	String	n/a
expanded	Determines if the node will be expanded or closed by default in the Acrobat Bookmarks tab.	■ true ■ false	n/a

14.2.4 bookmark-tree

Purpose:

Defines the beginning of the bookmark tree (the collection of links that exists outside of the PDF, but that acts like a table of contents and points to locations inside the PDF).

Equivalent in JClass ServerReport: `JCDocument.setBookmarkTree()`

Contained in: `<document>`

Sub-Elements: `<bookmark>`

Attributes:

Name	Definition	Possible Values	Equivalent Method
visible	Determines if the Bookmarks tab is visible when Acrobat is opened.	■ true ■ false	<code>JCDocument.setBookmarkTreeVisible()</code>

14.2.5 cell

Purpose:

Defines the contents of a cell within a row.

Equivalent in

JCFlowTable.printToCell, JCPageTable.printToCell

JClass ServerReport:**Contained in:**

<row>

Sub-Elements:

text or any [Document Body Tags](#)

Attributes:

Name	Definition	Possible Values	Equivalent Method
color	Background color of the cell.	color String	■ JCFlowTable.setBackgroundColor() ■ JCPageTable.setBackgroundColor()
bottom-border-precedence	Determines the precedence of borders when a bottom border has been specified for this cell and a top border for the one directly below.	■ this-cell ■ next-cell	■ pageTable.getCell(row, column).setBottomBorderPrecedence() ■ flowTable.getCell(column).setBottomBorderPrecedence()
bottom-border-style-name	The style of the bottom border of the cell.	name of a pre-defined draw-style	■ pageTable.getCell(row, column).setBottomBorderStyle(drawStyle) ■ flowTable.getCell(column).setBottomBorderStyle(drawStyle)
column-span	Number of columns spanned by this cell.	integer	■ JCPageTable.spanCells() ■ JCFlowTable.spanCells()
left-border-style-name	The style of the left border of the cell.	name of a pre-defined draw-style	■ pageTable.getCell(row, column).setLeftBorderStyle(drawStyle) ■ flowTable.getCell(column).setLeftBorderStyle(drawStyle)

Name	Definition	Possible Values	Equivalent Method
right-border-precedence	Determines the precedence of borders when a right border has been specified for this cell and a left border for the one directly to the right.	<ul style="list-style-type: none"> ■ this-cell ■ next-cell 	<ul style="list-style-type: none"> ■ pageTable.getCell(row, column).setRightBorderPrecedence() ■ flowTable.getCell(column).setRightBorderPrecedence()
right-border-style-name	The style of the right border of the cell.	name of a pre-defined draw-style	<ul style="list-style-type: none"> ■ pageTable.getCell(row, column).setRightBorderStyle(drawStyle) ■ flowTable.getCell(column).setRightBorderStyle(drawStyle)
row-span	Number of rows spanned by this cell. Note: This applies only to a <page-table> cell	integer	JCPageTable. spanCells()
text-style-name	The text style to use for cell contents. Text styles that can be used are those that are defined in XML or programmatically, as well as the Default styles defined in the JCTextStyle call.	name of a pre-defined text-style	<ul style="list-style-type: none"> ■ JCFlowTable.Cell.setStyle() ■ JCPageTable.Cell.setStyle()
top-border-style-name	The style of the top border of the cell.	name of a pre-defined draw-style	<ul style="list-style-type: none"> ■ pageTable.getCell(row, column).setTopBorderStyle(drawStyle) ■ flowTable.getCell(column).setTopBorderStyle(drawStyle)

14.2.6 chart-data

Purpose:

Specifies the location of a JClass ServerChart datasource as an XML stream.

For more information, see “Using XML with JClass ServerChart” in the *JClass ServerChart Programmer’s Guide*.

Equivalent in n/a

JClass ServerReport:

Contained in: <embed-chart>, <float-chart>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
data-view-name	The name of the ServerChart data view that will be populated with data from the specified XML stream. If absent, the data view is implied from the order of this tag within the list of <chart-data> tags.	data view name	n/a
data-url	Specifies the location of the chart data XML stream as an URL.	URL	n/a
data-servlet-resource	Specifies the location of the chart data XML stream as a file to be resolved with the <code>ServletContext.getResourceAsStream()</code> method. For this property to work correctly, a <code>ServletContext</code> class must be set in the <code>ResolvingServletContext</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
data-class-resource	Specifies the location of the chart data XML stream as a file to be resolved with the <code>Class.getResourceAsStream()</code> method. For this property to work correctly, a <code>Class</code> object must be set in the <code>ResolvingClass</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a

Name	Definition	Possible Values	Equivalent Method
data-file	Specifies the location of the chart data XML stream as a file name.	file name	n/a
data-relative-url	Specifies the location of the chart XML stream as a URL. The URL is relative to a prefix that has been set in the <code>RelativeURLPrefix</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a

14.2.7 column-info

Purpose:

Defines the width of a table column; multiple occurrences, one for each column in the table.

Equivalent in `JCPageTable` constructor, `JCFlowTable` constructor

JClass ServerReport:

Contained in: `<page-table>`, `<flow-table>`

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
border-style-name	Border style to apply to all column borders.	name of a pre-defined draw-style	<ul style="list-style-type: none"> ■ <code>JCFlowTable.setColumnBorder()</code> ■ <code>JCPageTable.setColumnBorder()</code>
color	Background color of the column.	color String	<ul style="list-style-type: none"> ■ <code>JCFlowTable.setBackgroundColor()</code> ■ <code>JCPageTable.setBackgroundColor()</code>
left-border-style-name	Border style to apply to the left border of the table.	name of a pre-defined draw-style	<ul style="list-style-type: none"> ■ <code>JCFlowTable.setLeftBorder()</code> ■ <code>JCPageTable.setLeftBorder()</code>
unit	Determines the unit of measurement used for width. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a

Name	Definition	Possible Values	Equivalent Method
width	Width of the column expressed in terms of unit. Required.	floating point number	JCPageTable constructor, JCFlowTable constructor

14.2.8 contents-list

Purpose:

Defines a contents list, such as a table of contents or list of figures.

Equivalent in JCContentsList
JClass ServerReport:

Contained in: <document>

Sub-Elements: <default-text-style>

Attributes:

Name	Definition	Possible Values	Equivalent Method
name	Specifies the name of this contents list. Required.	String	JCContentsList. setName()
template-list-name	Specifies the name of a list of page templates to use for the contents list. If none are specified, the default page templates for the document are used.	name of a pre-defined template list	JCContentsList. setTemplates()
flow-page-template	Specifies the name of the page template (from the front matter) to use after this contents list prints.	name of a pre-defined page-template	JCContentsList. setFlowPageTemplate()

14.2.9 contents-list-entry

Purpose:

Creates an entry in the specified contents list. For example, if the contents list is a table of contents, the entries would correspond to the headings in the document.

Equivalent in JClass ServerReport: JCContentsListEntry

Contained in: <flow> (body flow, not front matter flow)

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
auto-numbering	Determines whether the entries are auto-numbered. When the value is on or parent, numbers are inserted after the tag and before the text.	■ none ■ on ■ parent	JCContentsListEntry. setAutoNumbering()
blank-line	Determines whether this entry is a blank line.	■ true ■ false	JCContentsListEntry. setBlankLine()
contents-list-name	Specifies the contents-list to which this entry belongs. Required.	name of a pre-defined contents-list	n/a
hyperlink-auto-style	If hyperlink-used is true, turns on and off the auto styling features. A value of false allows linked text to appear as regular text.	■ true ■ false	JCContentsListEntry. setHyperlinkAutoStyle()
hyperlink-text-style-name	Specifies the text style to use for linked text.	name of a pre-defined text-style	JCContentsListEntry. setHyperlinkStyle()
hyperlink-used	Determines whether the entry text is a hyperlink.	■ true ■ false	JCContentsListEntry. setHyperlinkUsed()
level	Specifies the heading level. Required.	non-negative integer	JCContentsListEntry. setLevel()

Name	Definition	Possible Values	Equivalent Method
location-name	Specifies a marked location within the document. The page number for the entry is determined using this attribute. Required.	name of a pre-defined mark-location	JCContentsListEntry.setLocation()
page-break	Determines whether this entry is a page break. All other fields in the entry are ignored.	<ul style="list-style-type: none"> ■ true ■ false 	JCContentsListEntry.setPageBreak()
tag	Specifies text to appear before the entry's text.	String	JCContentsListEntry.setTag()
tag-printed	Determines whether the tag prints in the contents list.	<ul style="list-style-type: none"> ■ true ■ false 	JCContentsListEntry.setTagPrinted()
text	Specifies the main text of the entry. Required.	String	JCContentsListEntry.setText()
text-style-name	The text style to use when printing the entry in the contents list. If not specified, the default-text-style for the level is used.	name of a pre-defined text-style	JCContentsListEntry.setTextStyle()

14.2.10 current-text-style

Purpose:

Sets the current text style used by the document.

Equivalent in JFlow.setCurrentTextStyle()

JClass ServerReport:

Contained in: <flow>, <paragraph>, <section>, <cell>, <doc-frame>, <header-frame>, <footer-frame>,

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
name	Name of text style to be used. Required.	name of a pre-defined text-style	n/a

14.2.11 default-text-style

Purpose:

Sets a default text style to be used in contents lists.

Equivalent in JCContentsList.setDefaultTextStyle()

JClass ServerReport:

Contained in: <contents-list>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
name	Name of text style to be used. Required.	name of a pre-defined text-style	n/a
level	The level assigned to this text style. For example, 0 could be for chapter titles, 1 for heading level 1, etc. Required.	integer	n/a

14.2.12 doc-frame

- Purpose:**
References a pre-defined static frame defined in a full page template – contains text and tags belonging to this frame. Located in *ServerReportDocFrame.dtd*.
- Equivalent in JClass ServerReport:** JCPage.stringToFrame()
- Contained in:** <document>
- Sub-Elements:** text or any [Document Body Tags](#)

Attributes:

Name	Definition	Possible Values	Equivalent Method
name	Name of the previously defined frame to reference. Required.	String	n/a
page-template-name	If the page-name is not in the document's page template list, specify the name of the page template list that contains the page-name. Used to allow access to template lists from the front matter.	String	n/a
page-name	Name of the page that contains this previously defined frame. Required.	String	n/a

14.2.13 document

- Purpose:**
Defines the contents of a JClass ServerReport document.
- Equivalent in JClass ServerReport:** JCDocument class
- Contained in:** top-level tag
- Sub-Elements:** <bookmark-tree>, <page-template>, <text-styles>, <draw-styles>, <symbol-styles>, <table-styles>, <document properties>, <overflow-rules>, <bookmark-tree>, <contents-list>, <doc-frame>, <header-frame>, <footer-frame>, <flow> (required), <front-matter>, <external-java-code>
- Attributes:** none

14.2.14 document-properties

Purpose:

Sets document-level properties before document is populated. Supported for documents using the simple page templates only.

Equivalent in

Various `JCDocument` set methods.

JClass ServerReport:**Contained in:**

<document>

Sub-Elements:

none

Attributes:

Name	Definition	Possible Values	Equivalent Method
facing-pages	Determines whether facing pages are used. Default is <code>false</code> . When <code>true</code> , left- and right-side pages have different headers, footers, and gutters. When <code>false</code> , all pages use the same headers, footers, and gutters.	■ <code>true</code> ■ <code>false</code>	n/a

14.2.15 embed-chart

Purpose:
Embeds an instance of JClass ServerChart in the document at the current position. The instance of ServerChart is created from a user-specified XML stream; ServerChart data may be specified in its own separate XML stream using the <chart-data> tag.

If an instance of LoadServerProperties is set in the ChartLoadProperties property of ServerReport's XMLLoadProperties class, it will be used when creating the embedded chart. This allows you to specify properties to use when loading the chart XML, such as classes used to resolve file locations and a user-specified object that is used by the chart XML's <external-java-code> tag. If nothing is set in the ChartLoadProperties property, a default instance of LoadServerProperties is created using similar values from the XMLLoadProperties class.

For more information, see “Using XML with JClass ServerChart” in the *JClass ServerChart Programmer’s Guide*.

Note: JClass ServerChart must be installed for this feature to work.

Equivalent in JClass ServerReport: JCFLOW.embedComponent(), JCFRAME.embedComponent()

Contained in: <flow>, <paragraph>, <section>, <doc-frame>, <header-frame>, <footer-frame>, <cell>

Sub-Elements: <chart-data>, <alt-flow>

Attributes:

Name	Definition	Possible Values	Equivalent Method
align	Vertical alignment of the chart within the current line.	<ul style="list-style-type: none">■ center-in-line■ center-on-baseline■ center-above-baseline■ align-to-top■ align-to-bottom■ position-on-baseline■ position-below-baseline	n/a

Name	Definition	Possible Values	Equivalent Method
chart-charset	The character set to use.	String	n/a
chart-class-resource	Specifies the location of the chart XML stream as a file to be resolved with the <code>Class.getResourceAsStream()</code> method. For this property to work correctly, a <code>Class</code> object must be set in the <code>ResolvingClass</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
chart-file	Specifies the location of the chart XML stream as a file name.	file name	n/a
chart-relative-url	Specifies the location of the chart XML stream as a URL. This URL is relative to a prefix that has been set in the <code>RelativeURLPrefix</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
chart-servlet-resource	Specifies the location of the chart XML stream as a file to be resolved with the <code>ServletContext.getResourceAsStream()</code> method. For this property to work correctly, a <code>ServletContext</code> class must be set in the <code>ResolvingServletContext</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
chart-url	Specifies the location of the chart XML stream as an URL.	URL	n/a
height	Height of the chart within the document expressed in terms of unit.	floating point number	n/a
unit	Determines the unit of measurement with width and height. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
width	Width of the chart within the document expressed in terms of unit.	floating point number	n/a

14.2.16 embed-image

Purpose:

Embeds an image in the document at the current position.

Equivalent in

JCFlow.embedImage(), JCFrame.embedImage()

JClass ServerReport:**Contained in:**

<flow>, <paragraph>, <section>, <doc-frame>,
<header-frame>, <footer-frame>, <cell>

Sub-Elements:

<alt-flow>

Attributes:

Name	Definition	Possible Values	Equivalent Method
align	Vertical alignment of the image within the current line.	<ul style="list-style-type: none">■ center-in-line■ center-on-baseline■ center-above-baseline■ align-to-top■ align-to-bottom■ position-on-baseline■ position-below-baseline	n/a
height	Height of the image within the document expressed in terms of unit.	floating point number	n/a
source-class-resource	Specifies the location of the image as a file to be resolved with the <code>Class.getResource()</code> method. For this property to work correctly, a <code>Class</code> object must be set in the <code>ResolvingClass</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
source-file	Specifies the location of the image as a file name.	file name	n/a

Name	Definition	Possible Values	Equivalent Method
source-relative-url	Specifies the location of the image as a URL. This URL is relative to a prefix that has been set in the <code>RelativeURLPrefix</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
source-servlet-resource	Specifies the location of the image as a file to be resolved with the <code>ServletContext.getResource()</code> method. For this property to work correctly, a <code>ServletContext</code> class must be set in the <code>ResolvingServletContext</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
source-url	Specifies an image source or Base64 encoded image data. For an image source, the value of this property is the location of the image as a fully qualified URL. For Base64 encoded image data, the data is prefixed with the string "data:image/***;base 64," where xxx is the image format, such as png, jpg, or gif.	URL or Base64 data	n/a
unit	Determines the unit of measurement used with width and height. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
width	Width of the image within the document expressed in terms of unit.	floating point number	n/a

14.2.17 embed-media-clip

Purpose:

Embeds a media clip. If the media clip does not exist, the flow content included in the child `alt-flow` node will be added to the document flow.

Equivalent in JClass ServerReport: `JCFlow.embedMediaClip()`

Contained in: `<flow>`, `<paragraph>`, `<section>`, `<doc-frame>`, `<header-frame>`, `<footer-frame>`, `<cell>`

Sub-Elements: `<poster>`, `<alt-flow>`

Attributes:

Name	Definition	Possible Values	Equivalent Method
<code>align</code>	Vertical alignment of the clip within the current line.	<ul style="list-style-type: none">■ <code>center-in-line</code>■ <code>center-on-baseline</code>■ <code>center-above-baseline</code>■ <code>align-to-top</code>■ <code>align-to-bottom</code>■ <code>position-on-baseline</code>■ <code>position-below-baseline</code>	n/a
<code>description</code>	A description of the clip. Used for alternative text if the media clip cannot be played.	String	n/a
<code>height</code>	Height of the clip within the document expressed in terms of unit.	floating point number	n/a
<code>media-clip-class-resource</code>	Specifies the location of the media clip as a file to be resolved with the <code>Class.getResource()</code> method. For this property to work correctly, a <code>Class</code> object must be set in the <code>ResolvingClass</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
<code>media-clip-file</code>	Specifies the location of the media clip as a file name.	file name	n/a

Name	Definition	Possible Values	Equivalent Method
media-clip-relative-url	Specifies the location of the media clip as a URL. This URL is relative to a prefix that has been set in the <code>RelativeURLPrefix</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
media-clip-servlet-resource	Specifies the location of the media clip as a file to be resolved with the <code>ServletContext.getResource()</code> method. For this property to work correctly, a <code>ServletContext</code> class must be set in the <code>ResolvingServletContext</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
media-clip-url	Specifies the location of the media clip as a fully qualified URL.	URL	n/a
mime-type	Specifies the MIME type of the clip, which identifies the type of media clip used. Required. (Typical values include “video/mpeg”, “video/quicktime”, or “video/x-svideo”.)	String	n/a
name	A unique name for this media clip. Required.	String	n/a
title	Specifies a title for the clip.	String	n/a
unit	Determines the unit of measurement used for width and height. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
width	Width of the clip within the document expressed in terms of unit.	floating point number	n/a

14.2.18 external-java-code

Purpose:
Calls the specified Java class; for more information, please see [Using the external-java-code Tag](#).

Equivalent in JClass ServerReport: none

Contained in: <document>, <flow>, <paragraph>, <section>, <cell>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
class	Name of the Java class to be called. Required. For more information, see Using the external-java-code Tag .	String	n/a

14.2.19 float-chart

Purpose:

Floats an instance of JClass ServerChart in the document, meaning that it will appear whenever there is enough space for it to do so. The instance of ServerChart is created from a user-specified XML stream; ServerChart data may be specified in its own separate XML stream using the `<chart-data>` tag.

See the chapter entitled Using XML with JClass ServerChart in the JClass ServerChart programmer's guide for more information.

Note: JClass ServerChart must be installed for this feature to work.

Equivalent in

JCFlow.floatComponent(), JCFrame.floatComponent()

JClass ServerReport:

Contained in:

`<flow>`, `<paragraph>`, `<section>`, `<doc-frame>`, `<header-frame>`, `<footer-frame>`, `<cell>`

Sub-Elements:

`<chart-data>`, `<alt-flow>`

Attributes:

Name	Definition	Possible Values	Equivalent Method
align	Horizontal alignment of the image within the current frame.	■ left ■ right ■ center ■ justify ■ none	n/a
chart-url	Specifies the location of the chart XML stream as an URL.	URL	n/a
chart-charset	The character set to use.	String	n/a
chart-class-resource	Specifies the location of the chart XML stream as a file to be resolved with the <code>Class.getResourceAsStream()</code> method. For this property to work correctly, a <code>Class</code> object must be set in the <code>ResolvingClass</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
chart-file	Specifies the location of the chart XML stream as a file name.	file name	n/a

Name	Definition	Possible Values	Equivalent Method
chart-relative-url	Specifies the location of the chart XML stream as a URL. This URL is relative to a prefix that has been set in the <code>RelativeURLPrefix</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
chart-servlet-resource	Specifies the location of the chart XML stream as a file to be resolved with the <code>ServletContext.getResourceAsStream()</code> method. For this property to work correctly, a <code>ServletContext</code> class must be set in the <code>ResolvingServletContext</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
height	Height of the chart within the document expressed in terms of unit.	floating point number	n/a
unit	Determines the unit of measurement used for width and height. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
width	Width of the chart within the document expressed in terms of unit.	floating point number	n/a

14.2.20 float-image

Purpose:

Floats an image in the document, meaning that it will appear whenever there is enough space for it to do so.

Equivalent in

JCFlow.floatImage(), JCFrame.floatImage()

JClass ServerReport:

Contained in:

<flow>, <paragraph>, <section>, <doc-frame>, <header-frame>, <footer-frame>, <cell>

Sub-Elements:

<alt-flow>

Attributes:

Name	Definition	Possible Values	Equivalent Method
align	Horizontal alignment of the image within the current frame.	■ left ■ right ■ center ■ justify ■ none	n/a
height	Height of the image within the document expressed in terms of unit.	floating point number	n/a
source-class-resource	Specifies the location of the image as a file to be resolved with the <code>Class.getResource()</code> method. For this property to work correctly, a <code>Class</code> object must be set in the <code>ResolvingClass</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
source-file	Specifies the location of the image as a file name.	file name	n/a
source-relative-url	Specifies the location of the image as a URL. This URL is relative to a prefix that has been set in the <code>RelativeURLPrefix</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a

Name	Definition	Possible Values	Equivalent Method
source-servlet-resource	Specifies the location of the image as a file to be resolved with the <code>ServletContext.getResource()</code> method. For this property to work correctly, a <code>ServletContext</code> class must be set in the <code>ResolvingServletContext</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
source-url	Specifies an image source or Base64 encoded image data. For an image source, the value of this property is the location of the image as a fully qualified URL. For Base64 encoded image data, the data is prefixed with the string "data:image/***;base 64," where xxx is the image format, such as png, jpg, or gif.	URL or Base64 data	n/a
unit	Determines the unit of measurement used for width and height. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
width	Width of the image within the document expressed in terms of unit.	floating point number	n/a

14.2.21 float-media-clip

Purpose:

Floats a media clip in the document flow. If the media clip does not exist, the flow content included in the child `alt-flow` node will be added to the document flow.

Equivalent in

`JCFlow.floatMediaClip`

JClass ServerReport:

Contained in:

`<flow>`, `<paragraph>`, `<section>`, `<doc-frame>`,
`<header-frame>`, `<footer-frame>`, `<cell>`

Sub-Elements:

`<poster>`, `<alt-flow>`

Attributes:

Name	Definition	Possible Values	Equivalent Method
<code>align</code>	Vertical alignment of the clip within the current line.	<ul style="list-style-type: none">■ <code>center-in-line</code>■ <code>center-on-baseline</code>■ <code>center-above-baseline</code>■ <code>align-to-top</code>■ <code>align-to-bottom</code>■ <code>position-on-baseline</code>■ <code>position-below-baseline</code>	n/a
<code>description</code>	A description of the clip. Used for alternative text if the media clip cannot be played.	String	n/a
<code>height</code>	Height of the clip within the document expressed in terms of unit.	floating point number	n/a
<code>media-clip-class-resource</code>	Specifies the location of the media clip as a file to be resolved with the <code>Class.getResource()</code> method. For this property to work correctly, a <code>Class</code> object must be set in the <code>ResolvingClass</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a

Name	Definition	Possible Values	Equivalent Method
media-clip-file	Specifies the location of the media clip as a file name.	file name	n/a
media-clip-relative-url	Specifies the location of the media clip as a URL. This URL is relative to a prefix that has been set in the <code>RelativeURLPrefix</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
media-clip-servlet-resource	Specifies the location of the media clip as a file to be resolved with the <code>ServletContext.getResource()</code> method. For this property to work correctly, a <code>ServletContext</code> class must be set in the <code>ResolvingServletContext</code> property of the <code>XMLLoadProperties</code> class, which is passed to <code>JCDocument.loadXML()</code> .	file name	n/a
media-clip-url	Specifies the location of the media clip as a fully qualified URL.	URL	n/a
mime-type	Specifies the MIME type of the clip, which identifies the type of media clip used. Required. (Typical values include “video/mpeg”, “video/quicktime”, or “video/x-svideo”.)	String	n/a
name	A unique name for this media clip. Required.	String	n/a
title	Specifies a title for the clip.	String	n/a
unit	Determines the unit of measurement used for height and width. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
width	Width of the clip within the document expressed in terms of unit.	floating point number	n/a

14.2.22 flow

Purpose:

Starts the flow of text to the document – contains text and tags belonging to this document. Located in *ServerReportFlow.dtd*.

**Equivalent in
JClass ServerReport:** JCFlow

Contained in: <document>

Sub-Elements: text or any [Document Body Tags](#)

Attributes: none

14.2.23 flow-table

Purpose:

Creates a JCFlowTable instance and sends it to the flow – <row> is required.

**Equivalent in
JClass ServerReport:** JCFlowTable class

Contained in: <flow>, <paragraph>, <section>

Sub-Elements: <column-info>, <header-table>, <row>

Attributes:

Name	Definition	Possible Values	Equivalent Method
align	Horizontal alignment of table within current frame.	■ left ■ right ■ center ■ justify ■ none	JCFlowTable. setAlignment()
border-mode	Determines the type of top and bottom row borders to draw at page breaks.	■ external ■ internal	JCFlowTable. setBorderMode()
border-style-name	Border to use for the entire table.	name of a pre-defined draw-style	JCFlowTable. setAllBorders()
bottom-border-enabled	Determines whether a bottom border is used.	■ true ■ false	JCFlowTable.set BottomBorder Enabled()

Name	Definition	Possible Values	Equivalent Method
bottom-border-style-name	Border to use for the bottom border of the table.	name of a pre-defined draw-style	JCFlowTable.setBottomBorder()
color	Background color of the table.	color String	JCFlowTable.setBackgroundColor()
default-bottom-margin	Default bottom margin of a cell expressed in terms of unit.	floating point number	JCFlowTable.setDefaultBottomMargin()
default-cell-alignment	Default vertical alignment of text within cells.	<ul style="list-style-type: none"> ■ none ■ center ■ bottom ■ top 	JCFlowTable.setDefaultCellAlignment()
default-left-margin	Default left margin of a cell expressed in terms of unit.	floating point number	JCFlowTable.setDefaultLeftMargin()
default-right-margin	Default right margin of a cell expressed in terms of unit.	floating point number	JCFlowTable.setDefaultRightMargin()
default-text-style-name	Default text style to use for table cells. Text styles that can be used are those that are defined in XML or programmatically, as well as the Default style defined in the JCTextStyle class.	name of a pre-defined text-style	JCFlowTable.setDefaultStyle()
default-top-margin	Default top margin of a cell expressed in terms of unit.	floating point number	JCFlowTable.setDefaultTopMargin()
edge-border-enabled	Determines whether an edge border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JCFlowTable.setEdgeBorderEnabled()
edge-border-style-name	Border to use for all edge borders in the table.	name of a pre-defined draw-style	JCFlowTable.setEdgeBorders()
fit-to-frame	Stretch the width of the table to fit the current frame.	<ul style="list-style-type: none"> ■ true ■ false 	JCFlowTable.fitToFrame()
header-border-enabled	Determines whether a header border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JCFlowTable.setHeaderBorderEnabled()

Name	Definition	Possible Values	Equivalent Method
header-border-style-name	Border to use between the header and the rest of the table.	name of a pre-defined draw-style	JCFlowTable. setHeaderBorder()
horizontal-border-style-name	Border to use for all horizontal borders in the table.	name of a pre-defined draw-style	JCFlowTable. setHorizontalBorders()
internal-border-style-name	Border to use for all internal borders in the table.	name of a pre-defined draw-style	JCFlowTable.set InternalBorders()
left-border-enabled	Determines whether a left border is used.	■ true ■ false	JCFlowTable.set LeftBorder Enabled()
left-border-style-name	Border to use for the left border of the table.	name of a pre-defined draw-style	JCFlowTable. setLeftBorder()
num-columns	Number of columns in the table. Required.	integer	JCFlowTable constructor
right-border-enabled	Determines whether a right border is used.	■ true ■ false	JCFlowTable.set RightBorder Enabled()
right-border-style-name	Border to use for the right border of the table.	name of a pre-defined draw-style	JCFlowTable. setRightBorder()
row-column-dominance	Determines which of row or column backgrounds and borders are drawn first.	■ row ■ column	JCFlowTable.setRow ColumnDominance()
table-style-name	Table style to use for this table. Table styles that can be used are those that are defined in XML or programmatically, as well as the Default styles defined in the JCTableStyle class.	name of a pre-defined table-style	JCFlowTable constructor
top-border-enabled	Determines whether a top border is used.	■ true ■ false	JCFlowTable.set TopBorder Enabled()

Name	Definition	Possible Values	Equivalent Method
top-border-style-name	Border to use for the top border of the table.	name of a pre-defined draw-style	JCFlowTable. setTopBorder()
unit	Determines the unit of measurement used for default-bottom-margin, default-left-margin, default-right-margin, and default-top-margin. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
vertical-border-style-name	Border to use for all vertical borders in the table.	name of a pre-defined draw-style	JCFlowTable.set VerticalBorders()

14.2.24 footer-frame

Purpose:

References a pre-defined static footer frame that is defined in a simple page template

Equivalent in JCPage.getFooterFrame
JClass ServerReport:

Contained in: <document>

Sub-Elements: text or any [Document Body Tags](#)

Attributes:

Name	Definition	Possible Values	Equivalent Method
page-name	Name of the page that contains the referenced frame. Required.	String	n/a
page-template-name	If the page-name is not in the document's page template list, specify the name of the page template list that contains the page-name.	String	n/a

Name	Definition	Possible Values	Equivalent Method
type	<p>Specifies which footer to reference. Default is non-facing-page.</p> <p>When JCDocument FacingPages is False, use non-facing-page to specify the same footer on all pages.</p> <p>When JCDocument FacingPages is True, use facing-left-page or facing-right-page to specify different footers on left and right pages.</p> <p>When the first-page-different tag is True, use first-page-in-section to specify a different footer for the first page of the section.</p>	<ul style="list-style-type: none"> ■ non-facing-page ■ facing-left-page ■ facing-right-page ■ first-page-in-section 	n/a

14.2.25 front-matter

Purpose:

Adds the specified contents-list to the front matter flow.

Equivalent in None

JClass ServerReport:

Contained in: <document>

Sub-Elements: <flow>

Attributes:

Name	Definition	Possible Values	Equivalent Method
contents-list-name	Specifies the name of a contents list to be the first page of the front matter.	name of a pre-defined contents-list	n/a
template-list-name	Specifies a list of page templates to use for the front matter. If none are specified, the page templates for the document are used.	name of a pre-defined list of templates	n/a

14.2.26 header-frame

Purpose:

References a pre-defined static header frame that is defined in a simple page template.

Equivalent in

JCPage.getHeaderFrame()

JClass ServerReport:**Contained in:**

<document>

Sub-Elements:

text or any [Document Body Tags](#)

Attributes:

Name	Definition	Possible Values	Equivalent Method
page-name	Required. Name of the page that contains the referenced frame.	String	n/a
page-template-name	If the page-name is not in the document's page template list, specify the name of the page template list that contains the page-name.	String	n/a
type	<p>Specifies which header to reference. Default is non-facing-page.</p> <p>When JCDocument FacingPages is False, use non-facing-page to specify the same header on all pages.</p> <p>When JCDocument FacingPages is True, use facing-left-page or facing-right-page to specify different headers on left and right pages.</p> <p>When the first-page-different tag is True, use first-page-in-section to specify a different header for the first page of the section.</p>	<ul style="list-style-type: none">■ non-facing-page■ facing-left-page■ facing-right-page■ first-page-in-section	n/a

14.2.27 header-table

Purpose:

Creates the header table of a flow table or page table – `<row>` is required. If not specified, the external header border styles are inherited from the main table. Note that the bottom border style cannot be specified here; use the `header-border-style-name` attribute of the main table instead.

Equivalent in

`JCPageTable.createHeaders()`,

JClass ServerReport:

`JCFlowTable.createHeaders()`

Contained in:

`<page-table>`, `<flow-table>`

Sub-Elements:

`<row>`

Attributes:

Name	Definition	Possible Values	Equivalent Method
<code>left-border-enabled</code>	Determines whether a left border is used.	■ <code>true</code> ■ <code>false</code>	■ <code>JCFlowTable.setLeftBorderEnabled()</code> ■ <code>JCPageTable.setLeftBorderEnabled()</code>
<code>left-border-style-name</code>	Border style to use for the left border of the table.	name of a pre-defined draw-style	■ <code>JCFlowTable.setLeftBorder()</code> ■ <code>JCPageTable.setLeftBorder()</code>
<code>right-border-enabled</code>	Determines whether a right border is used.	■ <code>true</code> ■ <code>false</code>	■ <code>JCFlowTable.setRightBorderEnabled()</code> ■ <code>JCPageTable.setRightBorderEnabled()</code>
<code>right-border-style-name</code>	Border style to use for the right border of the table.	name of a pre-defined draw-style	■ <code>JCFlowTable.setRightBorder()</code> ■ <code>JCPageTable.setRightBorder()</code>
<code>top-border-enabled</code>	Determines whether a top border is used.	■ <code>true</code> ■ <code>false</code>	■ <code>JCFlowTable.setTopBorderEnabled()</code> ■ <code>JCPageTable.setTopBorderEnabled()</code>
<code>top-border-style-name</code>	Border style to use for the top border of the table.	name of a pre-defined draw-style	■ <code>JCFlowTable.setTopBorder()</code> ■ <code>JCPageTable.setTopBorder()</code>

14.2.28 horizontal-rule

Purpose:
Draws a horizontal line.

Equivalent in JCFLOW.hRule(), JCFrame.hRule()
JClass ServerReport:

Contained in: <flow>, <paragraph>, <doc-frame>, <header-frame>,
 <footer-frame>, <cell>, <section>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
draw-style-name	Name of the draw style to use to draw this line. Draw styles that can be used are those that are defined in XML or programmatically, as well as the Default styles that are defined in the JCDrawStyle class.	name of a pre-defined draw-style (For example, default double line)	n/a
percent-width	Width of the line as percent of width of the frame. Required.	floating point number	n/a

14.2.29 **hyperlink**

Purpose:

Creates a hyperlink.

Equivalent in

JCFlow.beginHyperlink(), JCFrame.beginHyperlink()

JClass ServerReport:**Contained in:**

<flow>, <doc-frame>, <header-frame>, <footer-frame>, <cell>, <section>, <paragraph>

Sub-Elements:

text or any [Document Body Tags](#)

Attributes:

Name	Definition	Possible Values	Equivalent Method
auto-style	Turns on and off the auto styling features. A value of false allows linked text to appear as regular text.	■ true ■ false	n/a
destination-name	Defines the name of the location or URL where the hyperlink points. Required.	String	n/a
destination-type	Determines if the link is internal or external. Required.	■ internal ■ external	n/a
link-text-style-name	States which text style will be applied to hyperlinked text. Text styles that can be used are those that are defined in XML or programmatically, as well as the Default styles defined in the JCellStyle class.	name of a pre-defined text-style	n/a

14.2.30 **italic**

Purpose:

Modifies the current text style to be italic; if the current style is bold, then the style will be bold-italic – contains text and tags that will use the modified text style.

Equivalent in

JCTextStyle.setFontStyle(Font.ITALIC)

JClass ServerReport:**Contained in:**

<flow>, <paragraph>, <section>, <doc-frame>, <header-frame>, <footer-frame>, <cell>

Sub-Elements:

text or any [Document Body Tags](#)

Attributes: none

14.2.31 **list**

Purpose:

Specifies an ordered or unordered list in the document flow. You can nest lists.

Equivalent in

JCList in JCFLOW and JCFrame

JClass ServerReport:**Contained in:**

<flow>, <paragraph>, <section>, <cell>

Sub-Elements:

<list>, <list-item>

Attributes:

Name	Definition	Possible Values	Equivalent Method
bullet-style	If type is unordered, determines whether text or a symbol is used for the bullet.	■ string ■ symbol	JCListBullet. setBullet Style()
numbering-style	If type is ordered, determines the type of alpha-numeric characters used in a numbered list.	■ numbered ■ alpha-lower ■ alpha-upper ■ roman-lower ■ roman-upper	JCListNumber. setNumbering Style()
tag-alignment	Determines the tab alignment of the item tag within the list. The tag is either a number or a bullet.	■ left ■ right ■ center	n/a

Name	Definition	Possible Values	Equivalent Method
tag-string	In an unordered list of type string, specifies the text to be used as a bullet. In other lists, specifies text to be added after the tag (only when the tag is a number, not a symbol). Default is a period, which looks like: 1. list item 2. list item 3. list item	String	JCList.set TagString
tag-symbol-style-name	Specifies a symbol style to use when the bullet-style is symbol.	name of a pre-defined symbol-style	JCListBullet. setBullet SymbolStyle()
tag-tab	Specifies the location of the tab stop for the tag expressed in terms of unit.	floating point number	JCList.set TagTab()
tag-text-style-name	Specifies a text style to use when rendering the tag-string.	name of a pre-defined text-style	JCList.setTag TextStyle()
text-tab	Specifies the location of the tab stop for the text expressed in terms of unit.	floating point number	JCList.set TextTab()
type	Determines whether the list is ordered or unordered.	<ul style="list-style-type: none"> ■ ordered ■ unordered 	JCList.set Type()
unit	Determines the unit of measurement used for tag-tab and text-tab. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a

14.2.32 list-item

Purpose:
An item in a list. List items may contain text, images, and hyperlinks.

Equivalent in JClass ServerReport: JListItem

Contained in: <list>

Sub-Elements: none

Attributes: none

14.2.33 macro

Purpose:
Specifies the use of one of JClass ServerReport's pre-defined macros.

Equivalent in JClass ServerReport: TextMacro class

Contained in: <flow>, <paragraph>, <section>, <doc-frame>, <header-frame>, <footer-frame>, <cell>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
name	The name of the macro to invoke. Required.	■ pageNum ■ romanPageNum ■ sectionNum ■ pageTotal ■ sectionPageNum ■ sectionPageTotal	n/a
text-style-name	The name of the text style to use for this macro. Text styles that can be used are those that are defined in XML or programmatically, as well as the Default styles defined in the JCTextStyle class.	name of a pre-defined text-style	n/a

14.2.34 mark-location

Purpose:

Specifies a location used for internal links.

Equivalent in**JClass ServerReport:**

```
JCFlow.markCurrentLocation(),  
JCFlow.markNextLocation(),  
JCFlow.markNextFloatLocation(),  
JCFlow.markExactFrameLocation(),  
JCFlow.markExactPageLocation(),  
JCFrame.markCurrentLocation(),  
JCFrame.markNextLocation(),  
JCFrame.markNextFloatLocation(),  
JCFrame.markExactLocation(),  
JCFrame.markExactPageLocation(),  
JCPage.markExactLocation()
```

Contained in:

<flow>, <cell>

Sub-Elements:

none

Attributes:

Name	Definition	Possible Values	Equivalent Method
location-name	Defines a specific name for that location. The name must be unique. Required.	String	n/a
location-type	Defines how the current insertion point will be associated to the location name. Required.	■ current ■ next ■ next-float ■ exact-frame ■ exact-page	n/a
exact-location-x	Defines the x position. Only used when location type is exact-frame or exact-page.	number	n/a
exact-location-y	Defines the y position. Only used when location type is exact-frame or exact-page.	number	n/a
unit	Determines the unit of measurement used when the location-type is exact-frame or exact-page. Default is inches.	■ inches ■ points ■ centimeters	n/a

14.2.35 meta-data

Purpose:

Specifies metadata for the document, such as the author and title.

Equivalent in None
JClass ServerReport:

Contained in: <document-properties>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
author	Specifies who wrote the document.	String	JCDocument. setAuthor()
creator	Specifies who created the document.	String	JCDocument. setCreator()
keywords	Specifies keywords to use when searching among documents.	String	JCDocument. setKeywords()
subject	Specifies what the document is about.	String	JCDocument. setSubject()
title	Specifies the title of the document.	String	JCDocument. setTitle()

14.2.36 new-column

Purpose:

Begins a new column.

Equivalent in JCFLOW.newColumn(), JCFrame.newColumn()
JClass ServerReport:

Contained in: <flow>, <doc-frame>, <header-frame>, <footer-frame>,
<section>

Sub-Elements: none

Attributes: none

14.2.37 new-line

Purpose:

Adds a carriage return.

**Equivalent in
JClass ServerReport:**

JCFlow.newLine(), JCFrame.newLine()

Contained in:

<flow>, <doc-frame>, <header-frame>, <footer-frame>,
<paragraph>, <cell>, <section>

Sub-Elements:

none

Attributes: none

14.2.38 new-page

Purpose:

Begins a new page.

**Equivalent in
JClass ServerReport:**

JCFlow.newPage()

Contained in:

<flow>, <section>

Sub-Elements:

none

Attributes: none

14.2.39 new-paragraph

Purpose:

Begins a new paragraph.

**Equivalent in
JClass ServerReport:**

JCFlow.newParagraph(), JCFrame.newParagraph()

Contained in:

<flow>, <doc-frame>, <header-frame>, <footer-frame>,
<cell>, <section>

Sub-Elements:

none

Attributes: none

14.2.40 new-section

Purpose:
Begins a new section.

**Equivalent in
JClass ServerReport:** JCFLOW.newSection()

Contained in: <flow>

Sub-Elements: none

Attributes: none

14.2.41 output-contents-list

Purpose:
Prints out a previously defined contents list to the front matter flow.

**Equivalent in
JClass ServerReport:** JCFLOW.print(JCContentsList)

Contained in: the front matter <flow>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
name	Specifies a contents list.	name of a contents list	

14.2.42 overflow-rules

Purpose:

Defines the rules used to place overflowing embedded and floated images and components in the document.

Equivalent in

JCDocument.setOverflowRules()

JClass ServerReport:

Contained in:

<document>, <doc-frame>, <header-frame>, <footer-frame>, <flow>, <paragraph>, <section>

Sub-Elements:

none

Attributes:

Name	Definition	Possible Values	Equivalent Method
float-method	Method used to modify overflowed floated item.	<ul style="list-style-type: none">■ truncate■ fit■ fit-maintain-horizontal■ fit-maintain-vertical■ fit-maintain-aspect■ divide-horizontal■ divide-vertical	OverflowRules. setFloatMethod()
float-threshold-measure	The threshold at which a floating item will not appear in the current frame, specified as an absolute position expressed in terms of unit.	floating point number	OverflowRules. setFloatThresholdMeasure()
float-threshold-percentage	The threshold at which a floating item will not appear in the current frame, specified as a percentage.	floating point number	OverflowRules. setFloatThresholdPercentage()

Name	Definition	Possible Values	Equivalent Method
method	Method used to modify overflowed embedded item.	<ul style="list-style-type: none"> ■ truncate ■ fit ■ fit-maintain-horizontal ■ fit-maintain-vertical ■ fit-maintain-aspect ■ divide-horizontal ■ divide-vertical 	OverflowRules. setMethod()
next-item-placement	Specifies the placement of the next item to follow the overflowed item.	<ul style="list-style-type: none"> ■ automatic ■ new-line ■ new-column ■ new-frame ■ new-page ■ next-section ■ threshold 	OverflowRules. setNextItem Placement()
next-item-threshold-measure	The threshold for use with next-item-threshold-placement, specified as an absolute position expressed in terms of unit.	floating point number	OverflowRules. setNextItem ThresholdMeasure()
next-item-threshold-percentage	The threshold for use with next-item-threshold-placement, specified as a percentage.	floating point number	OverflowRules. setNextItem Threshold Percentage()
next-item-threshold-placement	Specifies the placement of the next item to follow the overflowed item when next-item-placement is equal to threshold, and the specified threshold has been reached.	<ul style="list-style-type: none"> ■ automatic ■ new-line ■ new-column ■ new-frame ■ new-page ■ next-section 	OverflowRules. setNextItem Threshold Placement()
placement	Specifies the placement of the overflowed item.	<ul style="list-style-type: none"> ■ automatic ■ new-line ■ new-column ■ new-frame ■ new-page ■ next-section ■ threshold 	OverflowRules. setPlacement()

Name	Definition	Possible Values	Equivalent Method
threshold-measure	The threshold for use with threshold-placement, specified as an absolute position expressed in terms of unit.	floating point number	OverflowRules. setThresholdMeasure()
threshold-percentage	The threshold for use with threshold-placement, specified as a percentage.	floating point number	OverflowRules. setThresholdPercentage()
threshold-placement	Specifies the placement of the overflowed item when placement is equal to threshold, and the specified threshold has been reached.	<ul style="list-style-type: none"> ■ automatic ■ new-line ■ new-column ■ new-frame ■ new-page ■ next-section 	OverflowRules. setThresholdPlacement()
unit	Determines the unit of measure used for float-threshold-measure, next-item-threshold-measure and threshold-measure. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a

14.2.43 page-table

Purpose:

Creates a JCPageTable instance and sends it to the flow – <row> is required.

Equivalent in JCPageTable class

JClass ServerReport:

Contained in: <flow>, <paragraph>, <section>

Sub-Elements: <column-info>, <header-table>, <row>

Attributes:

Name	Definition	Possible Values	Equivalent Method
align	Horizontal alignment of table within current frame.	<ul style="list-style-type: none"> ■ left ■ right ■ center ■ justify ■ none 	JCPageTable. setAlignment()

Name	Definition	Possible Values	Equivalent Method
border-mode	Determines the type of top and bottom row borders to draw at page breaks.	<ul style="list-style-type: none"> ■ external ■ internal 	JCPageTable. setBorderMode()
border-style-name	Border to use for the entire table.	name of a pre-defined draw-style	JCPageTable. setAllBorders()
bottom-border-enabled	Determines whether a bottom border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JCPageTable.set BottomBorder Enabled()
bottom-border-style-name	Border to use for the bottom border of the table.	name of a pre-defined draw-style	JCPageTable. setBottomBorder()
color	Background color of the table.	color String	JCPageTable.set BackgroundColor()
default-bottom-margin	Default bottom margin of a cell in the table expressed in terms of unit.	floating point number	JCPageTable.set DefaultBottom Margin()
default-cell-alignment	Default vertical alignment of text within cells.	<ul style="list-style-type: none"> ■ none ■ center ■ bottom ■ top 	JCPageTable.set DefaultCell Alignment()
default-left-margin	Default left margin of a cell in the table expressed in terms of unit.	floating point number	JCPageTable.set DefaultLeftMargin()
default-right-margin	Default right margin of a cell in the table expressed in terms of unit.	floating point number	JCPageTable. setDefaultRight Margin()
default-text-style-name	Default text style to use for table cells. Text styles that can be used are those that are defined in XML or programmatically, as well as the Default styles defined in the JCTextStyle class.	name of a pre-defined text-style	JCPageTable.set DefaultStyle()
default-top-margin	Default top margin of a cell in the table expressed in terms of unit.	floating point number	JCPageTable. setDefaultTop Margin()

Name	Definition	Possible Values	Equivalent Method
edge-border-enabled	Determines whether an edge border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JCPageTable.setEdgeBorderEnabled()
edge-border-style-name	Border to use for all edge borders in the table.	name of a pre-defined draw-style	JCPageTable.setEdgeBorders()
fit-to-frame	Stretch the width of the table to fit the current frame.	<ul style="list-style-type: none"> ■ true ■ false 	JCPageTable.fitToFrame()
header-border-enabled	Determines whether a header border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JCPageTable.setHeaderBorderEnabled()
header-border-style-name	Border to use between the header and the rest of the table.	name of a pre-defined draw-style	JCPageTable.setHeaderBorder()
horizontal-border-style-name	Border to use for all horizontal borders in the table.	name of a pre-defined draw-style	JCPageTable.setHorizontalBorders()
internal-border-style-name	Border to use for all internal borders in the table.	name of a pre-defined draw-style	JCPageTable.setInternalBorders()
left-border-enabled	Determines whether left border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JCPageTable.setLeftBorderEnabled()
left-border-style-name	Border to use for the left border of the table.	name of a pre-defined draw-style	JCPageTable.setLeftBorder()
num-columns	Number of columns in the table. Required.	integer	JCPageTable constructor
overflow-mode	Determines behavior if table is too wide for the current frame.	<ul style="list-style-type: none"> ■ clip-columns ■ wrap-columns ■ wrap-columns-next 	JCPageTable.setOverflowMode()
right-border-enabled	Determines whether a right border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JCPageTable.setRightBorderEnabled()

Name	Definition	Possible Values	Equivalent Method
right-border-style-name	Border to use for the right border of the table.	name of a pre-defined draw-style	JCPageTable. setRightBorder()
row-column-dominance	Determines which of row or column backgrounds and borders are drawn first.	<ul style="list-style-type: none"> ■ row ■ column 	JCPageTable.setRow ColumnDominance()
table-style-name	Table style to use for this table. Table styles that can be used are those that are defined in XML or programmatically, as well as the Default styles defined in the JCTableStyle class.	name of a pre-defined table-style	JCPageTable constructor
top-border-enabled	Determines whether a top border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JCPageTable.set TopBorder Enabled()
top-border-style-name	Border to use for the top border of the table.	name of a pre-defined draw-style	JCPageTable. setTopBorder()
unit	Determines the unit of measurement used for default-bottom-margin, default-left-margin, default-right-margin, and default-top-margin. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
vertical-border-style-name	Border to use for all vertical borders in the table.	name of a pre-defined draw-style	JCPageTable.set VerticalBorders()

14.2.44 paragraph

Purpose:

Defines the contents of a paragraph – contains text and tags belonging to this paragraph.

Equivalent in

JCFlow.newParagraph(), JCFrame.newParagraph()

JClass ServerReport:**Contained in:**

<flow>, <doc-frame>, <header-frame>, <footer-frame>, <cell>, <section>

Sub-Elements:

text or any [Document Body Tags](#)

Attributes: none

14.2.45 poster

Purpose:

Specifies an optional poster image for an embedded media clip. Using the poster-* attributes, a poster image can be specified as either a fully qualified url, an url relative to a prefix that has been registered with the document, a resource relative to the current servlet context, a resource relative to the classpath or file specified with an absolute path.

Equivalent in

java.awt.Image

JClass ServerReport:**Contained in:**

<embed-media-clip>, <embed-media-clip>

Sub-Elements:

none

Attributes:

Name	Definition	Possible Values	Equivalent Method
poster-class-resource	Specifies the image using a resource relative to the classpath.	file name	n/a
poster-file	Specifies the absolute path to an image file.	a fully qualified file name	n/a
poster-relative-url	Specifies the image using a URL relative to a prefix that has been registered with the document.	a relative URL	n/a

Name	Definition	Possible Values	Equivalent Method
poster-servlet-resource	Specifies the image using a resource relative to the current servlet context.	file name	n/a
poster-url	Specifies the image using a URL.	a fully qualified URL	n/a

14.2.46 row

Purpose:

Defines the contents of a row of a table – `<cell>` is required.

Equivalent in

JCFlowTable.addRow, JCPageTable.addRow

JClass ServerReport:

Contained in:

`<page-table>`, `<flow-table>`, `<header-table>`

Sub-Elements:

`<cell>`

Attributes:

Name	Definition	Possible Values	Equivalent Method
border-style-name	Border style to use for all row borders within the table.	name of a pre-defined draw-style	<ul style="list-style-type: none"> ■ JCFlowTable.Row.setStyle() ■ JCPageTable.Row.setStyle()
color	Background color of the row.	color String	<ul style="list-style-type: none"> ■ JCFlowTable.setBackgroundColor() ■ JCPageTable.setBackgroundColor()
top-border-style-name	Border style to use for the top border of the row.	name of a pre-defined draw-style	<ul style="list-style-type: none"> ■ JCFlowTable.setRowBorder() ■ JCPageTable.setRowBorder()

14.2.47 section

Purpose:

Defines the extent of a section.

**Equivalent in
JClass ServerReport:**

JCFlow.newSection()

Contained in:

<flow>

Sub-Elements:

text or any [Document Body Tags](#)

Attributes: none

14.2.48 space

Purpose:

Sends a non-breaking space to the flow.

**Equivalent in
JClass ServerReport:**

JCFlow.print(" "), JCFrame.print(" ")

Contained in:

<flow>, <paragraph>, <doc-frame>, <header-frame>,
<footer-frame>, <cell>, <section>

Sub-Elements:

none

Attributes: none

14.2.49 tab

Purpose:

Moves the current insertion point to the next tab stop in the current text style.

**Equivalent in
JClass ServerReport:**

JCFlow.tab(), JCFrame.tab()

Contained in:

<flow>, <paragraph>, <doc-frame>, <header-frame>,
<footer-frame>, <cell>, <section>

Sub-Elements:

none

Attributes: none

14.2.50 unsupported-operation

Purpose:
Defined the behavior of JClass ServerReport when an operation is requested that is unsupported for the current output type. For more information on this feature, please refer to [Defining RTF Behavior for Unsupported Features](#), in Chapter 9.
Currently, all features are supported in PDF output; therefore, this will apply only to RTF output. For more information on the features that are not supported in RTF, please refer to Appendix B, [RTF Limitations and Unsupported Features](#).

Equivalent in JClass ServerReport: JCDocument.setUnsupportedOperationAction();

Contained in: <document>, <doc-frame>, <header-frame>, <footer-frame>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
action	<p>Defines the action that will be executed when an unsupported operation is encountered.</p> <p>When do-nothing, no action is taken.</p> <p>When try-related, JClass ServerReport attempts to perform a related action, if one is possible.</p> <p>When throw-exception, JClass ServerReport throws an UnsupportedOperationException each time an unsupported operation is requested. Note: Even if this exception is caught, document creation is interrupted resulting in a failed document. It is not possible to continue.</p>	<ul style="list-style-type: none">■ do-nothing■ try-related■ throw-exception	n/a

14.2.51 use-text-style

Purpose:
Defines the extent of the specified text style – contains tags and text that will use this text style.

Equivalent in JClass ServerReport: JCFLOW.setCurrentTextStyle()

Contained in: <flow>, <paragraph>, <section>, <doc-frame>, <header-frame>, <footer-frame>, <cell>

Sub-Elements: text or any Document Body Tags

Attributes:

Name	Definition	Possible Values	Equivalent Method
name	Name of the text style. Required.	name of a pre-defined text-style	n/a

14.3 Page Template

Page templates are contained in the *ServerReportPageTemplate.dtd*. The page template elements are described in Chapter 7, “Defining Page Templates” under Section 7.1, [Page Template Formats](#).

14.4 Text Styles

Text Styles are contained in *ServerReportTextStyle.dtd* and *JCServerReport.dtd*.

14.4.1 text-styles

Purpose:

Defines a collection of JClass ServerReport text styles.

Equivalent in

none

JClass ServerReport:

Contained in:

top-level tag if using *ServerReportTextStyle.dtd*; <document> if using *JCServerReport.dtd*

Sub-Elements:

<text-style>

Attributes: none

14.4.2 text-style

Purpose:

Defines a JClass ServerReport text style.

Equivalent in

JCTextStyle class

JClass ServerReport:

Contained in:

<text-styles> tag

Sub-Elements:

<tab-stop>

Attributes:

Name	Definition	Possible Values	Equivalent Method
align	Horizontal alignment of text. Default is left.	■ left ■ right ■ center ■ justify ■ none	JCTextStyle. setAlignment()
baseline-offset	Specifies superscript and subscript text. Default is none.	■ none ■ subscript ■ superscript	JCTextStyle. setBaselineOffset()
background	Color of the background behind the text.	color String	JCTextStyle. setBackground()
color	Color of text.	color String	JCTextStyle. setColor()

Name	Definition	Possible Values	Equivalent Method
font-family	Name of the font; used in conjunction with the font-style, point-size, and underline attributes.	String	JCTextStyle. setFontFamily()
font-style	Style of the font; used in conjunction with the font-family, point-size, and underline attributes.	<ul style="list-style-type: none"> ■ plain ■ bold ■ italic ■ bold-italic 	JCTextStyle. setFontStyle()
indent-left	Amount of space a line will be indented on its left side expressed in terms of unit.	floating point number	JCTextStyle. setLeftIndent()
indent-paragraph	Amount of space a paragraph will be indented on its first line expressed in terms of unit.	floating point number	JCTextStyle. setParagraphIndent()
indent-right	Amount of space a line will be indented on its right side expressed in terms of unit.	floating point number	JCTextStyle.set RightIndent()
line-spacing	Spacing between lines of text as a multiple of text height.	floating point number	JCTextStyle.set LineSpacing()
name	Name of this text style. Required.	String	JCTextStyle. setName()
paragraph-spacing	Spacing between paragraphs as a multiple of text height.	floating point number	JCTextStyle. setParagraphSpacing()
point-size	Point size of the font; used in conjunction with the font-name, font-style, and underline attributes.	integer	JCTextStyle. setPointSize()
subscript-ratio	Ratio of subscript or superscript text to normal text.	floating point number	JCTextStyle. setSubscriptRatio()
underline	Underline value of the font used for font-name, font-style, and point-size. Default is none.	<ul style="list-style-type: none"> ■ single ■ none 	JCTextStyle.set Underlining()

Name	Definition	Possible Values	Equivalent Method
unit	Determines the unit of measurement for indent-left, indent-right, and indent-paragraph. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a
wrap-mode	<p>Determines if the text wraps.</p> <p>When no-wrap, any text that does not fit on a line is discarded. Nothing new will print until the flow is advanced manually using new-line, new-paragraph, etc.</p>	<ul style="list-style-type: none"> ■ wrap ■ no-wrap 	JCTextStyle. setWrapMode()

14.4.3 tab-stop

Purpose:

Specifies a tab stop.

Equivalent in JCTab class

JClass ServerReport:

Contained in: <text-style> tag

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
align	Horizontal alignment of text with respect to the tab. Default is left.	<ul style="list-style-type: none"> ■ left ■ right ■ center ■ decimal 	JCTab. setTabAlignment()
fill	Character used to fill the space until the next tab. Default is none.	<ul style="list-style-type: none"> ■ none ■ dots ■ underline 	JCTab.setTabFill()
position	Tab position relative to the left side of the frame expressed in terms of unit. Required.	floating point number	JCTab.setPosition()

Name	Definition	Possible Values	Equivalent Method
unit	Determines the unit of measurement used for position. Default is inches.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a

14.5 Draw Styles

Draw Styles are contained in *ServerReportDrawStyle.dtd* and *JCServerReport.dtd*.

14.5.1 draw-styles

Purpose:

Defines a collection of JClass ServerReport draw styles.

Equivalent in JClass ServerReport: none

Contained in: top-level tag if using *ServerReportDrawStyle.dtd*; <document> if using *JCServerReport.dtd*

Sub-Elements: <draw-style>

Attributes: none

14.5.2 draw-style

Purpose:

Defines a JClass ServerReport draw style, which can be used with properties of other tags to define borders.

Equivalent in JClass ServerReport: JCDrawStyle class

Contained in: <draw-styles>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
color	Color of the draw style.	color String	JCDrawStyle.setForegroundColor()

Name	Definition	Possible Values	Equivalent Method
dash-length	Length of dashes if using dashed lines expressed in terms of unit.	floating point number	JCDrawStyle. setDashLength()
fill-rule	Rule used when deciding which areas of a shape to fill.	<ul style="list-style-type: none"> ■ even-odd ■ non-zero-winding 	JCDrawStyle. setFillRule()
line-spacing	Spacing between lines drawn expressed in terms of unit.	floating point number	JCDrawStyle. setLineSpacing()
line-type	Type of line to draw. Default is plain.	<ul style="list-style-type: none"> ■ none ■ blank ■ broken ■ dashed ■ double ■ plain ■ regular ■ single 	JCDrawStyle. setLineType()
line-width	Width of the line to draw expressed in terms of unit.	floating point number	JCDrawStyle. setLineWidth()
name	Name of the draw style. Required.	String	JCDrawStyle. setName()
unit	Determines the unit of measurement used for line-width, line-spacing, and dash-length. Default is points.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a

14.6 Symbol Styles

Symbol Styles are contained in *ServerReportSymbolStyle.dtd* and *JCServerReport.dtd*.

14.6.1 symbol-styles

Purpose:
Defines a collection of JClass ServerReport symbol styles.

Equivalent in JClass ServerReport: none

Contained in: top-level tag if using *ServerReportSymbolStyle.dtd*; <document> if using *JCServerReport.dtd*

Sub-Elements: <symbol-style>

Attributes: none

14.6.2 symbol-style

Purpose:
Defines a JClass ServerReport symbol style for use with bulleted lists.

Equivalent in JClass ServerReport: com.klg.jclass.sreport.JCSymbolStyle class

Contained in: <symbol-styles>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
color	Color of the symbol.	color String	JCSymbolStyle. setColor()
name	Name of the symbol style. Required.	String	JCSymbolStyle. setName()
shape	Specifies the shape of the symbol. Default is none.	■ none ■ dot ■ box ■ triangle ■ diamond ■ star ■ vert_line ■ horiz_line ■ cross ■ circle ■ square ■ rectangle	JCSymbolStyle. setShape()

Name	Definition	Possible Values	Equivalent Method
size	Specifies the size of the symbol expressed in terms of unit.	floating point number	JCSymbolStyle. setSize()
unit	Determines the unit of measurement used for size. Default is points.	<ul style="list-style-type: none"> ■ inches ■ points ■ centimeters 	n/a

14.7 Table Styles

Table Styles are contained in *ServerReportTableStyle.dtd* and *JCServerReport.dtd*.

14.7.1 table-styles

Purpose:

Defines a collection of JClass ServerReport table styles.

Equivalent in JClass ServerReport: none

Contained in: top-level tag if using *ServerReportTableStyle.dtd*; <document> if using *JCServerReport.dtd*

Sub-Elements: <table-style>

Attributes: none

14.7.2 table-style

Purpose:
Defines a JClass ServerReport table style, which can later be referenced in a <page-table> or <flow-table> tag.

Equivalent in JClass ServerReport: JCTableStyle class

Contained in: <table-styles>

Sub-Elements: <alternate>

Attributes:

Name	Definition	Possible Values	Equivalent Method
background	Background color of the table.	color String	JCTableStyle. setBackground()
bottom-border	Border style to use for the bottom border of the table.	name of a pre-defined draw-style	JCTableStyle. setBottomBorder()
bottom-border-enabled	Determines whether bottom border is used.	■ true ■ false	JCTableStyle.set BottomBorder Enabled()
column-border	Border style to use for all columns in the table.	name of a pre-defined draw-style	JCTableStyle. setColumnBorder()
first-column-special	Determines whether the first column is different than the other columns.	■ true ■ false	JCTableStyle constructor
header-border	Border style to use for the header border of the table.	name of a pre-defined draw-style	JCTableStyle. setHeaderBorder()
header-border-enabled	Determines whether header border is used.	■ true ■ false	JCTableStyle.set HeaderBorder Enabled()
header-style-name	Name of the defined table style to use for the header table.	name of a pre-defined table-style	JCTableStyle. setHeaderStyle()
left-border	Border style to use for the left border of the table.	name of a pre-defined draw-style	JCTableStyle. setLeftBorder()

Name	Definition	Possible Values	Equivalent Method
left-border-enabled	Determines whether left border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JTableStyle.setLeftBorderEnabled()
name	Name of this table style. Required	String	JTableStyle.setName()
right-border	Border style to use for the right border of the table.	name of a pre-defined draw-style	JTableStyle.setRightBorder()
right-border-enabled	Determines whether right border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JTableStyle.setRightBorderEnabled()
row-border	Border style to use for all row borders of a table.	name of a pre-defined draw-style	JTableStyle.setRowBorder()
show-first-row-top-border	Set whether to show the top border of the first row of a table.	<ul style="list-style-type: none"> ■ true ■ false 	JTableStyle.setShowFirstRowTopBorder()
text-style-name	Name of the text style to use for this table. Text styles that can be used are those that are defined in XML or programmatically, as well as the Default styles defined in the JCTextStyle class.	name of a pre-defined text-style	JTableStyle.setTextStyle()
top-border	Border style to use for the top border of the table.	name of a pre-defined draw-style	JTableStyle.setTopBorder()
top-border-enabled	Determines whether top border is used.	<ul style="list-style-type: none"> ■ true ■ false 	JTableStyle.setTopBorderEnabled()

14.7.3 **alternate**

Purpose:
Specifies an alternating sequence of colors for a table background.

Equivalent in JClass ServerReport: JCA**l**ternate class

Contained in: <table-style>

Sub-Elements: none

Attributes:

Name	Definition	Possible Values	Equivalent Method
color1	First color in the alternating sequence. Required.	color String	n/a
color2	Second color in the alternating sequence. Required.	color String	n/a
row-oriented	Specifies whether the alternating sequence of colors will alternate rows or columns. Required.	■ true ■ false	n/a

Part ***III***

*Using
JClass
ServerReport
Designer*

JClass ServerReport Designer

[Background Information](#) ■ [Launching JClass ServerReport Designer](#)
[The Tool Bar](#) ■ [The File Menu](#) ■ [The Help Menu](#)

JClass ServerReport Designer is an application that helps you generate tags for XML output through a GUI. You are able to create page and document layouts using a WYSIWYG interface. The XML tags are automatically created for you -- without your delving into code.

JClass ServerReport Designer outputs XML in the standard JClass ServerReport tag formats and creates both simple and full page template, text style, draw style, and table style tag objects for you.

Note: JClass ServerReport Designer requires JDK 1.5 or later.

15.1 Background Information

The *jdom.jar* is shipped with JClass ServerReport, and is automatically installed in the *JCLASS_SERVER_HOME/lib* directory. This JAR is required for JClass ServerReport Designer, and removing it will cause the designer to cease functioning.

15.2 Launching JClass ServerReport Designer

Unless the installer is specifically instructed not to, JClass ServerReport Designer is automatically installed with the JClass ServerReport installation. To launch the designer, simply double click the *jesreport-designer.bat* or *jesreport-designer.sh* file, located in your *JCLASS_SERVER_HOME/bin* directory. The JClass ServerReport Designer splash screen is displayed.

Note: On Windows systems, you can select **JClass ServerReport Designer** from the **Start > Programs > JClass ServerViews > JClass ServerReport** menu.

The Welcome window

Once you have successfully launched JClass ServerReport Designer, the *Welcome* window allows you to quickly start designing your document. From this menu, you can choose to open

an existing file, create a new document, create document styles, or open the last file that was worked on, simply by clicking the appropriate button.

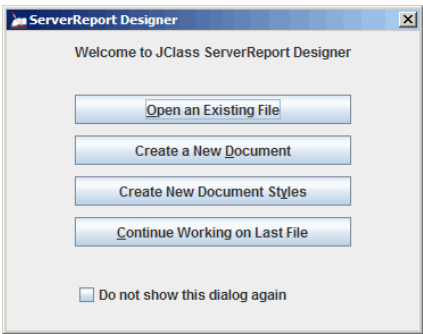






Figure 59 The Welcome window.

Note: Selecting the **Do not show this dialog again** checkbox will stop the *Welcome* window from appearing. You can also edit this property in your preferences; for more information, see [Selecting Show options](#) in Section 15.4.11, [Setting your preferences](#).

15.3 The Tool Bar

JClass ServerReport Designer comes equipped with a tool bar for easy access to common functions.

Button	Description
	Creates a new document.
	Opens a document.
	Saves a document.
<div>Preview Document</div>	Previews a document, page template, or a text style.
	Launches the JClass ServerReport Designer help.

15.4 The File Menu

The **File** menu is your basic starting point for JClass ServerReport Designer. This menu will help you open files, create new documents, and save your changes, and also has a history list of the last four documents that have been opened in JClass ServerReport Designer.

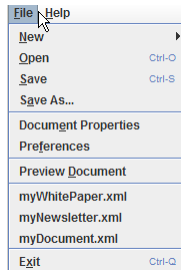


Figure 60 The **File** Menu in JClass ServerReport Designer.

The following procedures can be accomplished through the **File** menu:

File Menu

- Designing a new page template
 - Designing a new document
 - Designing new document styles
 - Designing a new text style
 - Designing a new draw style
 - Designing a new table style
 - Opening a ServerReport XML file
 - Saving a document or document styles
 - Specifying the file encoding type
 - Specifying the page template type
 - Setting your preferences
 - Previewing a document; see [Preview](#), in Chapter 16
-

15.4.1 Designing a new page template

Note: If you already have a file open in JClass ServerReport Designer with unsaved changes, you will be prompted to save the current document.

- From the **File** menu, select **New**, then **Page Template**.

15.4.2 Designing a new document

Note: If you already have a file open in JClass ServerReport Designer with unsaved changes, you will be prompted to save the current document.

- From the **File** menu, select **New**, then **Document**.

15.4.3 Designing new document styles

Note: If you already have a file open in JClass ServerReport Designer with unsaved changes, you will be prompted to save the current document.

- From the **File** menu, select **New**, then **Document Styles**.

15.4.4 Designing a new text style

Note: If you already have a file open in JClass ServerReport Designer with unsaved changes, you will be prompted to save the current document.

- From the **File** menu, select **New**, then **Text Style**.

15.4.5 Designing a new draw style

Note: If you already have a file open in JClass ServerReport Designer with unsaved changes, you will be prompted to save the current document.

- From the **File** menu, select **New**, then **Draw Style**.

15.4.6 Designing a new table style

Note: If you already have a file open in JClass ServerReport Designer with unsaved changes, you will be prompted to save the current document.

- From the **File** menu, select **New**, then **Table Style**.

15.4.7 Opening a ServerReport XML file

If the file you would like to open appears in the **File** menu's history list, select it from that list. Otherwise, you will have to navigate to find the file.

Note: If you already have a file open in JClass ServerReport Designer with unsaved changes, you will be prompted to save the current document.

1. From the **File** menu, select **Open**. The *Open* window will appear.
2. In the *Open* window, navigate to the location of the file you would like to open.
3. Either double click the file name in the *Open* window, or highlight it and click the **Open** button.

15.4.8 Saving a document or document styles

You have the option to save your work as either a Document or as Document Styles. A file that is saved as Document Styles can be loaded into JClass ServerReport easily, but contains only style tags (for more information, please refer to the XML chapter in the JClass ServerReport documentation). A file saved as a Document will generate page-template, text-style, draw-style, and table-style tags. Document tags, doc-frame tags and flow tags are also generated for quick content insertion in a text or XML editor.

1. From the **File** menu, select **Save As**.
2. Select either the **Document** or **Document Styles** radio button to determine how the information is saved. When saving as **Document Styles**, only styles are saved; no XML declarations, document tags, doc-frame tags, or flow tags are generated.
3. Enter a name for the file in the *File Name* text box.
4. Navigate to the location where you would like to save the file.
5. Click the **Save** button.

15.4.9 Specifying the file encoding type

You have the option to specify the encoding type for the files you save in JClass ServerReport Designer. You can choose from **UTF-8**, **ISO-8859-1**, or **US-ASCII**.

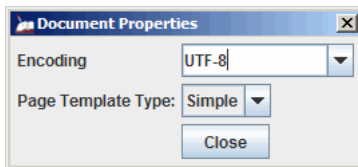


Figure 61 The Document Properties window.

Note: Advanced users may specify their own file encoding type. However, we caution those users to do so carefully. Specifying an incorrect encoding type can result in the inability to write the XML file, or the production of output that cannot be parsed.

1. From the **File** menu, select **Document Properties**.
2. In the *Document Properties* window, type the name of the encoding type, or select it from the drop-down list.
3. Click the **Close** button.

15.4.10 Specifying the page template type

JClass ServerReport Designer offers two different page template types: Simple and Full.

- **Simple:** A simple page template allows you define a basic page, including header, footer, body frame, gutters, and margins. Control over the layout is limited; most frame position and size information is inferred from other frames on the same page or from default values. You can not add extra frames. This format can be applied to PDF or RTF documents.
 - **Full:** A full page template allows you to fully specify the layout of a page. This format is used strictly for PDFs.
1. From the **File** menu, select **Document Properties**.
 2. In the *Document Properties* window, select the page template type from the drop-down list.
 3. The *Create a New Page Template* window will open, warning that all changes will be lost if the page template type is changed. To proceed, click the **OK** button.
 4. Click the **Close** button.

15.4.11 Setting your preferences

JClass ServerReport Designer allows you to customize your preferred home directory and Preview Viewer, import these settings from a file, and export the settings to a file. This allows you to save your preferences outside of the system variables for future use.

To get to the *Edit Preferences* window, select **Preferences** from the **File** menu.

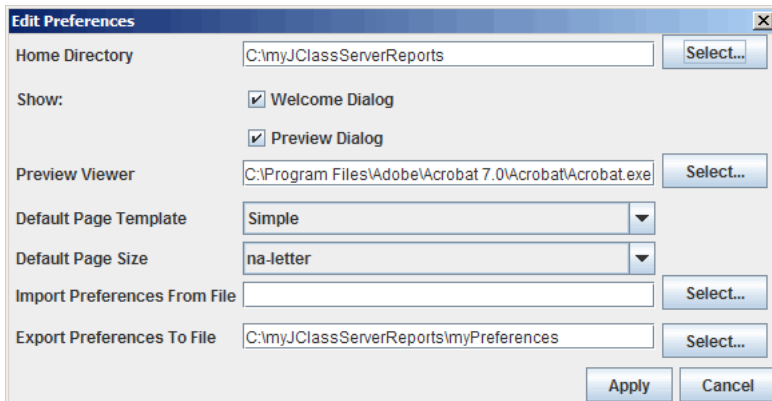


Figure 62 The *Edit Preferences* window.

Selecting Show options

JClass ServerReport Designer allows you to select whether or not the welcome and preview dialogs should be shown to the user.

- To show either the welcome dialog or the preview dialog, select the corresponding checkbox; otherwise, deselect it.

Choosing home directory

The home directory identifies which directory will be used as JClass ServerReport Designer's default directory.

1. To set your home directory, click the appropriate **Select...** button on the *Edit Preferences* window.
2. In the *Choose Home Directory* window, navigate to and select the directory you would like to set as your home directory.
3. Click the **Choose Home Directory** button to return to the *Edit Preferences* window.
4. Click the **Apply** button.

Choosing the default page template

The default page template option determines if the JClass ServerReport Designer default page template is Simple or Full.

1. To set the default page template, click the appropriate template from the **Default Page Template** drop-down list in the *Edit Preferences* window.
2. Click the **Apply** button.

Choosing the default page size

JClass ServerReport Designer is equipped with many different standard page sizes from which to chose the default.

1. To set the default page size, click the appropriate size from the **Default Page Size** drop-down list in the *Edit Preferences* window.
2. Click the **Apply** button.

Choosing the Preview Viewer

This setting will identify the default viewer that will be used when previewing your work.

1. To set your default viewer, click the appropriate **Select...** button on the *Edit Preferences* window.
2. In the *Select your preview viewer executable* window, navigate to and select the desired viewer.

Note: You must select a PDF or RTF executable. For example, *acroread.exe*. This value needs to be changed each time the preview document format is changed.

3. Click the **Select your preview document viewer executable** button to return to the *Edit Preferences* window.
4. Click the **Apply** button.

Importing from a file and exporting to a file

Preferences are saved in your system variables. JClass ServerReport Designer allows you to import a file containing information about saved preferences to update the current profile, or to export your preferences to a file for future use.

- To import preferences from a file, click the appropriate **Select...** button on the *Edit Preferences* window. In the *Open* window, navigate to and select the file you would like to import and click the **Open** button. In the *Edit Preferences* window, click the **Apply** button.
- To export preferences to a file, click the appropriate **Select...** button on the *Edit Preferences* window. In the **Save** window, navigate to and select the location where you would like to export the file and click the **Save** button. In the *Edit Preferences* window, click the **Apply** button.

15.5 The Help Menu

The **Help** menu is your gateway to the JClass ServerReport Designer help, as well as to the JClass ServerReport Designer's *About* window.

To access either feature, simply select it from the **Help** menu list.

Using JClass ServerReport Designer

[Basic Steps for Creating a ServerReport Document](#) ■ [Common Procedures](#) ■ [Preview](#)

There are some JClass ServerReport Designer procedures that you will use often: [Common Procedures](#).

16.1 Basic Steps for Creating a ServerReport Document


JClass ServerReport Designer will allow you to create a page template, text styles templates, as well as a whole document. A document will generate document tags, doc-frame tags, draw styles, table styles, and flow tags for quick content insertion in a text or XML editor, as well as all the information for page and text style templates.

Though it is not necessary for you to follow these steps, in general this is what you will need to do to create a complete document.

Step	See
1. Create a document	Designing new document styles
2. Add pages and customize	Designing a new page template and One Page Tab
3. Add frames to each of the pages and customize Note: This is only for templates taking advantage of the Full page template.	Creating a page, frame, text style, tab, draw style, or table style
4. Create the flow	Changing a frame's flow and All Pages Tab
5. Define text styles, draw styles, and table styles	Text Styles, Draw Styles, and Table Styles

16.2 Common Procedures


16.2.1 Creating a page, frame, text style, tab, draw style, or table style

Click the  button to create a new item.

- To create a new page, the button is beside the *Pages* list, on the **Page** tab. The new page will have the name “page*”, where * represents a String to provide the page with an original name.
- To create a new frame, ensure that you are working with the “full” page template. Click the button beside the *Flow Frames* list, on the **Page** tab. The new frame will have the name “frame*” where * represents a String that will provide the frame with an original name.
- To create a new text style, the button is beside the *Text Styles* list, on the **Text Style** tab. The new text style will have the name “text*”, where * represents a String to provide the text style with an original name.
- To create a new tab, the button is beside the *Tab* list, on the **Text Style** tab. The new tab’s name will reflect its position and unit of measurement, and is automatically generated.
- To create a new draw style, the button is beside the *Draw Styles* list, on the **Draw Styles** tab. The new draw style will have the name “draw*”, where * represents a String to provide the draw style with an original name.
- To create a new table style, the button is beside the *Table Styles* list, on the **Table Style** tab. The new table style will have the name “table*”, where * represents a String to provide the draw style with an original name.

16.2.2 Cloning a page, frame, text style, tab, draw style, or table style

Note: When you clone a page or frame, you are not only copying all of its properties, but also the page or frame’s flow.


1. Highlight the name of the page, frame, text style, tab, draw style, or table style that you would like to clone.
2. Click the  button. The cloned item will have the prefix “CopyOf” to distinguish it from the original item. Subsequent clones will have a hash symbol (#) appended.

16.2.3 Deleting a page, frame, text style, tab, draw style, or table style

Note: There must always be at least one item in the *Pages* list, the *Frames* list, the *Text Styles* list, and the *Draw Styles* list. For that reason, if you attempt to delete the last item in one of these lists, you will receive an error message.

Because JClass ServerReport Designer does not require any tabs, you can delete all of the tabs that appear in the *Tab* list.

1. Highlight the name of the page, frame, text style, tab, draw style, or table style that you would like to delete.

2. Click the  button.
3. In the *Confirm Delete* dialog box, click the **OK** button.

16.2.4 Naming a page, frame, text style, draw style, or table style

1. In the *Name* text box, delete the current text and type in the desired page, frame, text style, draw style, or table style name.
2. To commit the change, click anywhere outside of the *Name* text box or hit the return key.
To cancel the change, hit the *escape* key before committing the change.

Note: There are several stock text, draw, and table styles. While JClass ServerReport Designer allows styles you define to have the same name as the stock styles, doing so will result in losing the stock style definition; therefore, it is suggested that you use different names for the styles you define than those applied to the stock styles. The following is a list of the different stock style names:

Stock Text Styles	Stock Draw Styles	Stock Table Styles
<ul style="list-style-type: none"> ■ Bold ■ Bold Italic ■ Code ■ CodeIndented ■ H1 ■ H2 ■ H3 ■ H4 ■ H5 ■ H6 ■ H7 ■ Heading ■ HeadingBold ■ Indented ■ Normal ■ Plain ■ WhiteBold ■ default header ■ default text 	<ul style="list-style-type: none"> ■ dashed line ■ default 1pt line ■ default 2pt line ■ default blank ■ default double line ■ default line 	<ul style="list-style-type: none"> ■ default ■ Style 0 ■ Style 1 ■ Style 2 ■ Style 3 ■ Style 4 ■ Style 5 ■ Style 6 ■ Style 7 ■ Style 8 ■ Style 9 ■ Style 10 ■ Style 11 ■ Style 12 ■ Style 13 ■ Style 14 ■ Style 15 ■ Style 16 ■ Style 17

For more information on the different stock text styles, see [Predefined JCTextStyle Objects](#), in Chapter 2. For more information on the stock draw styles, see [Stock Draw Styles](#), in Chapter 3. For more information on the different stock table styles, see [Table Styles](#), in Chapter 5.

16.2.5 Changing the unit of measurement

You can change the unit of measurement for a page, frame, text style indentation, tab, or draw style. Changing the unit of measurement does not alter the size of the page or frame. Measurements will automatically be adjusted.

1. Click the *Units* drop-down list.
2. Select either **Inches**, **Points** or **Centimeters**.

16.2.6 Changing the size of a page or frame

1. Highlight the text inside either the *Width* or *Height* text box, depending on which you would like to alter, and delete it.
2. Enter the size in the text box.
Note: Input must be a valid positive number for your locale.
3. To commit the change, click anywhere outside the text box or hit the return key. To cancel the change, hit the *escape* key before committing the change.

Note: For frames, you can also use the resize handles to alter its size in the *Edit* window.

16.2.7 Changing the location of the page or frame

Note: Changing the location of a page dictates where the page will print on the selected output medium. For example, if you intend to print a document on paper that measures 8.5"x11" (standard page size), and your page template specifies that the pages are 6.5"x7", you can print in the center of the page by setting the X coordinate to 1" (which will leave a one inch margin at the left of the page) and the Y coordinate to 2" (which will leave a two inch margin at the top of the page).

1. Highlight the text inside either the *X* or *Y* text box, depending on which you would like to alter, and delete it.
2. Enter the coordinates in the text box.
Note: Input must be a valid positive number for your locale.
3. To commit the change, click anywhere outside the text box or hit the return key. To cancel the change, hit the *escape* key before committing the change.

Note: For frames, you can also click and drag it to where it should reside in the page in the *Edit* window.

16.2.8 Changing the color

To change the color of a page, frame, border, font for a text style, draw style line, or table style, you can either type the name or RGB value into the text field, or click the **Change** button to change the color dynamically.

Note: The color name or RGB value will automatically appear in the text field when a color has been selected.

1. Click the **Change** button next to the *Color* display. The *Choose a Color* window will appear.
2. There are three ways to choose a page color:
 - Select the **Swatches** tab to choose a color from a standard series of color swatches. From the display, click on the color of your choice. You will notice that previous color selections will appear in the Recent swatches.

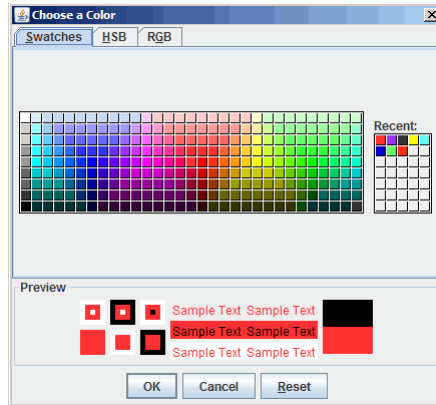


Figure 63 The Swatches tab in the Choose a Color window.

- Select the **HSB** tab to choose a color by hue, saturation or brightness. To adjust colors one property at a time, select either the radio button next to **H** to select a color by hue, next to **S** to select a color by saturation, or next to **B** to select a color by brightness. You can either click inside the color box to select a color or drag the arrow next to the color bar to adjust it. At all times, you can click the arrows next to the HSB values to increase or decrease their value, or enter in a number to replace the value. The corresponding RGB values of the selected HSB color are displayed in the *RGB* boxes below the HSB radio buttons.

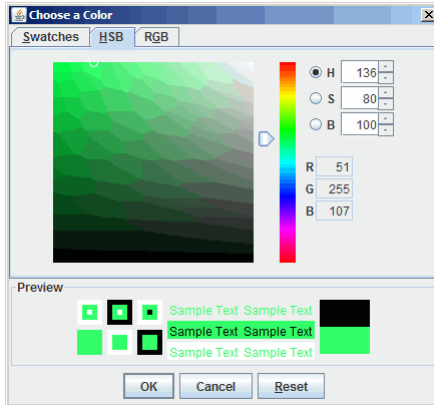


Figure 64 The HSB tab in the Choose a Color window.

- Select the **RGB** tab to choose a color based on its red, green and blue values. To do so, you can either input the color values into the text boxes associated with each color, click the arrows next to the associated values to increase or decrease the color value, or drag the associated arrows to the left or right to decrease or increase the amount of the associated color in the overall selected shade.

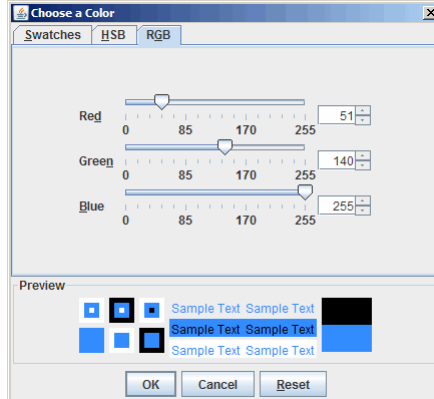


Figure 65 The RGB tab in the Choose a Color window.

Note: Regardless of which tab you use to choose a color, there is a *Preview* section at the bottom of the *Choose a Color* window that will display the color that is currently selected. This allows you to see what the color looks like next to black, white, or grey, as well as what text in that color will look like over white and grey, and what black text will look like over the color.

3. To commit your color selection, click the **OK** button.

If at any time you wish to return to color that was selected when you opened the *Choose a Color* window, click the **Reset** button. To exit the *Choose a Color* window at any time without committing your selection, click the **Cancel** button.

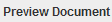
16.3 Preview

JClass ServerReport Designer allows you to preview your work in a PDF file.

Font styles that are defined in a text style, but are not available in the PDF viewer, will be displayed as a default font.

Note: You must set your preferences in order for the Preview feature to function properly. For more information on the JClass ServerReport Designer preferences, see Section 15.4.11, [Setting your preferences](#).

16.3.1 Previewing a document

1. To preview a document, either select **Preview** then **Document** from the **File** menu, or click the  button on the tool bar.
2. From the *Preview Document* window.
3. Select the type of file to preview from the **Preview Format** drop-down list.
4. Select the content to flow into the file.

The *Preview Document* window allows you to select the data you would like to use to display your work, as well as the option to export the output. By selecting the corresponding radio button, you can either use the current document's content, content from a sample file, or provide your own content file to preview the document.

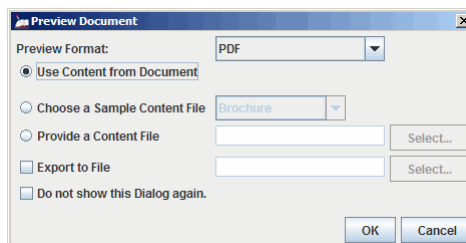


Figure 66The *Preview Document* window.

If you choose to use sample content, you have the choice of previewing a brochure, a short story, a novel, or a poem.

If you are going to provide your own content file, the file must be an XML file that is properly formatted with document tags. Optionally, it may include flow tags and


appropriate doc-frame tags; this file should not have any page template or text style tags, as they will be ignored.

5. To export the file, select the **Export to File** checkbox.
6. Click the **OK** button to launch your PDF viewer and display the document.

Note: The *Preview Document* window has a **Do not show this Dialog again** checkbox. Selecting the checkbox will preview documents automatically, using content from the document as it's default. You can also edit this property in your preferences; for more information, see [Selecting Show options](#) in Chapter 15, [JClass ServerReport Designer](#).

16.3.2 Previewing all

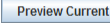
There are two different methods that can be used to preview all page templates, text styles, draw styles, or table styles:

- From the **File** menu, select **Preview** then **All Page Templates**, **All Text Styles**, **All Draw Styles**, or **All Table Styles**.
- OR
- Click the  button, either on the **Page Template** tab below the *Pages* list, on the **Text Styles** tab below the *Text Styles* list, on the **Draw Style** tab below the *Draw Styles* list, or on the **Table Styles** tab below the *Table Styles* list.

This will launch the PDF viewer, opening a file that displays either all of the page templates or all of the text styles.

16.3.3 Previewing the current page template or a current style

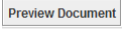
JClass ServerReport Designer allows you to preview one page template or one text style at a time, as long as it is the style currently selected. There are two different methods that can be used to preview the current page template or text style:

- From the **File** menu, select **Preview** then **Current Page Template**, **Current Text Style**, **Current Draw Style**, or **Current Table Style**.
- OR
- Click the  button, either on the **Page Template** tab below the *Pages* list, on the **Text Styles** tab below the *Text Styles* list, on the **Draw Styles** tab below the *Draw Styles* list, or on the **Table Style** tab below the *Table Styles* list.

This will launch the PDF viewer, opening a file that displays either the current page templates or the current text style.

16.3.4 Exporting the PDF

When previewing a document, you can export the PDF file. This option is only available when previewing a document; you cannot export the PDF files that are displayed when previewing only page templates or text styles.

1. Either select **Preview** then **Document** from the **File** menu, or click the  button on the tool bar, to open the Preview Document window.
2. From the *Preview Document* window, select the content to flow into the file. For more information, see [Previewing a document](#).
3. Select the **Export to File** check box. Enter a location, file name and extension in the corresponding field, or click the **Choose File...** button to navigate to the location.
4. Click the **OK** button to launch your PDF viewer and display the document.

Page Templates

One Page Tab ■ *All Pages Tab*

JClass ServerReport Designer allows you to design your own page templates. A Page Template can be loaded into JClass ServerReport easily, but contains only page-template tags

While designing page templates, you will discover that JClass ServerReport Designer has two different tabs, each providing you with a different view of the page template's design.

- The **One Page** tab provides you with a close-up of one page at a time, allowing you to pay close attention to details pertaining to frames and smaller design elements. This allows you to view the page as a single unit and make changes to its design and structure.
- The **All Pages** tab displays all of the pages that are a part of the current document, creating a visual representation of the relationships between the pages and displaying the document as a whole. This overhead view of the document as a whole allows for a space where you can visualize and apply document changes.

To change your view, simply select either the **One Page** or the **All Pages** tab at the bottom of the ServerReport Designer window.

Note: You do not have to save your changes before changing the tab. When you do save, all the changes you made in both will be saved, regardless of which screen you are presently viewing.

17.1 One Page Tab

The **One Page** tab provides you with a visual representation of one page layout at a time. This allows you to see where the frames are placed on the page, and gives you an accurate portrayal of what the page looks like. From this you can edit both frame and page properties.

The **One Page** tab displays different content depending on the type of page template (Simple or Full) currently being designed. Recall that you select the type of page template in the Document Properties dialog, available from the File menu.

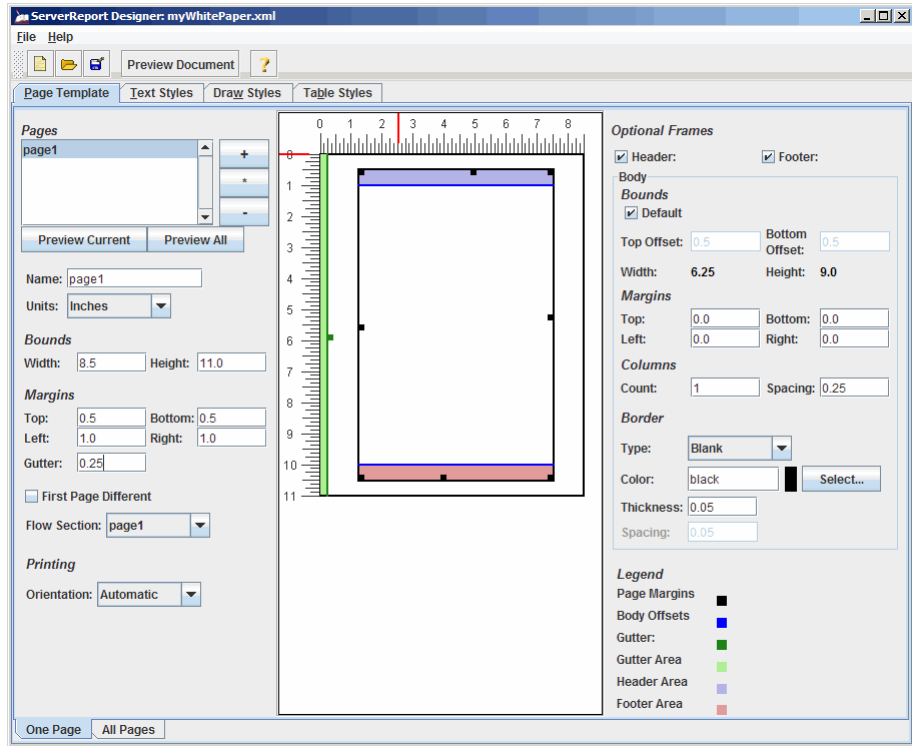


Figure 67 The One Page tab for Simple Page Templates in JClass ServerReport Designer.

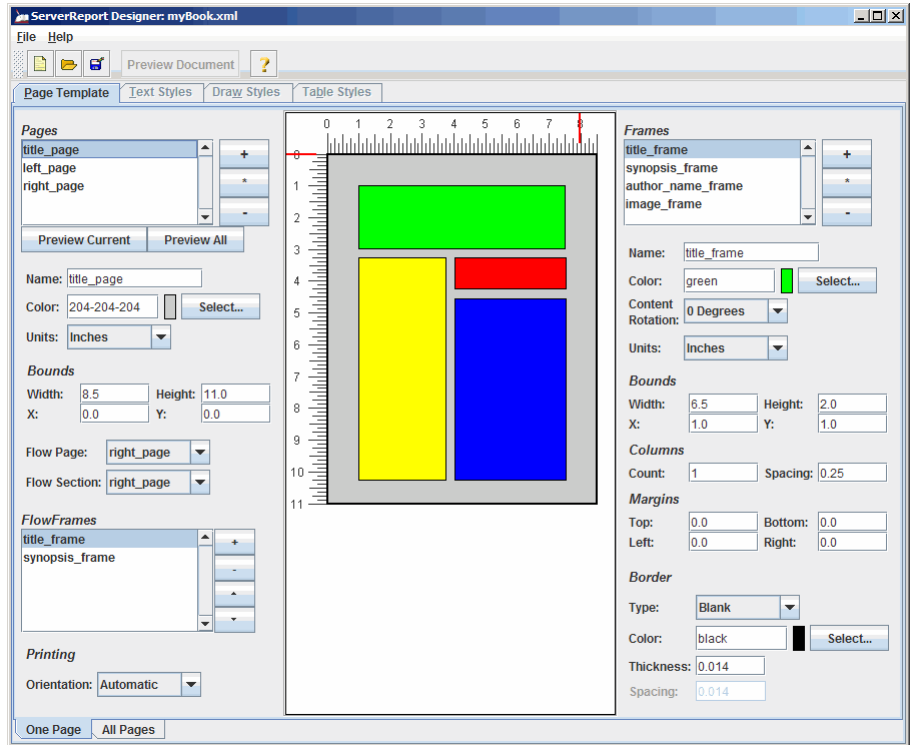


Figure 68 The One Page tab for Full Page Templates in JClass ServerReport Designer.

JClass ServerReport Designer allows you to have multiple pages in a document or page template. This facilitates the creation of larger documents. For example, you can have a different style of page for your title page, table of contents, body, and index pages.

To change any page attributes, you must first ensure that the page you would like to change is selected in the *Pages* list. Next, change the attributes in the *Current Page* section of the JClass ServerReport Designer.

ServerReport Designer also allows you to have multiple frames in a page, and helps you to design them. Frames are useful to segregate different types of information on a page. For example, you may want to have header, body, and footer frames on a page.

To change any frame attributes, you must first ensure that the frame you would like to change is selected in the *Frames* list. Next, change the attributes in the *Current Frame* section of the JClass ServerReport Designer GUI.

Some procedures that involve pages can be performed on any screen. Please refer to Section 16.2, [Common Procedures](#), for more information.

The following procedures can be performed from the **One Page** tab:

Procedures

- [Setting headers and footers in Simple page templates](#)
 - Creating a new page or frame; see [Creating a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Cloning a page or frame; see [Cloning a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Delete a page or frame; see [Deleting a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Name a page or frame; see [Naming a page, frame, text style, draw style, or table style](#), in Chapter 16
 - Change the color of a page or frame; see [Changing the color](#), in Chapter 16
 - Change the unit of measurement for a page or frame; see [Changing the unit of measurement](#), in Chapter 16
 - Change the size of a page or frame; see [Changing the size of a page or frame](#), in Chapter 16
 - Changing the location of a page or frame; see [Changing the location of the page or frame](#), in Chapter 16
 - [Changing a page's orientation](#)
 - [Changing a page's flow](#)
 - [Changing the orientation of a frame's contents](#)
 - [Changing the number of columns in a frame](#)
 - [Changing a frame's margins](#)
 - [Changing a frame's border](#)
 - [Changing a frame's flow](#)
-

17.1.1 Setting headers and footers in Simple page templates

To set a header or footer in a Simple page template, simply select the corresponding checkbox.

17.1.2 Changing a page's orientation

By default, JClass ServerReport Designer selects the page orientation automatically.

1. Click the *Orientation* drop-down list.
2. Select either **Automatic**, **Portrait** or **Landscape**.

17.1.3 Changing a page's flow

When you create pages in JClass ServerReport Designer, you must make sure to include information about how they relate to one another.

Note: You can also change a page's flow from the **All Pages** tab.

- Click the **Flow Page** or **Flow Section** drop-down menu and select the next page in the flow.

To view the page flow for the entire file, click the **All Pages** tab in the **Page Template** tab.

Note: Flow pages and frames are only available for Full page templates.

17.1.4 Changing the orientation of a frame's contents

The rotation of content flowed into a frame can be altered by setting its **Content Rotation**.

1. Select the frame for which you would like to change the content rotation.
2. From the drop-down list, select the rotation. Options are **0 Degrees**, **90 Degrees**, **180 Degrees**, and **270 Degrees**.

Note: This option is only available for Full page templates.

17.1.5 Changing the number of columns in a frame

You can choose to have columns in your frame. If you do not want columns, leave the column count at 1.

1. Highlight the text inside the *Count* text box and delete it.
2. Enter the number of columns you would like in your frame in the text box.

Note: Valid numbers must be positive integers.

3. To commit the change, click anywhere outside the text box. To cancel the change, hit the *escape* key before committing the change.
4. Highlight the text inside the *Spacing* text box and delete it.
5. Enter the amount of space you would like to leave between each column in the text box.
6. To commit the change, click anywhere outside the text box or hit the return key. To cancel the change, hit the *escape* key before committing the change.

17.1.6 Changing a frame's margins

1. Highlight the text inside either the *Top*, *Bottom*, *Left* or *Right* text box, depending on which margin you would like to change.
2. Enter the size of the margin in the text box.

Note: Input must be a valid positive number for your locale.

3. To commit the change, click anywhere outside the text box or hit the return key. To cancel the change, hit the *escape* key before committing the change.

17.1.7 Changing a frame's border

You can change a few different aspects of a frame border's appearance.

Note: By default, frames all have borders around them. If you would like to remove the border, set the thickness to zero.

Color

To learn how to change the color of a frame's border, please refer to [Changing the color](#), in Chapter 16.

Type

1. Click the *Type* drop-down list.
2. Select either **Blank**, **Plain**, **Dashed** or **Double**.

Thickness

1. Highlight the text in the *Thickness* text box and delete it.
2. Enter a value for the thickness of the border in the text box.

Note: Input must be a valid positive number for your locale.

3. To commit the change, click anywhere outside the text box or hit the return key. To cancel the change, hit the *escape* key before committing the change.

Setting Bounds


For Simple page templates, the offset of the frame is determined by setting bounds.


- To set the bounds, enter a value in both the **Top Offset** and **Bottom Offset** fields.



17.1.8 Changing a frame's flow

FlowFrames dictates how the text will flow from frame to frame on a page.

1. Make sure that all the frames you would like to direct the flow of text through have been added to the *FlowFrames* list. Please note that any frames in the *FlowFrames* list will be a part of the text flow.

To add a frame to the list, highlight it in the *Frames* list, and click the *FlowFrame's*  button.

To remove a frame from the list, highlight it in the *FlowFrames* list and click the  button.

2. To change the flow order, highlight the frame that needs to be moved in the *FlowFrames* list and click the  or  button to adjust its position in the list.

17.2 All Pages Tab

The **All Pages** tab shows you a visual representation of the way in which the pages and/or sections flow.

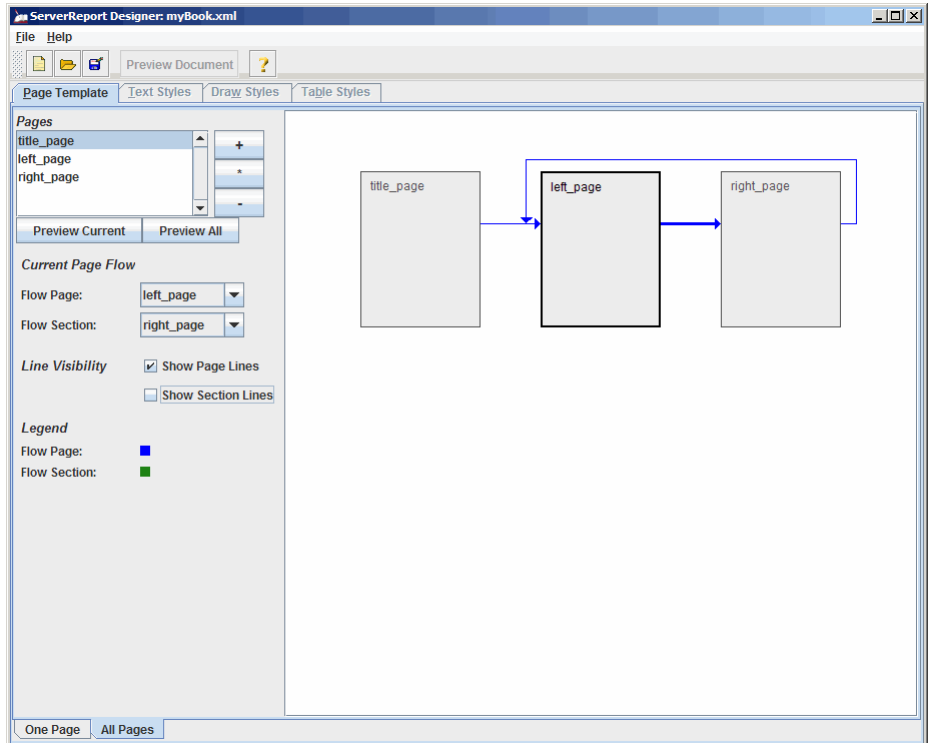


Figure 69 The All Pages tab in JClass ServerReport Designer.

Note: You can change the order in which the pages appear, but this does not change anything in your document or page template, except that the first page is used as the starting page.

Some procedures that involve pages can be performed on any screen. Please refer to [Common Procedures](#), in Chapter 16, for more information on creating, cloning, and deleting a page, as well as changing the page's flow.

The following procedures can be made from the **All Pages** tab:

Procedures

- Creating a new page; see [Creating a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Cloning a page; see [Cloning a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Delete a page; see [Deleting a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - [Changing a page's flow](#)
 - [Changing the visibility of the flow lines](#)
-

17.2.1 Changing a page's flow

When you create pages in JClass ServerReport Designer, you must make sure to include information about how they relate to one another. There are two methods that you can use to change both the Flow Page and the Flow Section in the All Pages window.

Note: You can also change a page's flow from the **One Page** tab.

- Click the **Flow Page** or **Flow Section** drop-down menu and select the next page in the flow.
- Click and drag the flow line to the next page in the flow.

17.2.2 Changing the visibility of the flow lines

By default, all the page and section flow lines are visible in the **All Pages** tab. When creating a large document, this can become cumbersome. For that reason, you can chose to make the lines invisible.

- To remove the page flow lines, click the **Show Page Lines** check box to deselect it.

Note: This option is only available for Full page templates.

- To remove the section flow lines, deselect the **Show Section Lines** check box.

Note: Below the *Line Visibility* section there is a legend which will indicate the color of each of the lines.

18

Text Styles

Designing Text Styles

JClass ServerReport Designer allows you to design your own text styles to customize the text you will flow into your document. Text Styles can be saved as part of a document, or alone as a Text Styles template. A Text Styles template can be loaded into JClass ServerReport easily, but contains only text-style tags.

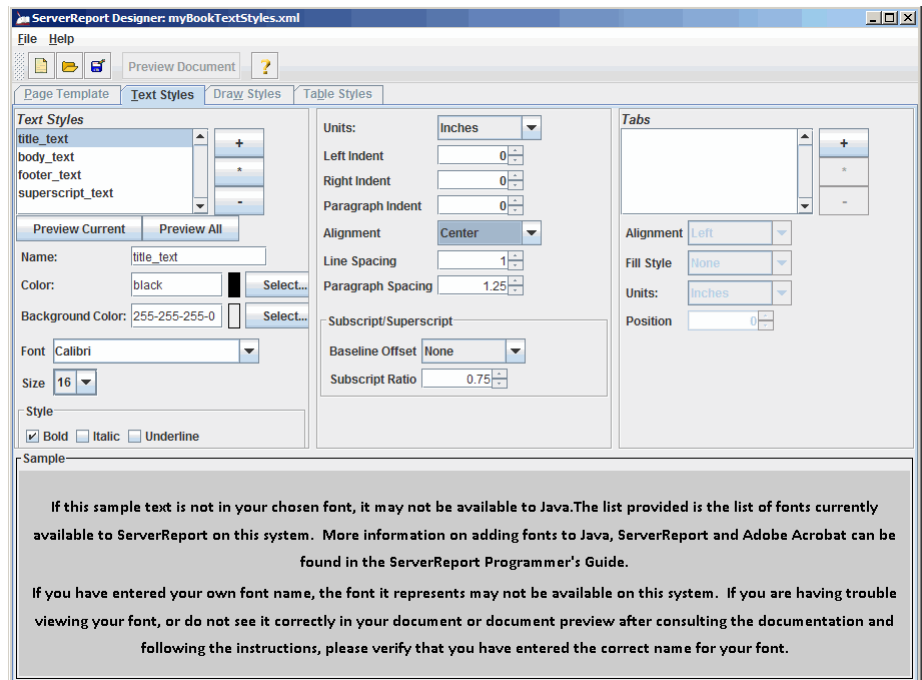


Figure 70 The Text Styles tab in JClass ServerReport Designer.

18.1 The Sample Area

The *Sample* area appears below the JClass ServerReport Designer's controls, and will update dynamically as you change your text style settings. This provides you with a visual representation of the text style without having to preview it as a PDF. The Sample area is not editable; it is intended to allow you to see the outcome of your changes, not to provide an area in which to make those changes.

The following list outlines basic procedures that can be accomplished in the **Text Styles** tab:

Procedures

- Creating a new text style; see [Creating a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Cloning a text style; see [Cloning a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Delete a text style; see [Deleting a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Name a text style; see [Naming a page, frame, text style, draw style, or table style](#), in Chapter 16
 - Change the color of a font; see [Changing the color](#), in Chapter 16
 - [Changing line and paragraph spacing](#)
 - [Changing paragraph indentation](#)
 - [Changing font properties](#)
 - [Setting tabs](#)
-

18.2 Designing Text Styles

18.2.1 Changing line and paragraph spacing

Line spacing (or leading) refers to the amount of space between different lines of text in the same paragraph; paragraph spacing refers to the amount of space between the first line of a paragraph and the last line of the paragraph preceding it.

Paragraph and line spacing is proportional to the height of a regular line of text. A spacing of one will be equal to one line. Both values must be between zero and five.

- To change line or paragraph spacing, you can either type a value in the respective text field, or use the up and down arrows to adjust the value by 0.05.

18.2.2 Changing paragraph indentation

The left indent determines how much space is between the left side of the frame and where the text begins; the right indent determines the minimum amount of space between where the text ends and the right side of the frame. A paragraph indent determines the indentation for the first line of every paragraph.

Note: The right indent must not cross either the left indent or the paragraph indent. When this happens, an error message will appear in the Sample window.

Procedures

- Changing the unit of measurement used for setting indents; see [Changing the unit of measurement](#), in Chapter 16
 - [Setting indentation](#)
-

Setting indentation

- To change the left indent, you can either type a value into the *Left Indent* text field, or use the up and down arrows to adjust the value.
- To change the right indent, you can either type a value into the *Right Indent* text field, or use the up and down arrows to adjust the value.
- To change the paragraph indent, you can either type a value into the *Paragraph Indent* text field, or use the up and down arrows to adjust the value.

18.2.3 Changing font properties

Font properties determine what the font for the selected text style will look like. JClass ServerReport Designer allows you to alter many font properties, including size, style, and alignment.

Procedures

- [Changing the font family, size, and style](#)
 - [Changing font alignment](#)
 - [Changing baseline offset and subscript ratio](#)
-

Note: The list provided is the list of fonts currently available to ServerReport on this system; if the sample text is not in your chosen font, it may not be available to Java. For more information on adding fonts to Java, to JClass ServerReport, and to Adobe Acrobat, see [Defining Text-Based Content](#), in Chapter 2. If you have entered your own font name, the font it represents may not be available on this system. If you are having trouble viewing the font in your sample, or do not see it correctly in your document or when previewing your document, and you have

consulted the JClass ServerReport documentation, please verify that you have entered the correct name for your font

Changing the font family, size, and style

The **Font** and **Size** properties are not editable fields; this means that you must select from the values provided in the drop down list. All of the fonts that are available on your system will automatically appear in the *Font* drop down list; font sizes range from 8 to 72.

- To change the font family, click the *Font* drop down list, and select the font to use for the text style.
- To change the font's size, use the *Size* drop down list to select the size of the font.
- To apply bold, italic, or underline font styles to your font, select the **Bold**, **Italic**, or **Underline** checkbox, respectively. If one or more of these styles, or style combinations, are not available for your font, the checkboxes will be disabled.

Changing font alignment

The font's alignment determines how a line of text will be arranged in the frame.

- To change the alignment for your text style, click the *Alignment* drop down list and select one of **Left**, **Right**, **Center**, **Justify**, or **None**.

Note: **Justify** means that all text lines (excluding the last line) will be stretched or condensed to align against both the right and left margins, eliminating any ragged edges.

Changing baseline offset and subscript ratio

Subscript and superscript text is determined by the baseline offset and subscript ratio properties. The baseline offset determines if text will be above (superscript), below (subscript), or on the baseline; the subscript ratio will change the size of the font, adjusting it proportionally to the font size. A subscript ratio of one will be equal to the font size; the ratio can range from 0 to 1.5.

Note: The subscript ratio only affects text whose baseline offset is either **Subscript** or **Superscript**; this ensures that it is only applied to subscript or superscript text.

1. To create subscript or superscript text, first change the baseline offset in the *Baseline Offset* drop down list. Select either **Subscript** or **Superscript**.
2. To adjust the font size, type in a value from 0 to 1.5 in the *Subscript Ratio* text box, or use the up and down arrows to adjust the value by 0.05.

18.2.4 Setting tabs

There are four different types of tab alignment that can be designed in the JClass ServerReport Designer: left, right, center, and decimal. The left tab alignment aligns the content with a straight left edge (to the right of the tab stop); the right tab alignment aligns the content with a straight right edge (to the left of the tab stop). Center tab alignment centers content, where the tab stop is the center. Tab stop using decimal alignment aligns numbers by their decimal points.

Note: Decimal alignment is intended to be used with numerical data. If the data entered is not numerical, it will be right aligned.

2.157	2.157	2.157	2.157
15.25	15.25	15.25	15.25
2.1	2.1	2.1	2.1
left tab stop	right tab stop	center tab stop	decimal tab stop

Figure 71 Example of each tab alignment.

Tabs are automatically provided with a name, that reflects their unit of measurement and position. If more than one tab exists using the same unit of measurement and at the same position, a hash symbol (#) will be appended to the name to distinguish them.

Note: The tab names used in the JClass ServerReport Designer are for informational purposes only; tabs do not have a name property in JClass ServerReport XML.

Procedures

- Creating a new tab; see [Creating a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Cloning a tab; see [Cloning a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Deleting a tab; see [Deleting a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
 - Changing the unit of measurement used for tabs; see [Changing the unit of measurement](#), in Chapter 16
 - [Setting tab values](#)
-

Setting tab values

To set up a tab, it must have a specified position and alignment. These do not need to be unique. Tab fill styles can also be applied to tabs; they will fill in any blank space left from a tab with either dots or an underline.

- To set the tab position, type in a value in the *Position* text box, or use the up and down arrows to adjust the value.
- To set the alignment, select either **Left**, **Right**, **Center**, or **Decimal** from the *Alignment* drop down list.
- To set the fill style, select either **None**, **Dots**, or **Underline** from the *Fill Style* drop down list.

19

Draw Styles

Designing Draw Styles

JClass ServerReport uses draw styles for various purposes, such as drawing shapes or for table borders. The Draw Styles tab allows you to customize your own draw styles.

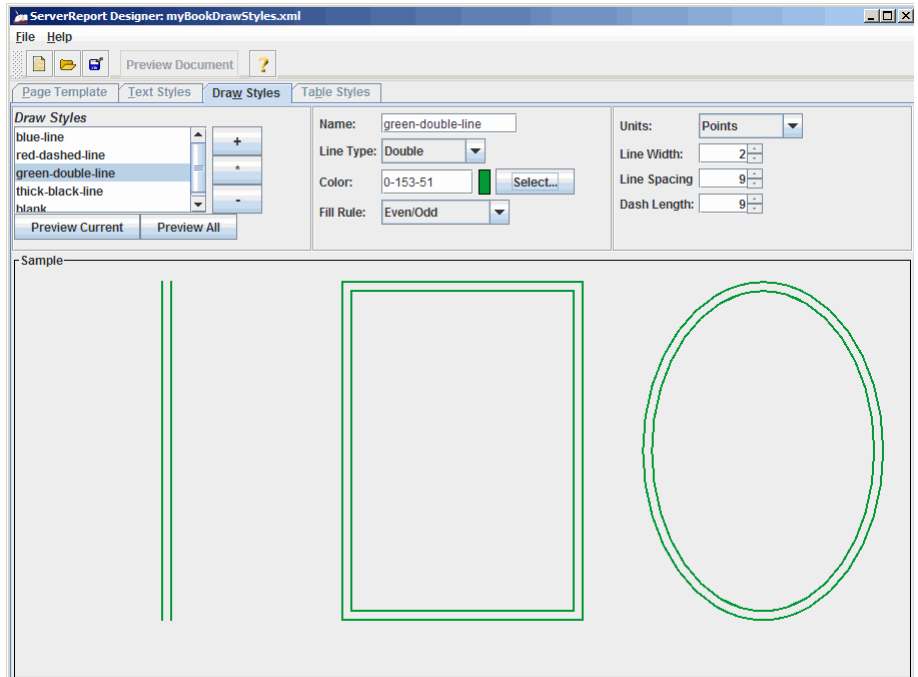


Figure 72 The Draw Styles tab in JClass ServerReport Designer.

19.1 The Sample Area

The *Sample* area appears below the JClass ServerReport Designer's controls, and will update dynamically as you change your draw style settings. This provides you with a visual representation of the draw style without having to preview it as a PDF. The Sample area is not editable; it is intended to allow you to see the outcome of your changes, not to provide an area in which to make those changes.

The following table lists basic procedures that can be done in the **Draw Styles** tab:

Procedures
<ul style="list-style-type: none">■ Creating a new draw style; see Creating a page, frame, text style, tab, draw style, or table style, in Chapter 16■ Cloning a draw style; see Cloning a page, frame, text style, tab, draw style, or table style, in Chapter 16■ Deleting a draw style; see Deleting a page, frame, text style, tab, draw style, or table style, in Chapter 16■ Naming a draw style; see Naming a page, frame, text style, draw style, or table style, in Chapter 16■ Changing the color of a draw style; see Changing the color, in Chapter 16■ Changing the unit of measurement for a draw style; see Changing the unit of measurement, in Chapter 16■ Defining the line type■ Defining the fill rule■ Defining the line width■ Changing line spacing■ Changing the dash length

19.2 Designing Draw Styles

19.2.1 Defining the line type

The line type determines what kind of line to draw for that particular draw style.

- To change the line type, select either **Plain**, **Dashed**, **Double**, or **Blank** from the drop-down list.

19.2.2 Defining the fill rule

The fill rule determines which areas of a shape to fill; JClass ServerReport Designer allows you to choose between even/odd and non-zero winding.

Note: Defining the fill rule does not update the *Sample* area.

- To change the fill rule, select either **Even/Odd** or **Non-zero Winding** from the drop-down list.

19.2.3 Defining the line width

Line width controls the thickness of the draw style line.

- To change line width, you can either type a value in the respective text field, or use the up and down arrows to adjust the value.

19.2.4 Changing line spacing

If you design a multi-line draw style, the line spacing determines the amount of space between the different lines. Therefore, changing the line spacing only results in a change in the line style if the line type is **Double**.

- To change line spacing, you can either type a value in the respective text field, or use the up and down arrows to adjust the value.

19.2.5 Changing the dash length

The dash length controls the length of the dashes and the spaces between them. This option only results in a change in the line style if the line type is **Dashed**.

- To change the dash length, you can either type a value in the respective text field, or use the up and down arrows to adjust the value.

Table Styles

Designing Table Styles

The **Table Styles** tab allows you to customize the appearance of tables in the JClass ServerReport Designer.

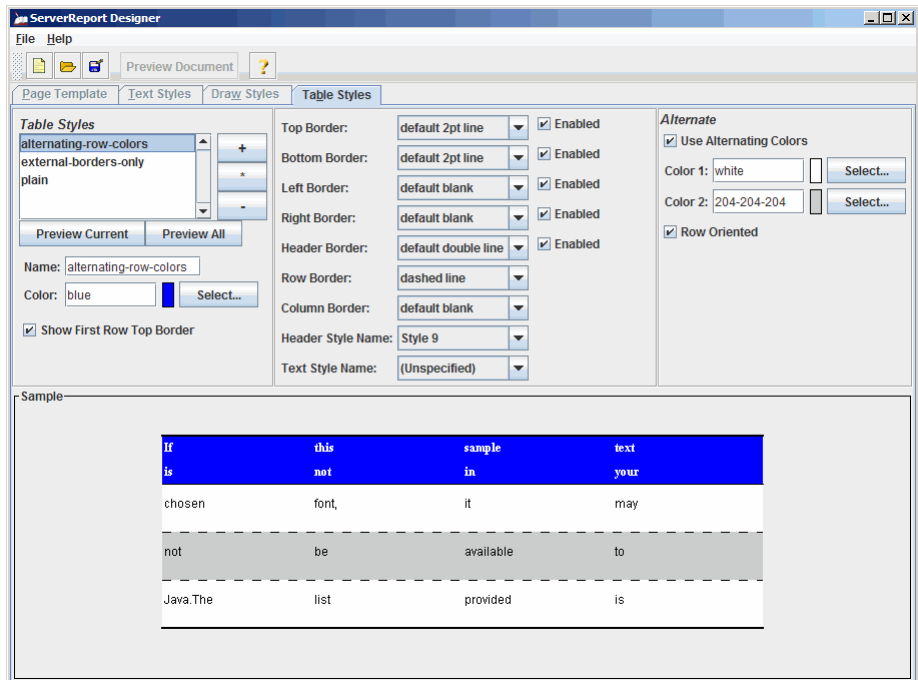


Figure 73 The Table Styles tab in JClass ServerReport Designer.

20.1 The Sample Area

The *Sample* area appears below the JClass ServerReport Designer's controls, and will update dynamically as you change your table style settings. This provides you with a visual representation of the table style without having to preview it as a PDF.

The *Sample* area is not editable; it is intended to allow you to see the outcome of your changes, not to provide an area in which to make those changes.

The following list outlines basic procedures that can be accomplished in the **Table Styles** tab:

Procedures

- Creating a new table style; see [Creating a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
- Cloning a table style; see [Cloning a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
- Deleting a table style; see [Deleting a page, frame, text style, tab, draw style, or table style](#), in Chapter 16
- Naming a table style; see [Naming a page, frame, text style, draw style, or table style](#), in Chapter 16
- Changing the color of a table style; see [Changing the color](#), in Chapter 16.

Note: The background color will only be apparent if your table does not use alternate colors for rows or columns.

- [Showing the first row's top border.](#)
 - [Defining borders.](#)
 - [Selecting the header and text style names.](#)
 - [Alternating colors for rows or columns.](#)
-

20.2 Designing Table Styles

20.2.1 Showing the first row's top border

Show First Row Top Border allows the top border for the first row following the header row in the table to display its border. For example, if the row borders are set to **default 1pt line** and the header border is set to **default blank**, selecting **Show First Row Top Border** allows the 1pt line at the top of the first row to show; otherwise, the header border style will override the first row's top border setting and only the header border will be shown (in this case, no border).

- To enable **Show First Row Top Border**, select the corresponding checkbox.

20.2.2 Defining borders

JClass ServerReport Designer uses draw styles as table border styles. While there are several borders to define when creating a table style, for all borders the options are: **(Unspecified)**, **dashed line**, **default 1pt line**, **default 2pt line**, **default blank**, **default double line**, **default line**, and any draw styles that you have previously defined for the document. For more information on defining draw styles using the JClass ServerReport Designer, see Chapter 19, [Draw Styles](#). For more information on the stock draw styles, see [Stock Draw Styles](#), in Chapter 3.

- To change the border, select the draw style from the drop-down list that corresponds to the border you would like to change. The borders that you can change are:
 - *Top Border*, which defines the top border of the entire table
 - *Bottom Border*, which defines the bottom border of the entire table
 - *Left Border*, which defines the left border of the entire table
 - *Right Border*, which defines the right border of the entire table
 - *Header Border*, which defines the bottom border of the Header table
 - *Row Border*, which defines the borders between different rows in the table
 - *Column Border*, which defines the borders between different columns in the table

Note: *Top Border*, *Bottom Border*, *Left Border*, *Right Border*, and *Header Border* all have a corresponding **enabled** checkbox. When this checkbox is selected, the border is visible and editable.

20.2.3 Selecting the header and text style names

The header and text style names will determine which style is used for the header table and text in the entire table, respectively. For header style names, options are: **default**, **Style 0** through **Style 17**, and any table styles you have previously defined for the document. For text style names, options are: **(Unspecified)**, **Bold**, **Bold Italic**, **Code**, **CodeIndented**, **H1** through **H7**, **Heading**, **HeadingBold**, **Indented**, **Italic**, **Normal**, **Plain**, **WhiteBold**, **default header**, **default text**, and any text styles that you have previously defined for the document.

For more information on the different stock text styles, see [Predefined JCTextStyle Objects](#), in Chapter 2. For more information on creating your own text styles using JClass ServerReport, see Chapter 2, [Defining Text-Based Content](#).

For more information on the different stock table styles, see [Table Styles](#), in Chapter 5.

Style (name property)	Description	Example																				
default	<ul style="list-style-type: none">■ thin left, right, top, bottom, horizontal, header, and column borders■ no column shading■ regular heading font	<div>-- default --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 0	<ul style="list-style-type: none">■ no left, right, top, bottom, header, horizontal, or column borders■ no column shading■ regular heading font	<div>-- Style 0 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 1	<ul style="list-style-type: none">■ thick header border; no other borders■ no column shading■ regular heading font	<div>-- Style 1 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 2	<ul style="list-style-type: none">■ thin header border; thick top and bottom borders; no other borders■ no column shading■ regular heading font	<div>-- Style 2 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 3	<ul style="list-style-type: none">■ thick top, bottom, and header borders; thin horizontal borders; no column, left, or right borders■ no column shading■ regular heading font	<div>-- Style 3 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 4	<ul style="list-style-type: none">■ no left, right, top, bottom, horizontal, or column borders; thick header border■ header colored (black); reverse type for header text■ no column shading	<div>-- Style 4 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

Style (name property)	Description	Example																				
Style 5	<ul style="list-style-type: none">■ thick right, left, bottom, and header borders; no top, column, or horizontal borders■ header colored (black); reverse type for header text■ no column shading	<div>-- Style 5 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 6	<ul style="list-style-type: none">■ thick right, left, bottom and header borders; thin horizontal and column borders; no top border■ header colored (black); reverse type for header text■ no column shading	<div>-- Style 6 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 7	<ul style="list-style-type: none">■ thick right, left, top, bottom and header borders; no horizontal or column borders■ header colored (gray); reverse type for header text■ no column shading	<div>-- Style 7 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 8	<ul style="list-style-type: none">■ thick right, left, top, bottom, and header borders; thin horizontal and column borders■ header colored (gray); reverse type for header text■ no column shading	<div>-- Style 8 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

Style (name property)	Description	Example																				
Style 9	<ul style="list-style-type: none">■ thin header border; thick top, and bottom borders; no right, left, horizontal, and column borders■ header colored (gray); reverse type for header text■ no column shading	<div>-- Style 9 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 10	<ul style="list-style-type: none">■ thick right, left, top, and bottom borders; thin header, horizontal, and column borders■ regular heading font■ no column shading	<div>-- Style 10 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 11	<ul style="list-style-type: none">■ no right, left, top, and bottom borders; thin header, horizontal, and column borders■ plain headers■ no column shading	<div>-- Style 11 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 12	<ul style="list-style-type: none">■ thick right, left, top, bottom, and header borders; no horizontal or column borders■ header colored (gray); reverse type for header text■ gray column shading	<div>-- Style 12 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 13	<ul style="list-style-type: none">■ thick right, left, bottom, and header borders; thin horizontal border; no top or column borders■ header colored (black); reverse type for header text■ no column shading	<div>-- Style 13 --</div> <table><tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th><th>Column 5</th></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

Style (name property)	Description	Example																				
Style 14	<ul style="list-style-type: none">■ thin right, left, top, bottom, and horizontal borders; thick header border; no column border■ plain header■ no column shading	<p>-- Style 14 --</p> <table><tr><td>Column 1</td><td>Column 2</td><td>Column 3</td><td>Column 4</td><td>Column 5</td></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 15	<ul style="list-style-type: none">■ thin right, left, top, bottom, header, and column borders; no horizontal border■ plain header■ no column shading	<p>-- Style 15 --</p> <table><tr><td>Column 1</td><td>Column 2</td><td>Column 3</td><td>Column 4</td><td>Column 5</td></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 16	<ul style="list-style-type: none">■ thin top, bottom, left, right, and horizontal borders; thick header border; thin border between last two columns■ header colored (dark gray); reverse type for header text■ no column shading■ alternate row shading (light gray and dark gray)	<p>-- Style 16 --</p> <table><tr><td>Column 1</td><td>Column 2</td><td>Column 3</td><td>Column 4</td><td>Column 5</td></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		
Style 17	<ul style="list-style-type: none">■ thin top, bottom, right, left, and horizontal borders; thick header border; thin border between last two columns■ header reverse type■ alternate column shading (light gray and dark gray)	<p>-- Style 17 --</p> <table><tr><td>Column 1</td><td>Column 2</td><td>Column 3</td><td>Column 4</td><td>Column 5</td></tr><tr><td>Cell (0,0)</td><td>Cell (0,1)</td><td>Cell (0,2)</td><td>Cell (0,3)</td><td>Cell (0,4)</td></tr><tr><td>Cell (1,0)</td><td>Cell (1,1)</td><td>Cell (1,2)</td><td>Cell (1,3)</td><td>Cell (1,4)</td></tr><tr><td>Cell (2,0)</td><td>Cell (2,1)</td><td>Cell (2,2)</td><td>Cell (2,3)</td><td>Cell (2,4)</td></tr></table>	Column 1	Column 2	Column 3	Column 4	Column 5	Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)	Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)	Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)
Column 1	Column 2	Column 3	Column 4	Column 5																		
Cell (0,0)	Cell (0,1)	Cell (0,2)	Cell (0,3)	Cell (0,4)																		
Cell (1,0)	Cell (1,1)	Cell (1,2)	Cell (1,3)	Cell (1,4)																		
Cell (2,0)	Cell (2,1)	Cell (2,2)	Cell (2,3)	Cell (2,4)																		

- To change the header or text style name, select the desired style from the corresponding drop-down list.

Note: If the background color of the header style is `null`, the default background color for the header table will be the same as the background color for the body style. Should the body style background color also be set to `null`, the default background color is white. Alternating row or column colors will have no effect.

20.2.4 Alternating colors for rows or columns

JClass ServerReport Designer allows you to design tables that alternate colors based on either rows or columns.

Note: If your tables uses alternate colors for rows or columns, you will no longer be able to see the background color.

1. To create a table with alternating rows or columns, select the **Use Alternate Colors** checkbox.
2. In the *Color 1* field, select the color for the first row or column, depending on how the colors will alternate. In the *Color 2* field, select the alternating color.

For more information about changing colors, see [Changing the color](#), in Chapter 16.

3. By default, tables alternate colors based on columns. If you would like to alternate them by row, select the **Row Oriented** checkbox.

Part **IV**

*Reference
Appendices*

Render Objects

Render Object Categories ■ Subclasses of the Render object

Render objects allow a description of the page-marking actions needed to draw a page stored in memory as the document is created. There are two kinds of render objects: those directly representing page-marking primitives (such as drawn lines or formatted text), and those representing additions to the flow of conceptual objects (such as horizontal rules). The current set of provided render objects includes all supported graphical primitives and all higher-level objects (for instance, images and tables) which can be added to the flow.

A.1 Render Object Categories

There are several interfaces that provide categorization of the types of render objects. The basic interfaces comprise:

- **Embedable.** Embedable objects can be added into the text flow as flowed objects because the interface provides the necessary support for determining the position of this object relative to the line contents. Text is not considered to be embedded because it defines the baseline of a line and doesn't need alignment attributes. `ImageRender` and `FrameRender` are the only Embedable render objects.
- **Floatable.** Floatable objects can be added into the document flow as independent paragraph-level objects that do not break lines and allow normal flowed content to be added while they are awaiting sufficient space. `ImageRender` and `FrameRender` are the only Floatable objects.
- **FlowMarker.** FlowMarker objects are not graphical primitives, but represent logical elements of the flow, which may have printer-specific or variable handling. The current FlowMarker objects are `HRuleMarker`, `ImageMarker`, and `TableMarker`.
- **Splitable.** Splitable objects are flow objects that implement methods that allow them to be broken into smaller pieces in order to fit on lines. The best example is text, which can be divided into words or even individual letters. `StringRender` is currently the only Splitable object.

A.2 Subclasses of the Render object

The `com.klg.jclass.sreport.render` package also contains classes that are not render objects; there are a number of page numbering and counting macros. The macro classes each implement a single page numbering or counting mechanism, such as representing the page number in roman numerals, and can be distinguished by names ending in `...Macro.java`. The following table describes all objects that can be considered render objects.

Object	Description
<code>ArcRender</code>	Represents a graphical primitive in the shape of a circular arc. A complete circle is a special case of this render object.
<code>BoxRender</code>	A graphical primitive describing a rectangular area.
<code>FrameRender</code>	Represents a <code>JCFrame</code> object that can be placed within another frame.
<code>HRuleMarker</code>	Indicates that at this point in the flow, a line is drawn between the margins of the current frame. Storing a horizontal rule in this fashion allows the printer to make a decision about how it is reproduced rather than simply copying a stored line.
<code>ImageMarker</code>	Records the point in the flow at which a floating image was added. This marker allows the HTML printer to reposition images at their point of addition to the document, rather than at the point at which they may finally have been placed on a page.
<code>ImageRender</code>	Represents a drawable object such as an <code>Image</code> or a <code>Component</code> .
<code>LineRender</code>	A graphical primitive for a line with an indefinite number of straight segments. Curves are not supported.
<code>MacroRender</code>	This render object stores a reference to a text macro that is to be evaluated and the result entered at the given point in the document.
<code>RoundRectRender</code>	Describes a rectangular graphical primitive with rounded corners.
<code>StringRender</code>	Represents all text elements in the document. As such it exports a number of methods designed to allow manipulation of the contents of text <code>Strings</code> .
<code>SymbolRender</code>	Instances of this class represent actions with respect to the text flow, such as <code>newLine</code> and <code>newParagraph</code> . A <code>SymbolRender</code> marking a new line action distinguishes an application-driven <code>newLine()</code> – the parent application called <code>newLine()</code> on the <code>JCFlow</code> or <code>JCFrame</code> – from an implicit <code>newLine()</code> , which occurs when a line becomes filled by the <code>flowPrint()</code> mechanism.

Object	Description
TableCellRender	This object locates and encapsulates a <code>CellRenderer</code> object so that the contents of a table cell can be drawn at the correct location.
TableMarker	Records the insertion in the document of a table, and subsequently the actual point of placement of a range of cells of a <code>JCPageTable</code> .

RTF Limitations and Unsupported Features

Unsupported Features ■ *Limitations*

RTF is a limited format which does not support all of the available JClass ServerReport functionality. The following lists all of the features that are not available, as well as limitations of the format.

B.1 Unsupported Features

For information on how to determine JClass ServerReport's behavior if an unsupported feature is encountered, please refer to [Defining RTF Behavior for Unsupported Features](#), in Chapter 9.

- Encryption and security features are PDF specific features.
- Paragraph styles must apply to the whole paragraph.
- Paragraph spacing must be positive.
- Custom user macros cannot be converted; however, they are evaluated and the result is placed in the RTF file. If the macro depends on information that is not known to JClass ServerReport during RTF output (for example, page position or page number), the result may be incorrect.
- Hyperlinks cannot surround any macros.
- Fonts cannot be embedded.
- Embedded components are converted into raster images before they can be embedded.
- Vertically aligning embedded components has no effect.
- Floating images, frames, and components are not supported.
- Embedded frames are not supported.
- Front matter and contents lists are not supported.
- Pasting images and components is not supported
- Frame draw commands are not supported.
- Overflow is only rudimentary, and generally is not supported.

- The `markExactLocation()` method is not supported.
- Bookmark trees are not supported, and thus cannot be created.
- Exact row heights for tables cannot be specified.
- Watermarks are not supported.
- The rotation of the contents of a frame is not supported.
- Flush policy has little effect on the output, as content is automatically flushed at the end of each paragraph.
- Output policy has little effect on the output; the page-by-page value is negated because RTF requires that most of the document is built before it can be output.
- Events behave differently, as the layout of text across pages is left to the viewer application (Microsoft Word) and only allows one `pageBegin()/pageEnd()/frameComplete()` sequence (unless `newPage()` is explicitly called), and one `frameBegin()/frameEnd()/frameComplete()` sequence for the flow frame, the header, and the footer.
- Tabs that pass the end of a line behave differently.

B.2 Limitations

Depending on the version of Microsoft Word, the limitations of the RTF functionality may change.

Supported Functionality	Microsoft Word 97	Microsoft Word 2000	Microsoft Word 2002
Horizontal Rules ^a	No	Yes	Yes
Table Cell Margins	No	Yes	Yes
Nested Tables	No	Yes	Yes
Colors	limited to built-in palette	Full RGB	Full RGB

a. Horizontal rules utilize RTF commands that are supported by Microsoft Word 2000 and above. In Microsoft 97, these commands can result in lines being drawn in the wrong position. To output RTF that is compatible with Word 97 and prevent horizontal rules from drawing, set the `CompatibilityLevel` property of the `RTFOutputProperties` class to `COMPATIBILITY_WORD_97`. An instance of the `RTFOutputProperties` class may be retrieved by calling `JCDocument.getOutputProperties()`.

XML Background Information

XML Primer ■ *DTD Primer*

XML can be used in conjunction with JClass ServerReport in many ways including:

- transforming your XML data into PDF
- separating page and typeface design and content from programming
- quickly and easily devising custom document “look-and-feel” elements

Primers on XML and document type definition (DTD) files are provided. These give you the background to work with JClass ServerReport to convert your XML to PDF. Prior to using JClass ServerReport for XML, you should be comfortable writing Java code, creating and revising XML, and creating and running servlets.

Note: The [JClass ServerReport Designer](#) creates XML output intended to be used with JClass ServerReport.

Demos and Examples

JClass ServerReport comes with many examples and demos to highlight features of JClass ServerReport, as well as to help you understand the ways in which you can configure JClass ServerReport in your own applications, servlets, and Java Server Pages (JSPs).

JClass ServerReport demos are installed into *JCLASS_SERVER_HOME/demos/sreport/* and examples are installed into *JCLASS_SERVER_HOME/examples/sreport/*. Servlet demos and examples are also included in the JClass ServerReport WAR file (*sreport-samples.war*), installed into *JCLASS_SERVER_HOME/examples/war/*.

C.1 XML Primer

XML – eXtensible Markup Language – is a scaled-down version of SGML (Standard Generalized Markup Language), the standard for creating a document structure. XML was designed especially for Web documents, and allows designers to create customized tags (“extensible”), thereby enabling common information formats for sharing both the format and the data on the Internet, intranets, et cetera.

XML is similar to HTML in that both contain markup tags to describe the contents of a page or file. But HTML describes the content of a Web page (mainly text and graphic images) only in terms of how it is to be displayed and interacted with. XML, however, describes the content in terms of what data is being described. This means that an XML file can be used in various ways. For instance, an XML file can be utilized as a convenient way to exchange data across heterogeneous systems. As another example, an XML file can be processed (for example, via XSLT [Extensible Style Sheet Language Transformations]) in order to be visually displayed to the user by transforming it into HTML.

Please note that in XML, certain special characters need to be “escaped” if you want them to be displayed. For example, you cannot simply put an ampersand (&), a less than sign (<), or a greater than sign (>) into a block of text; these special characters are represented as `&`, `<`, and `>`, respectively. For details on this topic, please see http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/sax/4_refs.html#chars.

Further Information About XML

Here are links to Web sites that contain information on XML.

<http://www.w3.org/XML/> – another W3C site; contains information on standards

<http://www.ucc.ie/xml> – an extensive FAQ devoted to XML

<http://java.sun.com/docs/index.html> – Sun’s XML site

C.2 DTD Primer

The document type definition (DTD) file has one purpose: to specify the structure of an XML file. The DTD describes, in XML Declaration Syntax, the particular type of document, and sets out what names are to be used for various elements, where these elements may occur, and how they work together.

Glossary

A

Abstract Class

Any class that cannot be instantiated because it contains at least one abstract method or is declared as abstract, is an Abstract Class. It is possible to extend an abstract class, or make it concrete, by implementing the abstract method.

See [Abstract Method](#).

Abstract Method

An Abstract Method is simply a method which does not have a body, and therefore cannot have an implementation. Abstract Methods are only signature definitions.

Abstract Windowing Toolkit (AWT)

The Abstract Windowing Toolkit is a series of graphical user interface components. These components are implemented using their native-platform versions, and provide a common subset of functionality.

Adobe Font Metrics (AFM)

AFM refers to files supplied by Adobe which contain font metrics for all Adobe-supported fonts. The file extension is .afm.

Applet

An Applet is a Java program invoked from an HTML page and run in a Java-enabled browser or applet viewer.

Argument

The data item specified in a method call is an Argument. Also referred to as a parameter.

See [Method](#).

Array

An Array is a collection of objects of the same type. Each object has its own position, that is specified by an integer.

Attribute

An Attribute is directly associated with the entity of the instance or instances for which it exists, and is simply a named value or relationship to the entity.

AWT Name**Abstract Windowing Toolkit Name**

The AWT Name is the font name recognized by your Java environment, and thus by JClass ServerReport. When programming a font change, make sure to always use the AWT Name.

B**Binary Operator**

A Binary Operator is an operator that takes two parameters, which can be expressions, numbers, or one of each.

See [Operators \(Binary and Unary\)](#).

C**Class**

A Class is a collection of data and its methods.

Object implementations are defined in Classes, as well as the interface it implements and its superclass (which is Object by default). A Class also includes instance and class variables and methods.

Classes are arranged hierarchically so as to allow classes to inherit from their superclasses.

Classpath

A Classpath points to files, archives, and directories so that the JVM and other Java programs can find them. It is an environment variable that must be set for JClass products to function properly.

Client

The Client is a computer that sends a request to the server. This is the machine that the server responds to by sending a reply message.

Clone

A Clone is an exact duplicate of the original.

Component

A Component is simply a software unit (for example, an applet). It exists at the application-level and is supported by a container. Components are configurable at deployment time.

Constructor

Constructors are instance methods that have the same name as their class, and are used to do any initialization required for new objects.

Container

A Container surrounds a component to provide it with security, deployment, runtime services, and component-specific services.

CSS**Cascading Style Sheets**

CSS allow styles to be defined, and later used, in an HTML file. These styles can be stored in-line in the HTML file or in a separate CSS file.

D**Data Loading**

Data Loading is the process by which JClass ServerReport receives the data from the database to use in the chart.

Data Model

A Data Model is the result of physically organizing data in a logical fashion. It is used as a template or interface through which a data source is constructed. In the case of a database, a Data Model is useful because it contains information pertaining to the contents of a database, including how the database information is used and how items in the database relate to one another.

Data Source

Data Source refers to the database from which JClass ServerReport has gathered the data used to draw the chart. For example, JClass DataSource.

DBMS

DataBase Management System

DBMS is the software (or set of software) that controls the major functions of a database (for example, organization, storage, retrieval of data, and security).

Derived Class

A class which is used to extend another class is a Derived Class.

For example, if class X extends class Y, class X can be said to be derived from class Y.

Document Type Definition (DTD)

In JClass ServerReport, the DTD is a file, included with the installation, that defines the tags and attributes used to specify that appearance of a chart when using XML.

DTD tags are built into the JClass ServerReport API.

The *ServerReportPageTemplate.dtd* is in the JClass ServerReport JAR file.

dpi

dpi is a resolution measurement, provided in dots per inch.

Dynamic Web Content

Dynamic Web Content refers to content on a web page that, with the help of a servlet, can be modified as it is being viewed on a user's screen. This content can be modified as a result of a user activity, as a part of a timing mechanism, or otherwise.

E

Embedded Object

An Embedded Object is a non-text object added to the flow as an element of a line.

These count as part of the line for the purpose of computing line height, though their effect on the overall line height will depend on their vertical alignment within the line.

Embedding

Embedding is the action that places an image on the current line of text. Text will wrap around an embedded object if there is not enough space on the current line to hold the image.

Event

An Event is a significant occurrence in a program.

Exception

When running a program, an Exception is something that will stop the program's normal execution. Generally, an error will be produced.

Expression

Expressions are elements in the program code that consist of constants, functions, operators, parentheses, and variables. Upon the completion of its execution, an Expression will return a value.

Expression List

An Expression List is a container that lists a group of values. This will permit you to perform an operation on that group of values.

Extensible Markup Language (XML)

Extensible Markup Language is a standard information document exchange format. XML is a simplified version of SGML, which creates very structured documents that is intended for use with Web documents. XML allows for the creation of customized tags; its structure is defined in a DTD file.

See [Document Type Definition \(DTD\)](#).

Extensible Style Sheet Language (XSL)

Extensible Style Sheet Language is a W3C standard for defining stylesheets for XML. XSL uses the XML language.

Extensible Style Sheet Language Transformation (XSLT)

Extensible Style Sheet Language Transformation is the language used to alter and manipulate documents, transforming XML documents.

F

File Extension

A File Extension is the part of the file name which indicates the data type stored in the file. It is located after the final point in the file name.

Floating Object

A Floating Object is an object added to the flow which, instead of being added as part of a line of text, remains above the rest of the elements added to the flow.

Flow

Flow refers to the conceptual stream of content added to a document which is automatically continued from frame to frame across multiple pages. Anything added to the document through a `JCFLOW` method is considered to belong to the flow.

Flow Mechanism

The Flow Mechanism is responsible for allocating content to pages and creating new pages as necessary.

Font Metrics

Font Metrics are measurements of the size and shape of the glyphs of a font which are used to compute the space required by text and to position letters in a printed text stream. Some information given by font metrics objects are font-wide, other information is applicable to individual glyphs.

See [Glyph](#).

Frame

The Frame is the basic unit of the flow. Frames contain all graphics (text, shapes, and images) rendered to the document.

G**GIF****Graphic Interchange Format**

GIF is a loss-less compression image format. GIF files use 256 colors, and are therefore most commonly used for drawings and icons.

Glyph

A Glyph is the actual drawn symbol which represents a particular character in a particular character set. The same character may have more than one glyph, depending on character set, context, ligatures, and so on.

Graphic User Interface (GUI)

A GUI is the combination of all graphical elements and techniques (including mouse and keyboard) that can be used to create an interface a user can interact with.

H

Headlessness / Headless Operation

Headlessness is the ability of a component to render itself without drawing to a screen or having any rendering system present. No peripherals need to be present during a Headless Operation.

Hex Values (color)

Hex Values are a way to define colors. Hex values are presented in the format #RRGGBB, #RRRRGGGGBBBB, or #N.

HSB color

HSB values are a way to define colors using their hue, saturation, and brightness as parameters. HSB values are set in a GUI component, using triple digit numbers to represent each HSB value.

I

Inheritance

The variables and methods defined in a class are passed down, or inherited, by their subclasses. This concept is known as Inheritance.

Inheritance Hierarchy

An Inheritance Hierarchy is the diagram that represents, in a hierarchical fashion, classes and subclasses, starting with a root class.

Inset

An Inset is an object that has been placed in such a way that it is intended to obscure part of a frame, but not actually overlay the contents of the frame.

Instantiate

Instantiating refers to the production of a particular object from its class template or definition.

Interface

An Interface is what is seen and used or manipulated on the computer screen by a user (e.g. IBM's Visual Age).

Internationalization

Internationalization is the process of making software that is ready for adaptation to various languages and regions without engineering changes. JClass products are internationalized if you purchase the source code.

J

JAR

Java Archive

A JAR is a compressed file which includes all classes necessary for a Java applet to work.

Java Application

A Java Application is a standalone Java program run by the Java virtual machine.

JavaBeans

Java Beans are logical, portable, platform-independent, and reusable components that are written and used by Java.

See [Component](#).

Java DataBase Connectivity (JDBC)

JDBCs provide database connectivity, which will allow you to query the database.

JavaServer Pages (JSP)

A JSP returns dynamic content to a client through the use of template data, custom elements, scripting languages, and server-side Java objects. Because JSP is an extensible Web technology, the template data is typically HTML or XML, and the client is often a Web browser.

JDK

Java Development Kit

A JDK is a development environment, created by Sun, which allows a user to create Java code.

JSDK

Java Servlet Development Kit

JSDK refers to software that is intended to make the development of Java servlets easier.

Justification

Justification refers to the alignment of text. Text can be right-justified (text is aligned on the right and ragged on the left), left-justified (text is aligned on the left and ragged on the right), center justified (text is centered on each line and is ragged on both the left and right sides), or full-justified (text is aligned on the left and right)

JVM

Java Virtual Machine

A JVM is a virtual machine that is used to safely execute byte code in Java class files on a microprocessor.

L

Line Spacing

Line Spacing is the additional spacing (in addition to the space required by ascenders and descenders) between lines of text. It is measured from the baseline of a preceding line of text to the baseline of the next line of text inside a paragraph. In typographical jargon, this is referred to as leading.

M

Method

A Method is a function which is defined in a class and, unless otherwise specified, is not static. When a class is instantiated, its Methods will run.

N

Nested Objects

A Nested Object is an object that is referenced by a different object of the same class.

For example, if you have a referenced sub-tree object within another sub-tree, the referenced sub-tree is a Nested Object.

O

Object

An Object is used to contain the variable data and method definitions that are needed to instantiate a class, and is the building block of object-oriented programming.

ODBC

Open DataBase Connectivity

ODBC is a subprotocol used by the JDBC DriverManager to access a database without being noticed.

Operators (Binary and Unary)

An Operator is simply a symbol that is used as a function.

A Binary Operator has two arguments; a Unary Operator has only one.

See [Binary Operator](#) and [Unary Operator](#).

P

Package

A Package is a group of classes or interfaces, and is declared with the package keyword. Often, packages are groups of classes or interfaces that have been grouped together to provide a specific type of functionality.

Page

JClass ServerReport prints data to a Page. The Page is simply the unit where frames are placed, allowing for data to be printed.

Page Break

A Page Break is a flow action that forces the flow to traverse to the first flow frame on a new page. All pending content on the current page is written, and a new page is generated based on the `flowPageTemplate` attribute of the current page.

Page Template

Page Templates are `JCPage` objects whose structure and layout are copied to create new blank pages for a document. A typical template will specify the set of `JCFrames` (position, size, number of columns, etc.) which appear on the page, the set of frames that will be used by the `JCFlow` object to hold document content (and the order), and the

template which is to be used for the document page which follows the one generated from this template.

Page Templates may also be used to refer to the XML data which describes the setup of a template page.

Page Templates are not restricted to page structure, and content in non-flow frames (like headers and footers) can be specified and will be copied onto document pages generated from those templates.

Paragraph Spacing

Paragraph Spacing refers to the amount of space between paragraphs in a document. It is measured from the baseline of the last line of text in the preceding paragraph to the baseline of the first line of text in the following paragraph.

Parameter

A Parameter is a data item specified in a method call. Also referred to as an argument.

Parent Frame

A Parent Frame is a frame that holds other subordinate frames or tables.

Parent Object

A Parent Object is an object on which another object is based.

See [Object](#).

Parser

A Parser determines the syntactic structure of a sentence or String.

Pasted Objects

A Pasted Object is an object added in such a way that it is locked at a set of user-defined coordinates. Pasted objects will always appear at the same location.

PDF

Portable Document Format

PDF allows a document to be represented in a form that is independent from the original software, hardware, and operating system that was used to create it. Adobe System's Acrobat uses this file format, and allows PDF files to contain text, graphics, and images. PDF files are known to be versatile and well suited for distribution.

Point

A Point is a typographical measure of distance. One point is approximately one seventy-second of an inch. Points are the standard measure for specifying the size of fonts.

PostScript (PS)

PostScript is an Adobe format that describes characters and images as a series of Bezier curves.

Printer

In JClass ServerReport, a Printer defines the type of print output produced. For example, a PDF printer would produce a PDF document.

Procedure

A Procedure is the instructions used to perform a specific task.

Property

Properties are the named method attributes of a class that can affect its appearance or behavior.

Property Parameters

Property Parameters are programming options referring to parameters.

See [Parameter](#).

R

Render List

A Render List is created by the order in which components are added at execution time. It represents the order in which child objects are to be drawn.

Render Object

A Render Object records an atomic element of the document layout, a piece of text, an image, a graphical primitive (line, circle) or some other simple object. The render object stores size and position information (location, bounding box) and any other information required to describe the object it represents to a printer.

Because a PageLayout document is first laid out in memory and then printed, it is necessary to store a representation of the document's layout.

Rendering

Rendering is the act of converting high-level object-based description into graphical elements for display (for example, a document).

RGB Value (color)

RGB Values are a way to define colors using red, green, and blue values. RGB values are presented in the format #RRGGBB.

RMI**Remote Method Invocation**

RMI allows one computer running a Java program to access methods and objects that are running in a Java program on another machine.

Root

The Root is the highest level in the hierarchical structure. Root classes and Nodes are the ones that other classes are based on, and are normally very general.

S**Section**

A section is a sub-division of the document JClass ServerReport is to print.

For example, if the document being produced is a book, each chapter would be its own section.

Section Break

A Section Break is a flow action specifying that the current section of the document is ending and another one is about to start. Conceptually, this might occur at the beginning of a new chapter of a book.

Server

The Server is the machine that responds to a client's request by sending a reply message. A Server provides a service to the client.

Servlet

A Servlet is an applet that will be executed on a server, not on a client machine, with the use of a servlet engine. A Servlet extends a Web Server's functionality.

See [Applet](#).

SGML**Standard Generalized Markup Language**

SGML is a markup language used to represent documents through the description of the relationship between a document's content and structure. SGML output is recognized across platforms, and can be shared and reused.

Spanning Cells

Spanning Cells is the process by which the borders between specified cells are eliminated, creating one merged cell.

SQL**Structured Query Language**

SQL is a programming language used primarily on relational database managements systems for creating, updating, and querying.

String

A String is a sequence of alphanumeric, punctuation, and white spaces.

SubClass

A Subclass is a class that is derived from another class. There might be one or more classes between them. A Subclass inherits the variables and methods contained in the superclass.

See [Class](#).

SVG**Scalable Vector Graphics**

SVG refers to XML defined images, that use vectors instead of pixels to define 2D graphics.

Swing

Swing is a set of classes which extend the Java AWT package and is used for creating a graphical user interface.

T**Top-Level Object**

A Top-Level-Object is a Java Object class on which other classes base themselves.

U

Unary Operator

A Unary Operator is an operator that only takes one parameter, which can be an expression or a number.

See [Operators \(Binary and Unary\)](#).

Unicode

Unicode, developed by the Unicode Consortium, provides a universal character encoding standard for the representation of text, no matter what platform, program, or language you are working with. Unicode provides a unique number for each character.

W

WAR

A WAR file packages information to move your web application anywhere, in a very easy to build and execute format.

Web Server

A Web Server provides services to access the Internet, an intranet, or an extranet. Web Servers are software, and are used to host web sites, provide support for HTTP and other protocols, and execute server-side programs.

Index

A

- abstract
 - class, definition 371
 - method, definition 371
- accessibility 185
- adobe font metrics, definition 371
- All Pages tab 341
- alternate tag 314
- alt-flow tag 253
- applet, definition 371
- argument, definition 371
- array, definition 371
- assistive technology 185
- attribute, definition 372
- AWT Name
 - definition 372
 - mapping to platform-specific fonts 32
- AWT, definition 371

B

- background color
 - table, setting 112
 - text, setting 34
- Base64-encoded images, adding 77
- binary images in XML 77
- binary operator
 - definition 372
 - in JClass ServerReport 195
- body flow 166
- body tag 141
- bold tag 253
- bookmark tag 254
- bookmarks
 - creating 82
 - creating in XML 84
- bookmark-tree tag 254
- border tag 142, 146
- borders
 - cell, setting 116
 - frame, setting in the Designer 340
 - table header, setting 111
 - table, setting 108
- bullet style 42

C

- cascading style sheet, definition 373
- cell
 - See* table cells
- cell tag 255
- chart-data tag 257
- circles, drawing 65
- class, definition 372
- classpath, definition 372
- client, definition 372
- clone
 - definition 373
 - frame, in the Designer 326
 - page, in the Designer 326
- color
 - HSB, setting in the Designer 329
 - page, setting in the Designer 328
 - RGB, setting in the Designer 330
 - setting, in a full page template 144
 - String 251
 - swatches, setting in the Designer 329
- table background, setting 112
- table rows/columns, setting in the Designer 360
- text background, setting 34
- column tag 142, 146
- column-info tag 258
- columns
 - See also* table columns
 - frame, adding 154
 - frame, adding in the Designer 339
- component
 - definition 373
 - importing 67
 - overflow rules 70
- constructor, definition 373
- container, definition 373
- content
 - rotating within a frame 155
 - rotating, in the Designer 339
- contents list, creating 85
- contents-list tag 259
- contents-list-entry tag 260
- counting macro, creating 46
- crimson.jar 244
- CSS, definition 373
- current-text-style tag 262

custom macros 47

D

dash length

- setting, in the Designer 351

data loading, definition 373

data model, definition 373

data source, definition 373

database management system, definition 374

DBMS, definition 374

default-text-style tag 262

derived class, definition 374

Designer

- background information 317

- color

 - HSB, setting 329

 - RGB, setting 330

 - swatches, setting 329

- document

 - creating a basic 325

 - creating a new 320

 - file encoding type, selecting 321

 - saving 321

 - style, creating a new 320

- draw styles

 - cloning 326

 - creating 326

 - creating new 320

 - dash length, setting 351

 - deleting 326

 - Draw Styles tab 349

 - fill rule, setting 351

 - line type, setting 350

 - line width, setting 351

 - name, setting 327

File menu 319

frame

- border, setting 340

- cloning 326

- columns, adding 339

- content, rotating 339

- creating 326

- deleting 326

- location, setting on the page 328

- margins, setting 339

- name, setting 327

- size, setting 328

launching 317

overview 317

page

- All Pages tab 341

- cloning 326

- color, setting 328

- creating 326

- creating a new template 319

- deleting 326

- flow lines, showing 342

- flow, setting 339, 342

- location, setting 328

- name, setting 327

- One Page tab 335

- orientation, setting 338

- Page Template tab 335

- size, setting 328

- template, selecting simple or full 321

- unit of measurement, setting 328

preferences 322

preview work 331

set preferences

- export to a file 324

- home directory 323

- import from a file 324

starting 317

tab

- cloning 326

- creating 326

- deleting 326

- position 347

- setting 346

table styles

- alternating colors, setting 360

- borders, defining 355

- cloning 326

- creating 326

- creating new 320

- deleting 326

- designing 354

- name, setting 327

- Table Styles tab 353

text styles

- cloning 326

- creating 326

- creating new 320

- deleting 326

- designing 344

- font properties, setting 345

- line spacing 344

- name, setting 327

- paragraph spacing 344

- paragraph, indenting 345

- setting, for tables 355

- tabs, setting 346

- Text Styles tab 343

Tool bar 318

Welcome window 317

doc-frame tag 263

document

- creating, in the Designer 325

- draw styles, designing in the Designer 349

- file encoding type, setting in the Designer 321

- flowing content into 163
 - page templates, designing in the Designer 335
 - saving from the Designer 321
 - table styles, designing in the Designer 354
 - text styles, designing in the Designer 344
 - XML, creating 244
- document tag 263
- document tags
 - See* DTD document tags
- document type definition
 - See* DTD
- document-properties tag 264
- dpi, definition 374
- draw styles
 - See also* Designer - draw styles
 - built-in styles 61
 - fill properties, setting 62
 - line properties, setting 61
 - shapes, drawing 62
 - stock styles 61
 - vector-based images, creating 61
 - XML, creating 308
- Draw Styles tab in the Designer 349
- draw-style tag 308
- draw-styles tag 240, 308
- DTD
 - definition 374
 - document tags 252
 - alt-flow 253
 - bold 253
 - bookmark 254
 - bookmark-tree 254
 - cell 255
 - chart-data 257
 - column-info 258
 - contents-list 259
 - contents-list-entry 260
 - current-text-style 262
 - default-text-style 262
 - doc-frame 263
 - document 263
 - document-properties 264
 - embed-chart 265
 - embed-image 267
 - embed-media-clip 269
 - external-java-code 271
 - float-chart 272
 - float-image 274
 - float-media-clip 276
 - flow 278
 - flow-table 278
 - footer-frame 281
 - front-matter 282
 - header-frame 283
 - header-table 284
 - horizontal-rule 285
 - hyperlink 286
 - italic 287
 - list 287
 - list-item 289
 - macro 289
 - mark-location 290
 - meta-data 291
 - new-column 291
 - new-line 292
 - new-page 292
 - new-paragraph 292
 - new-section 293
 - output-contents-list 293
 - overflow-rules 294
 - page-table 296
 - paragraph 300
 - poster 300
 - row 301
 - section 302
 - space 302
 - tab 302
 - unsupported-operation 303
 - use-test-style 304
 - draw styles 308
 - draw-style 308
 - draw-styles 308
 - files, list of DTD 237
 - page template
 - body tag 141
 - border tag 142, 146
 - column tag 142, 146
 - first-page-different tag 141
 - flow-frame tag 146
 - flow-page tag 146
 - flow-section tag 142, 146
 - footer tag 141
 - frame tag 145
 - gutter tag 141
 - header tag 141
 - location tag 145
 - margin tag 141, 145
 - overview 151
 - page tag 144
 - simple-page tag 141
 - size tag 141, 145
 - watermark tag 142, 146
 - primer 370
 - ServerReport templates 147
 - symbol styles
 - symbol-style 310
 - symbol-styles 310
 - table styles 311
 - alternate 314
 - table-style tag 312
 - table-styles 311
 - text styles 304

- tab-stop 307
- text-style 305
- text-styles 305

dynamic web content, definition 374

E

embed

- component 67
- images 59
- media clip 68

embed-chart tag 265

embedded object, definition 374

embedding, definition 374

embed-image tag 267

embed-media-clip tag 269

encryption in a PDF document 183

Euro symbol 56

event, definition 374

events 178

exception, definition 375

export to a file from the Designer 324

expression

- definition 375
- in formulas 191

expression list

- definition 375
- in formulas 197

external XML files, referencing 161

external-java-code tag 242, 271

F

file

- DTD, list of 237
- encoding type, setting in the Designer 321
- exporting to, from the Designer 324
- extension, definition 375
- printing to 175

fill properties

- draw style, setting 62
- draw style, setting in the Designer 351

first-page-different tag 141

float

- component 67
- image 60
- media clip 68

float-chart tag 272

float-image tag 274

floating object, definition 375

float-media-clip tag 276

flow

- body flow 166
- definition 376

- example 168
- frame 163
- front matter flow 167
- mechanism, definition 376
- overview 26
- page template for next page 150
- page template for next section 150
- page, setting in the Designer 339, 342
- showing in the Designer 342

flow tag 278

flow-frame tag 146

flow-page tag 146, 150

flow-section tag 142, 146, 150

flow-table tag 128, 278

FlushPolicy for printing 176

font metrics, definition 376

fonts

- adding 49
- AWT names 32
- Euro symbol 56
- font metrics file 49
- map, creating 52
- properties, setting in the Designer 345
- TrueType, setting font properties 54
- working with 32

footer

- creating 153
- creating in XML 208
- creating, in the Designer 338

footer tag 141

footer-frame tag 281

formulas 189

- exceptions 198
- expressions 191
- hierarchy 190
- mathematical operation 198
- operations 194
- results 191

frame

- See also* Designer - frame
- definition 376
- fitting an item within 74
- flow versus static 163
- overflow rules 70
- rotating content 155
- table size fit to 112
- truncating an item to fit 75

frame tag 145

front matter flow 167

front-matter tag 282

full page template, creating in the Designer 321

G

GIF, definition 376

glyph, definition 376
GUI, definition 376
gutter tag 141

H

hand-held devices 185
header
 creating 153
 creating in XML 208
 creating, in the Designer 338
header tag 141
header-frame tag 283
header-table tag 284
headlessness, definition 377
hex values, definition 377
home directory, setting in the Designer 323
horizontal-rule tag 285
HSB color
 definition 377
 setting, in the Designer 329
HTML, converting from XML 246
hyperlink tag 286
hyperlinks
 creating 81
 creating in XML 82

I

icons, importing 60
images
 embedding 59
 floating 60
 importing 59
 overflow rules 70
 pasting 60
 raster-based 59
 resolution 59
import
 component 67
 images 59
 JClass ServerChart 67
 preferences from a file, in the Designer 324
 Swing icons 60
indent paragraphs 35
inheritance hierarchy, definition 377
inheritance, definition 377
inset, definition 377
instantiate, definition 377
interface, definition 377
internal links, creating 79
internationalization
 definition 378
 in JClass ServerViews 55

italic tag 287

J

JAR, definition 378
Java application, definition 378
JavaBeans, definition 378
JavaServer pages, definition 378
jaxp.jar 244
JCAAlternate 108
JCContentsList 88
JCContentsListEntry 85
JCDocument
 overview 23
 XML data, loading 244
JCDrawStyle 108
JCFlow
 instantiating 164
 interaction with JCFrame 149
 overview 26
JCFlowEvent 178
JCFlowListener interface for JCFlowEvents 178
JCFlowTable
 See also table
 overview 94
 versus JCPageTable 92
JCFrame
 in the JCPage 26
 interaction with JCFlow 149
 overview 148
JClass LiveTable, converting 124
JClass ServerChart, importing 67
JList 39
JListBullet 42
JListItem 39
JCNumbertList 40
JCPage
 anatomy 26
 overview 148
JCPageTable
 See also table
 overview 97
 versus JCFlowTable 92
JCPageTemplate 160
JCPDFPrinter 177
JCPDFSecurity 184
JCPrintEvent 179
JCPrintListener interface for JCPrintEvents 179
JCServerReport.dtd
 description 237
 tags 252
JCSymbolStyle 42
JCTab 37
JCTableStyle 102
JCTextStyle 29

JDBC

- converting to table 126
 - definition 378
- JDK, definition 378
- JSDK, definition 378
- JSP, definition 378
- JTable, converting 125
- justification, definition 379
- JVM, definition 379

L

- large documents, printing 176
- line
- dash length, setting in the Designer 351
 - drawing 62
 - properties, setting in a draw style 61
 - spacing, definition 379
 - spacing, setting 36
 - type, setting in the Designer 350
 - width, setting in the Designer 351
- links
- See* hyperlink
- list tag 287
- listeners 178
- list-item tag 289
- lists, defining 39
- loading XML data 245
- location tag 145
- locations
- marking 79
 - marking in XML 80

M

- macro tag 289
- macros
- creating custom 47
 - numbering 45
 - page total 46
 - using standard 44
- mapping fonts 52
- margin tag 141, 145
- margins
- frame, setting in the Designer 339
 - setting, in cells 115
- marked locations 79
- mark-location tag 290
- mathematical operations 198
- MathMatrix class 193
- MathScalar class 192
- MathValues abstract class 191
- MathVector class 192
- measurement units 27

- converting 27
 - setting defaults 27
- media clip, adding 68
- meta-data tag 291
- metadata, setting in PDF documents 181
- method, definition 379

N

- named location 79
- nested
- lists, creating 44
 - object, definition 379
- new-column tag 291
- new-line tag 292
- new-page tag 292
- new-paragraph tag 292
- new-section tag 293
- numbering pages 45

O

- object, definition 380
- ODBC, definition 380
- offset, subscript or superscript 33
- One Page tab 335
- Operations abstract class 194
- operators
- binary 195
 - definition 380
 - unary 194
- ordered list 40
- orientation
- page, setting in the Designer 338
 - setting in simple page template 141
 - setting, in full page template 144
- output-contents-list tag 293
- OutputPolicy for printing 175
- overflow rules 70
- conditional settings 73
 - fitting in a frame 74
 - pasting portion of item 76
 - threshold 73
 - truncating in a frame 75
- overflow-rules tag 294

P

- package, definition 380
- page break
- border precedence 119
 - definition 380
- page tag 144

- page template
 - See also* Designer - page
 - applying 160
 - built-in template styles 147
 - columns, adding to a frame 154
 - definition 380
 - designing 139
 - external XML files, referencing 161
 - flow to page 150
 - flow to section 150
 - full page template tags 144
 - header and footer, adding 153
 - overview 151
 - page numbers, inserting 45
 - rotating content within a frame 155
 - sample 150
 - simple page template tags 140
 - watermark, adding 157
 - XML strings, loading 161, 246
- page total, inserting 46
- page, definition 380
- page-table tag 128, 296
- paragraph spacing, definition 381
- paragraph tag 300
- paragraphs
 - attributes, setting 34
 - indenting 35
 - indenting, in the Designer 345
 - spacing, setting 36
- parameter, definition 381
- parent frame, definition 381
- parent object, definition 381
- parser, definition 381
- password, setting in a PDF document 183
- pasted object, definition 381
- pasting images 60
- PDF
 - compression 176
 - content list, creating 85
 - converting XML to PDF 243, 246
 - definition 381
 - document-level features, setting 181
 - metadata, setting 181
 - password, setting 183
 - printing from the command line (UNIX) 176
 - restrictions 177
 - security, setting 183
 - structured element tags, adding 186
 - structured, creating 185
- points
 - defining on a page 28
 - definition 381
 - units of measurement 27
- polygons, drawing 66
- poster tag 300
- PostScript, definition 382

- precedence
 - border, page break 119
 - borders, drawing 117
- preferences
 - export to a file, in the Designer 324
 - import from a file, in the Designer 324
 - setting, in the Designer 322
- preview, in ServerReport Designer 331
- printer, definition 382
- printing 175
 - FlushPolicy 176
 - from the command line (UNIX) 176
 - JCDocument methods and policies 175
 - large documents 176
 - notification 179
 - OutputPolicy 175
 - policies 175
 - to a file 175
- procedure, definition 382
- property parameter, definition 382
- property, definition 382

R

- raster-based image, adding 59
- rectangles, drawing 63
- reference list, creating 85
- reflowing a document 185
- render list, definition 382
- render object, definition 382
- render objects
 - categories 363
 - subclasses 364
- rendering, definition 382
- results, in formulas 191
- RGB color
 - definition 383
 - setting, in the Designer 330
- RMI, definition 383
- root, definition 383
- rotate frame contents 155
- rounded rectangles, drawing 64
- row
 - See* table rows
- row tag 301

S

- section break, definition 383
- section tag 302
- section, definition 383
- security in a PDF document 183
- server, definition 383
- servlet, definition 383

- SGML, definition 384
- shapes, drawing 62
- simple page template, setting in the Designer 321
- simple-page tag 141
- size
 - frame, setting in the Designer 328
 - page, setting in the Designer 328
- size tag 141, 145
- space tag 302
- spacing
 - line, setting 36
 - paragraph, setting 36
- spanning cells, definition 384
- SQL, definition 384
- static frame 163
- stock draw styles 61, 244
- String, definition 384
- structured PDF 185
- subclass, definition 384
- subscript 33
- superscript 33
- support 19
- SVG, definition 384
- swatches color, setting in the Designer 329
- Swing
 - definition 384
 - icons, importing 60
 - JTable, converting 125
- symbol styles, defining for lists 42
- symbol-style tag 310
- symbol-styles tag 241, 310

T

- tab
 - See also* Designer - tab
 - alignment, setting 37
 - fill, setting 38
 - inserting 37
 - leader, setting 38
 - position, setting 38
- tab tag 302
- table
 - See also*
 - Designer - table styles, JCFLOWTable, JCPAGEtable
 - appearance, customizing 107
 - attributes 93
 - background color, setting 112
 - borders, setting 108
 - cells
 - border precedence 117
 - borders, setting 116
 - customizing 113
 - margins, setting 115
 - mathematical operations 198

- printing data to individual 97, 101
- spanning 121
- vertical alignment, setting 113
- columns
 - alternating colors 108
 - colors, setting 112
 - dominance 107
 - wrapping extra (JCPAGEtable only) 122
- converting 123
- JClass LiveTable 124
- JDBC databases 126
- Swing JTables 125
- header
 - adding 96, 100
 - borders, setting 111
- JCFlowTable, creating 94
- JCPAGEtable, creating 97
- overview 91
- properties 93
- rows
 - adding 96, 98
 - alternating colors 108
 - colors, setting 112
 - dominance 107
 - flowing to next frame 122
- size, fitting to frame 112
- structure 91
- styles
 - built-in styles 102, 356
 - setting 102
 - XML tags 311
- XML, creating 127
- table of contents, creating 85
- table-style tag 312
- table-styles tag 311
- tab-stop tag 307
- technical support 19
- template
 - See* page template
- text styles 304
 - See also* Designer - text styles
 - background color, setting 34
 - paragraph, indenting 345
 - standard
 - list 30
 - standard, using 29
 - subscripts and superscripts 33
 - tab-stop 307
 - underlining 33
 - using and creating 31
 - XML tags 304
- text-style tag 305
- text-styles tag 239, 305
- Tool bar 318
- top-level object, definition 384
- TrueType fonts

- adding 49
- using 54
- TrueTypeFontProperties class 54
- tutorial, XML 203
- Type 1 fonts, adding 49

U

- unary operator
 - definition 385
 - in JClass ServerReport 194
- underlining text 33
- unicode, definition 385
- unit of measurement 27
 - converting 27
 - for a page in the Designer 328
 - setting defaults 27
 - setting, in a full page template 144
 - setting, in a simple page template 141
- UNIX, printing from the command line 176
- unordered list 42
- unsupported operation tag 303
- use-text-style tag 304

V

- vector-based image, creating 61
- vertical alignment in cells, setting 113
- video, adding 68

W

- WAR, definition 385
- watermark tag 142, 146
- watermark, adding 157
- web server, definition 385
- Welcome window, in the Designer 317

X

- xalan.jar 247
- XML
 - background information 369
 - Base64-encoded images, adding 77
 - bookmarks 84
 - converting to HTML 246
 - converting to PDF 243, 246
 - creating a document 244
 - data loading 245
 - definition 375
 - demos 369
 - DTD primer 370
 - DTD tags

- document tags 252
- draw styles 308
- table styles 311
- text styles 304
- escaped characters 370
- examples 369
- external template files, referencing 161
- external-java-code tag 242
- hyperlinks 82
- images 77
- JClass ServerReport Designer 317
- loading document files 245
- locations, named 80
- parsing overview 243
- primer 369
- required parser 244
- sample page template 150
- ServerReport templates 147
- setting up other XML files for applications 238
- specifying objects 238
 - applying a page-template from XML 238
 - applying draw-styles from XML 240
 - applying symbol-styles from XML 241
 - applying table-styles from XML 241
 - applying text-styles from XML 239
- stock draw styles 244
- tables 127
 - setting borders 128
 - spanning cells 129
 - spanning rows 129
 - tags supported in cells 127
- template DTD 151
- tutorial 203
 - complete code 229
 - draw styles 215
 - hyperlinks 213
 - introduction 203
 - lexicon 205
 - macros 215
 - overview of changes 205
 - page template 207
 - sections 210
 - starting an XML document 206
 - tables 215
 - text, adding 211
 - text, formatting 211
- using with JClass ServerReport 203, 369
 - combining linear and structured approaches 252
 - linear usage 251
 - structured usage 251
- XML strings, loading 161, 246
- XSL
 - background information 247
 - definition 375
- XSLT
 - creating transformations 248

definition 375
overview 247
processor 247