Generate PDF files from Java applications dynamically

A step-by-step guide to iText, an open source library that makes PDF creation easy

Amit Tuli
Senior Integration Architect
IBM

Surbhi Agarwal
Staff Software Engineer
IBM

Skill Level: Intermediate

Date: 24 Jan 2006
(Updated 12 Dec 2012)

If your application needs to generate PDF documents dynamically, you need the iText library. The open source iText library makes PDF creation a snap. This article introduces iText and gives a step-by-step guide to using it to generate PDF documents from Java™ technology applications. We create a sample application to better understand iText.
30 Oct 2012 - *Authors updated article to reflect changes for iText version 5.3.0 (original article used iText version 1.3).*

12 Dec 2012 - *Authors provided updated sample code in downloadable JAR file. (See Download.)*

Many applications demand dynamic generation of PDF documents. Such applications range from banks generating customer statements for email delivery to readers buying specific book chapters and receiving them in PDF format. The list is endless. In this article, we will use the iText Java library to generate PDF documents. We'll take you through a sample application so you can do it yourself and understand it better.

Get familiar with iText version 5.3.0

iText is a freely available Java library from http://itextpdf.com/ (see Resources). The iText library is powerful and supports the generation of HTML, RTF, and XML

documents, in addition to generating PDFs. You can choose from a variety of fonts to be used in the document. Also, the structure of iText allows you to generate any of the above-mentioned types of documents with the same code.

The iText library contains classes to generate PDF text in various fonts, generate tables in PDF document, add watermarks to pages, and so on. There are many more features available with iText. It is not possible to demonstrate all of them in a single article. We will cover the basics required for PDF generation. For more detailed information, refer to the documentation from the vendor (see Resources).

We will use Eclipse for the development of our sample application. Being an open source IDE, Eclipse is freely available and quite powerful. You can download Eclipse now (see Resources).

The iText API: Closer look

The com.itextpdf.text.Document is the main class for PDF document generation. This is the first class to be instantiated. Once the document is created, you require a writer to write into it. The com.itextpdf.text.pdf.PdfWriter is a PDF writer. Some of the other commonly used classes are given below:

- **com.itextpdf.text.Paragraph**—This class represents an indented paragraph.
- **com.itextpdf.text.Chapter**—This class represents a chapter in the PDF document. It is created using a `Paragraph` as title and an `int` as chapter number.
- **com.itextpdf.text.Font**—This class contains all specifications of a font, such as family of font, size, style, and color. Various fonts are declared as static constants in this class.
- **com.itextpdf.text.List**—This class represents a list, which, in turn, contains a number of `ListItems`.
- **com.itextpdf.text.pdf.PDFPTable**—This is a table that can be put at an absolute position but can also be added to the document as the class Table.
- **com.itextpdf.text.Anchor**—An Anchor can be a reference or a destination of a reference.

Downloading and configuring iText in Eclipse

Being a pure Java library, iText comes in the form of a JAR file (see Resources). Once you have downloaded the library (let's say, at path C:\temp), the following steps will configure the iText library in an Eclipse environment:

1. Create a new Java project in Eclipse named iText.
2. Right-click on the iText project in Package Explorer view and select **Properties**.
3. Click **Java Build Path**. On the Libraries tab, click **Add External JARs**.
4. Browse to the C:\temp directory and select the itext-5.3.0.jar in this directory.
5. Click **OK**.

iText is now configured, and Eclipse is ready to create Java applications to generate dynamic PDF documents.

Sample application

What can demonstrate any technology better than a working sample, created by your own hands? Now that you have the requisite tools (Eclipse IDE) and libraries (iText library), we are all set to design and develop a sample running program.

Let's create a simple PDF document that contains some basic elements like plain text, colored text with nondefault font, table, list, chapter, section, etc. The purpose of this application is to familiarize you with the way to use iText library. There are plenty of classes that do lots of work for you related to PDF document generation. It will not be possible to cover all those classes. The javadocs of iText are a good source of information on how to use those classes. Let's start coding.

The first step is to create a document. A document is the container for all the elements of a PDF document.

Listing 1. Instantiation of document object

```
Document document = new Document(PageSize.A4, 50, 50, 50, 50);
```

The first argument is the page size. The next arguments are left, right, top, and bottom margins, respectively. The type of this document is not yet defined. It depends on the type of writer you create. For our sample, we choose com.itextpdf.text.pdf.PdfWriter. Other writers are HtmlWriter, RtfWriter, XmlWriter, and several others. Their names explain their purposes self-sufficiently.

Listing 2. Creation of PdfWriter object

```
PdfWriter writer = PdfWriter.getInstance(document, \

new FileOutputStream("C:\\ITextTest.pdf"));

document.open();
```

The first argument is the reference to the document object, and the second argument is the absolute name of the file in which output will be written. Next, we open the document for writing.

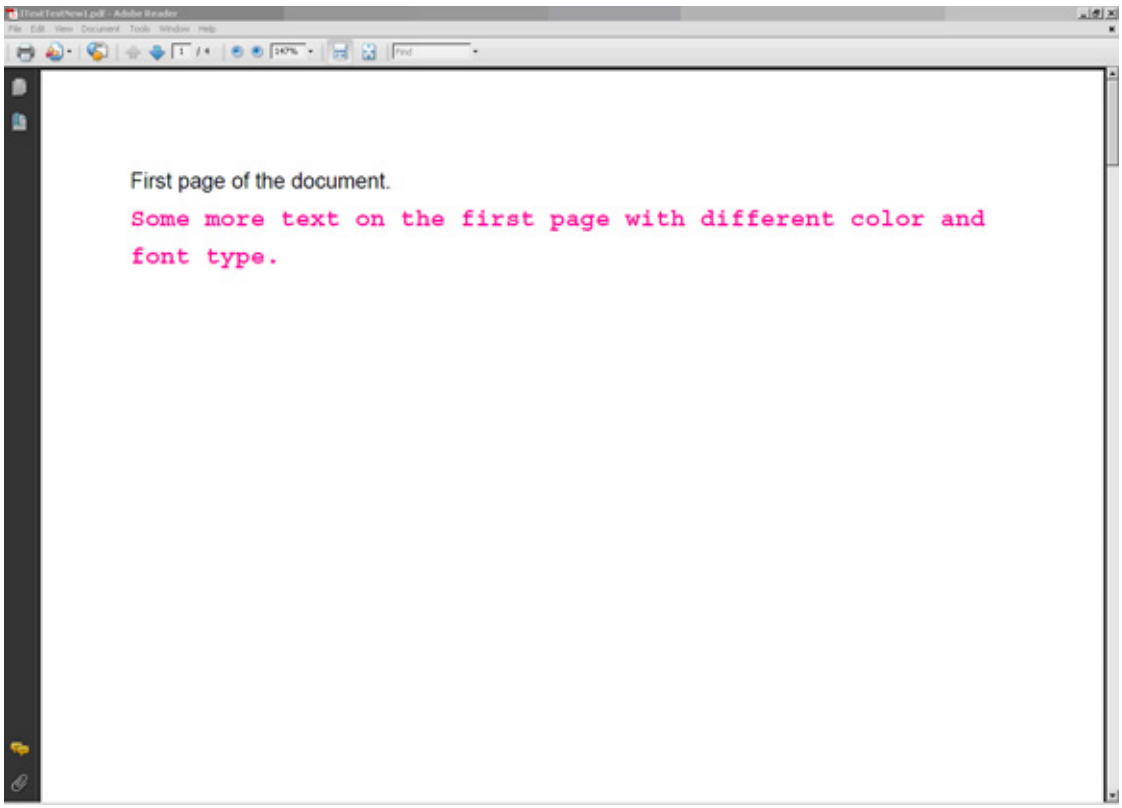
Now, we will add some text on the first page of the document. Any text is added with the help of com.itextpdf.text.Paragraph. You can create a default paragraph with your text and default settings of font, color, size, and so on. Otherwise, you can provide your own font. In this article, we will also discuss the to anchor (link) to the PDF document. In this PDF, we used the backToTop as the link. When you click on the backToTop link, it brings you to the first page of the document. You need to set the text as the anchor target to the first page. Let's see how to set the anchor target and set the font to the added paragraph.

Listing 3. Creation of paragraph object

```
Anchor anchorTarget = new Anchor("First page of the document.");
      anchorTarget.setName("BackToTop");
      Paragraph paragraph1 = new Paragraph();

      paragraph1.setSpacingBefore(50);

      paragraph1.add(anchorTarget);
      document.add(paragraph1);

document.add(new Paragraph("Some more text on the \

first page with different color and font type.",

FontFactory.getFont(FontFactory.COURIER, 14, Font.BOLD, new CMYKColor(0, 255, 0, 0))));
```

Figure 1 shows sample output of the code in Listing 3. To close the document, add `document.close();` at the end of the code in Listing 3.

Figure 1. Sample output of code in Listing 3



You just saw how to add plain text into PDF document. Next, we need to add some complex elements into the document. Let's start with creating a new chapter. A chapter is a special section, which starts with a new page and has a number displayed by default.

Listing 4. Creation of chapter object

```
Paragraph title1 = new Paragraph("Chapter 1",
   FontFactory.getFont(FontFactory.HELVETICA,
   18, Font.BOLDITALIC, new CMYKColor(0, 255, 255,17)));
Chapter chapter1 = new Chapter(title1, 1);
chapter1.setNumberDepth(0);
```

In the code in Listing 4, we created a new chapter object, `chapter1`, with the title "This is Chapter 1." Setting number depth to **0** will not display the chapter number on page.

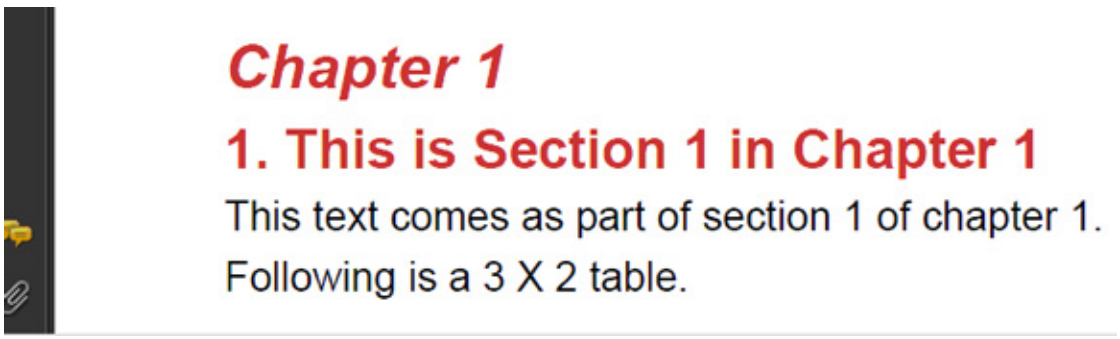
A section is a subelement of a chapter. In the code in Listing 5, we create a section with the title "This is Section 1 in Chapter 1." To add some text under this section, we create another paragraph object, `someSectionText`, and add it to section object.

Listing 5. Creation of section object

```
Paragraph title11 = new Paragraph("This is Section 1 in Chapter 1",
       FontFactory.getFont(FontFactory.HELVETICA, 16, Font.BOLD,
       new CMYKColor(0, 255, 255,17)));
Section section1 = chapter1.addSection(title11);
Paragraph someSectionText = new Paragraph("This
text comes as part of section 1 of chapter 1.");
section1.add(someSectionText);
someSectionText = new Paragraph("Following is a 3 X 2 table.");
section1.add(someSectionText);
```

Before we add the table, let's look at what the document looks like. Add the following two lines to close the document in Figure 2. Then compile and execute the program to generate the PDF document: `document.add(chapter1);document.close();`.

Figure 2. Sample output of chapter



Next, we create a table object. A table contains a matrix of rows and columns. A cell in a row can span more than one column. Similarly, one cell in a column can span more then one row.

Listing 6. Creation of table object

```
PdfPTable t = new PdfPTable(3);

      t.setSpacingBefore(25);

      t.setSpacingAfter(25);

      PdfPCell c1 = new PdfPCell(new Phrase("Header1"));

      t.addCell(c1);

      PdfPCell c2 = new PdfPCell(new Phrase("Header2"));

      t.addCell(c2);

      PdfPCell c3 = new PdfPCell(new Phrase("Header3"));

      t.addCell(c3);

      t.addCell("1.1");

      t.addCell("1.2");

      t.addCell("1.3");

      section1.add(t);
```

In the code in Listing 6, we create a PDFPTable object, `t`, with three columns and keep adding the rows. Next, we create three PDFPcell objects, with different text. We keep on adding them into the table. They are added in the first row, starting from the first column, moving to the next column in the same row. Once the row is complete, the next cell is added to the first column of the next row. A cell can also be added to the table by just providing the text of the cell, such as `t.addCell("1.1");`. At the end, the table object is added to the section object.

Finally, let's see how to add a list to the PDF document. A list contains a number of `ListItem`s. A list can be numbered or non-numbered. Passing the first argument as *true* means you want to create the numbered list.

Listing 7. Creation of list object

```
List l = new List(true, false, 10);

l.add(new ListItem("First item of list"));

l.add(new ListItem("Second item of list"));

section1.add(l);
```

We had added everything to the `chapter1` object. Now we add a image in the java project. We can scale images using one of these Image methods:

- scaleAbsolute()

- scaleAbsoluteWidth()
- scaleAbsoluteHeight()
- scalePercentage()
- scaleToFit()

In Listing 8, we used the scaleAbsolute. And then add the image object to the Section.

Listing 8. Adding Image to the main Document

```
Image image2 = Image.getInstance("IBMLogo.bmp");

      image2.scaleAbsolute(120f, 120f);

      section1.add(image2);
```

The com.itextpdf.text.Anchor class in iText represents an link, either to an external website, or internally in the document. The anchor (link) can be clicked just like a link in a web page. To add the anchor we need to create a new anchor and set the reference to the Anchor target created in the Listing 3. Then add the anchor to the section and add the section to the document.

Listing 9. Adding Anchor to the main document

```
Paragraph title2 = new Paragraph("Using Anchor",

        FontFactory.getFont(FontFactory.HELVETICA, 16, Font.BOLD,

        new CMYKColor(0, 255, 0, 0)));

section1.add(title2);

title2.setSpacingBefore(5000);

Anchor anchor2 = new Anchor("Back To Top");

anchor2.setReference("#BackToTop");

section1.add(anchor2);
```

With no more elements to add to `chapter1`, it's time to add `chapter1` to the main `document`. We will also close the document object here as we are done with the sample application.

Listing 10. Addition of a chapter to the main document

```
document.add(chapter1);

document.close();
```

Running the sample application

1. Download the sample application, j-itextsample.jar (see Download).

2. Unzip j-itextsample.jar in a directory. If you extract it into C:\temp, for example, this places the source and class files into C:\temp\com\itext\test.
3. Open a command prompt and change the directory to C:\temp.
4. Set your system's classpath on this command prompt. Include `C:\temp\itext-5.3.0.jar` in system's classpath. On Windows®, execute the command `set classpath=C:\temp\itext-5.3.0.jar;%classpath%`.
5. Run the application with the command `java com.itext.test.ITextTest`.

The program will generate the ITextTest.pdf document at C:\. Figure 3 shows a screen capture of the first page of the PDF document.

Figure 3. Screen capture of PDF document

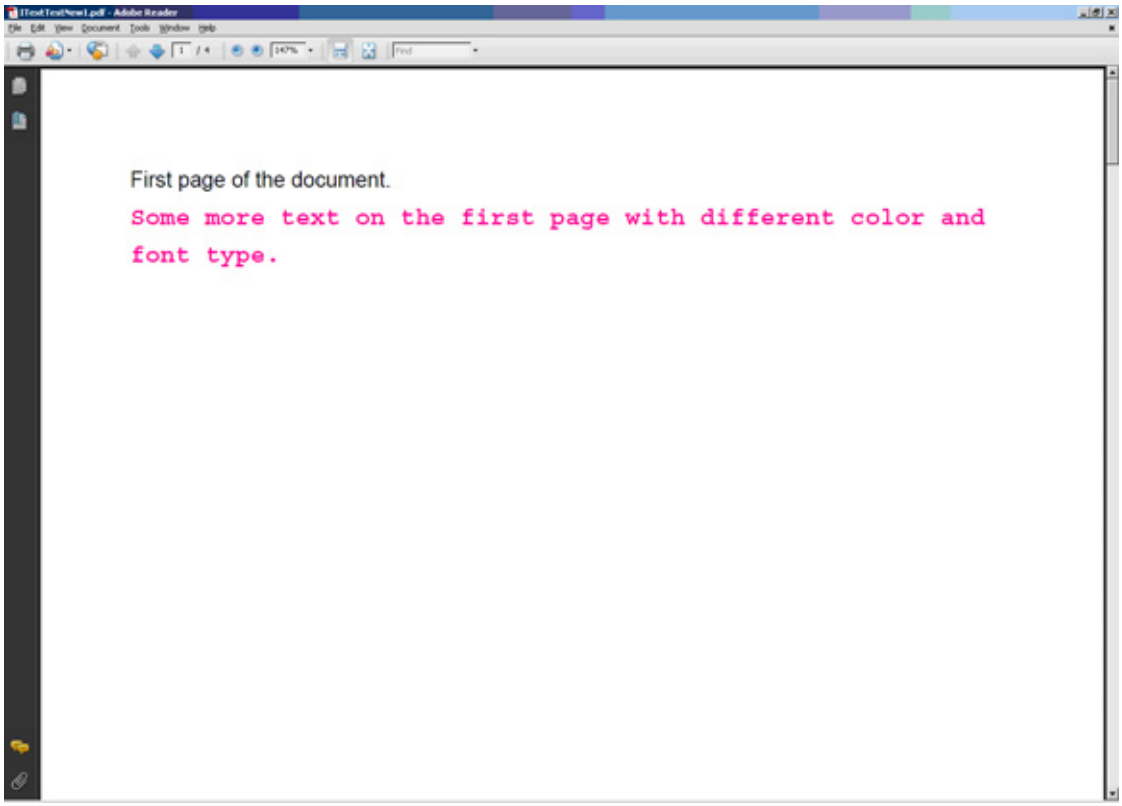


Figure 4 shows a screen capture of Chapter 1 and its section, text, table, list, and image in the PDF document.

Figure 4. Screen capture of PDF document

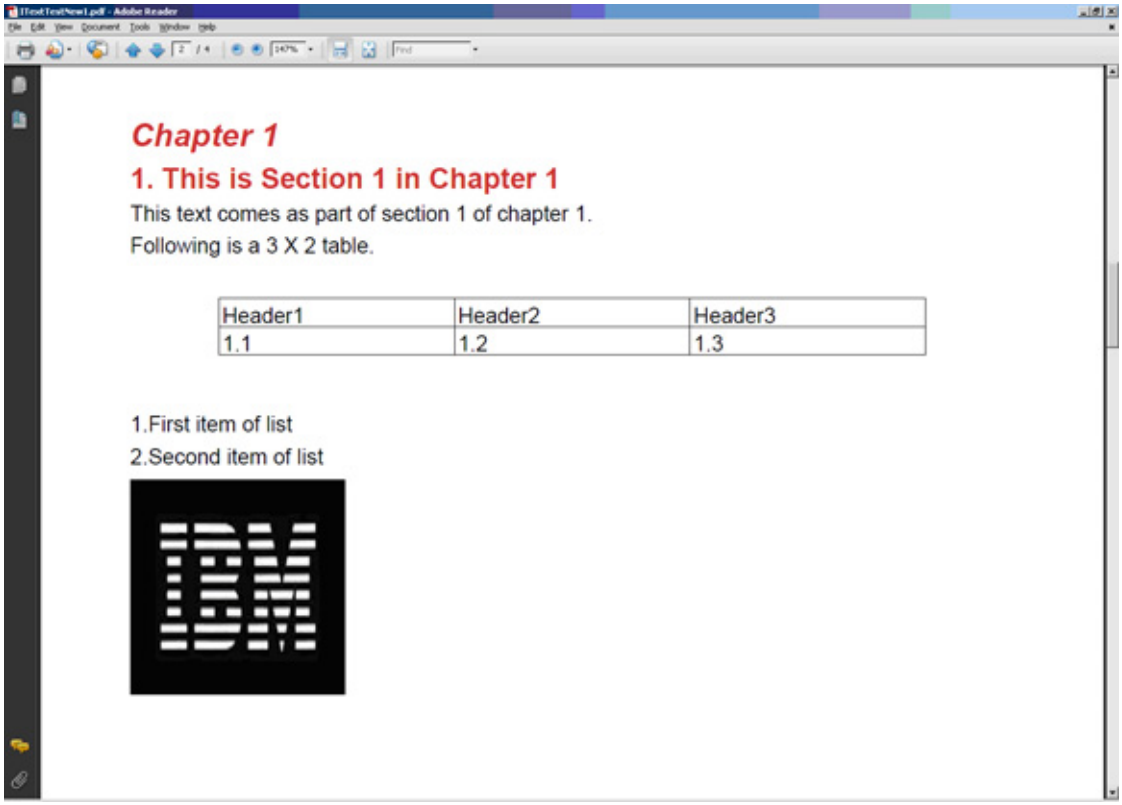
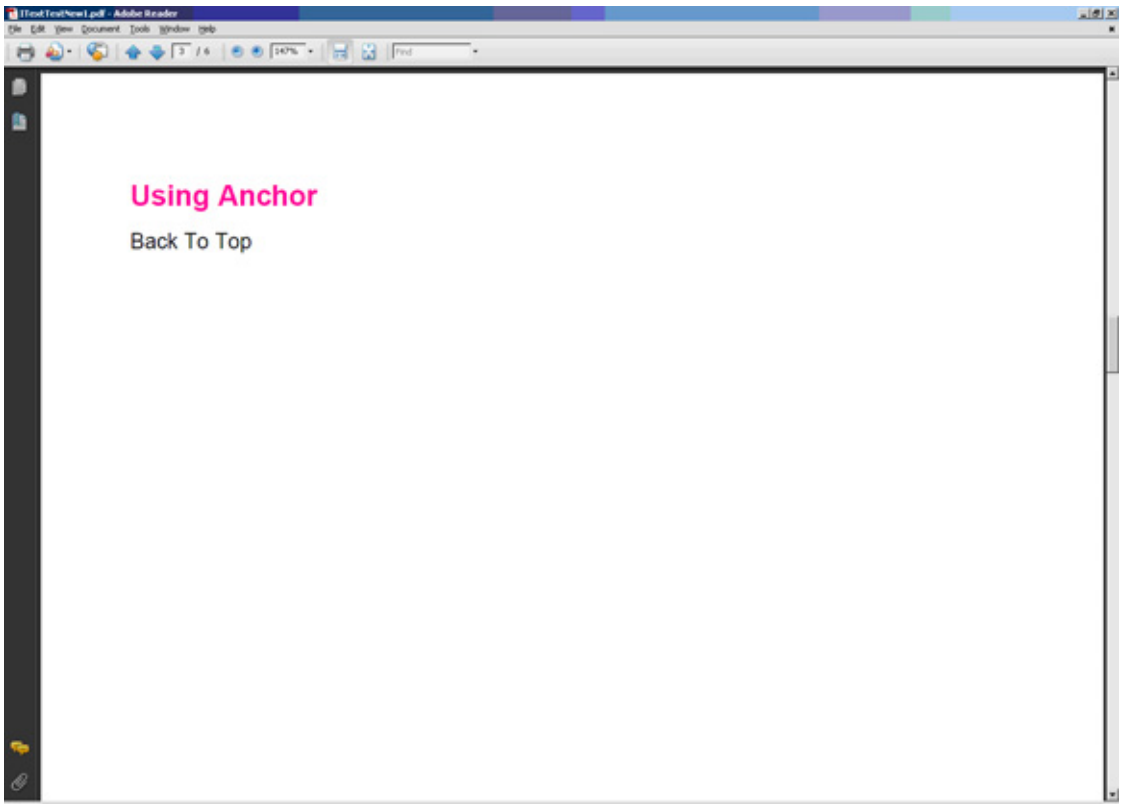


Figure 5 shows a screen capture of the Anchor link in the PDF document.

Figure 5. Screen capture of PDF document



Conclusion

You have just seen some basic elements of PDF generation. The beauty of iText is that you can use the same element's syntax in different types of writers. Also, you can redirect the output of the writer to the console (in the case of XML and HTML writers), the output stream of servlets (in the case of a response to web requests for PDF documents), or any other kind of OutputStream. iText also comes in handy in those situations where the response is the same, but the type of response varies from PDF, RTF, HTML, or XML. iText allows you to create watermarks, encrypt the document, and other output niceties.

Downloads

Description	Name	Size	Download method
Sample code, updated	os-javapdf-itextsample.jar	14KB	HTTP

Information about download methods

Resources

Learn

- Get more information about iText.
- Tutorial: iText by Example contains, as its name implies, tutorials, as well as a list of iText's features.
- Visit Eclipse.org for comprehensive information about the program and how to use it.
- "Getting started with the Eclipse Platform" (David Gallardo, developerWorks, November 2002) provides a history and overview of Eclipse, including details on how to install Eclipse and plug-ins.
- Stay current with developerWorks technical events and webcasts.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Check out upcoming conferences, trade shows, webcasts, and other Events around the world that are of interest to IBM open source developers.
- Follow developerWorks on Twitter.

Get products and technologies

- As someone interested in development with Eclipse, you might want to check out a trial of IBM's Rational Application Developer Standard Edition, a commercial development tool built on Eclipse technology.
- Access IBM trial software (available for download or on DVD) and innovate in your next open source development project using software especially for developers.

Discuss

- Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis. Help build the Real world open source group in the developerWorks community.

About the authors

Amit Tuli

Amit Tuli is an IBM Certified Senior IT Architect with IBM Global Business Services. He is an active author of articles and IBM Redbooks publications as well as many reusable assets. Amit is an active speaker at technical conferences. Almost all 12 years of experience has been with IBM, ranging from product development teams, IBM research, and product services to client engagement teams. He has provided integration oriented architectural solutions to some of the largest and renowned customers in India and other countries for some of the most complex transformations.

Surbhi Agarwal

Surbhi has over 7 years of experience in the IT industry. She has wide experience working with the AIX product team, working with IBM customers on challenging integration projects, and providing solutions around IBM middleware products. She has good end-to-end experience in complex transformation programs. Surbhi has deep expertise in IBM business integration and databases technologies which include IBM DB2, Oracle, Solid Db, WebSphere MQ, WebSphere Message Broker, WebSphere Process Server, WebSphere Adapters, WebSphere Business Monitor, and WebSphere Business Modeller. She is also IBM AIX Certified Professional.