
JADE 6

 JADE Development Centre

Printing in JADE


In this paper . . .

INTRODUCTION	1
A SIMPLE REPORT	2
<i>Single page report</i>	2
<i>Multiple page report</i>	6
<i>Headers and footers</i>	6
<i>Multiple frames per page</i>	9
<i>Frame positioning</i>	9
<i>Free-format printing</i>	10
<i>Conclusion</i>	15
PREVIEWING YOUR OUTPUT	17
THE PRINTING CLASSES	19
PRINTER SETUP	21
PROGRESS DIALOG	22
MULTIPLE PAGE FIELDS	23
THIN CLIENT CONSIDERATIONS	29

Introduction

This paper provides a practical guide about how you can generate printed output from your JADE system outside of the JADE Report Writer.

We start with the simplest of single page reports. To the basic report, additional features are added, one feature at a time. These features cover topics such as multiple pages, headers and footers, and free-format graphics.

Later in this paper we introduce JADE's print preview, print progress, and printer setup dialogs. We also discuss the role of the JADE classes associated with printing.

The article concludes with a note on things you should know when printing in thin client mode.

A Simple Report

Before we can generate a report, we need some test data. For this example, we will be using an application consisting of a company, its branches, and its customers as defined in the following (Figure 1) data model diagram.

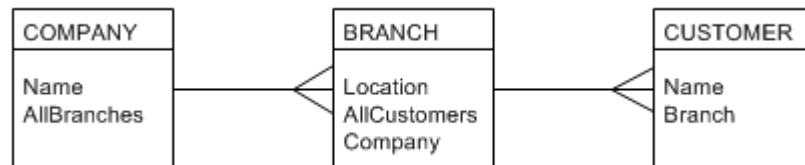


Figure 1 – Data Model of test data.

If you intend trying the code presented in this article, you will need to create instances of these classes. We will have one instance of **Company** with approximately three instances of **Branch**. Two of these branches will have two or three customers, but the third branch will need approximately 30 customers.

Single page report

The report we are going to produce will be very simple. It will show the number of customers belonging to the first branch found for our company, as shown in Figure 2.

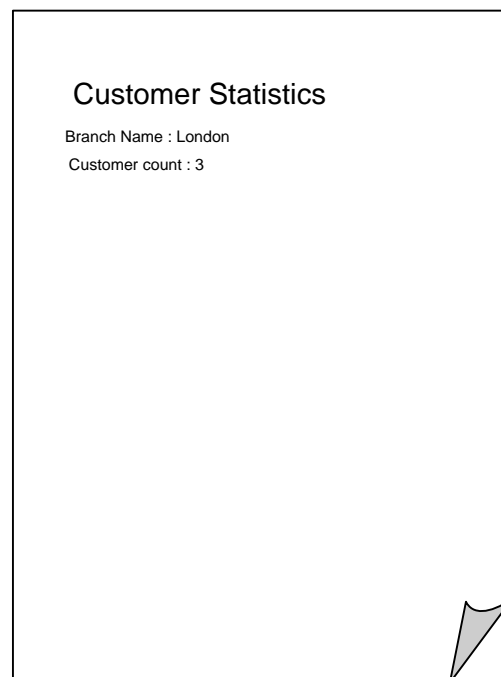


Figure 2 – Sample of the report we intend creating.

There are two steps involved in producing our report

1. Design the report layout in the JADE Painter
2. Create JADE logic to read the data and produce the output

Using the JADE Painter, we specify the design or layout of our report. When creating a new form in the Painter, the New Form Dialog prompts us for a 'Form Style', our choices being printer, screen, and web. For the generation of reports, you would normally select a printer style, although a report can be generated using any style of form. If you have chosen the printer style, during painting a number of minor differences can be seen when compared with a screen form. In particular, the form background will be white and frames will be printed without their normal raised border. Controls will also generally be shown with a light gray border (which isn't printed at run time). However screen and printer forms are, from the JADE model point of view, identical.

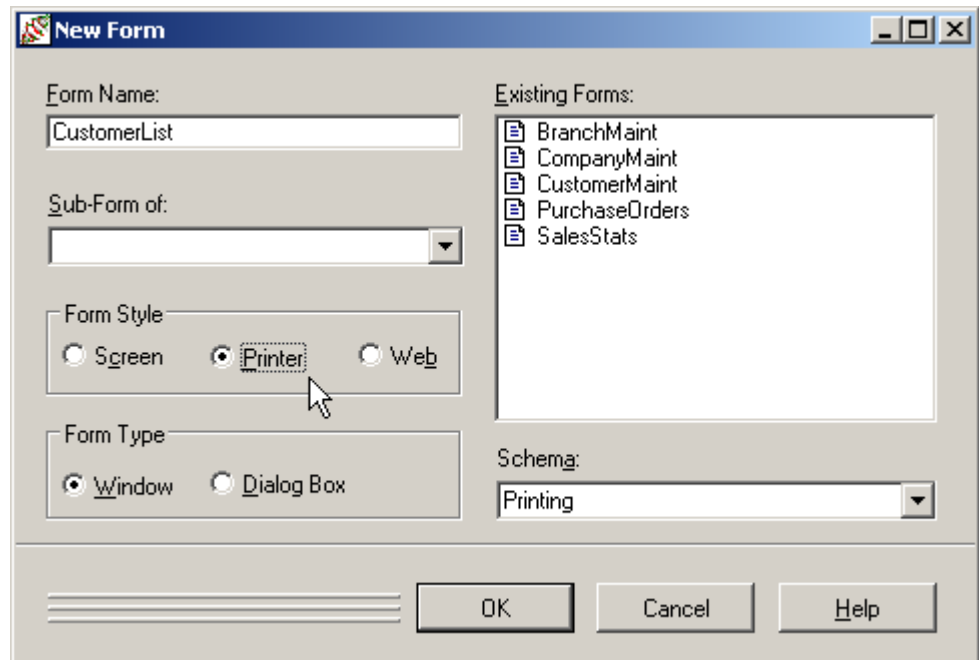


Figure 3 – New Form dialog showing selection of 'Printer' style.

The JADE printing commands are based around Frames, that is, generally you are printing the contents of a **Frame** control. This allows a group of controls to be printed in one operation.

We will now start building a report form. Add a frame to the form created above (name=fraBody). Anywhere will do, as the vertical position is irrelevant to printing, while the horizontal position will determine the distance from the left margin of the paper. Notice that no border is displayed (compared with a frame painted onto a screen-style form). However, the Painter shows a light gray line around the control, which makes it easier to find the area of a given control. Onto this frame, we now paint three labels (**lblTitle**, **lblBranchName**, and

lblCustomerCount). Using the Painter Properties dialog, make the following adjustments to the controls.

1. Remove the caption text from the frame.
2. Increase the font size of our title label (to Arial 20 points) and set the caption text to **Customer Statistics**.
3. Stretch the remaining two labels so that they have room to display **Branch Name : xxxxxxxxxxxx** and **Customer Count : xx**

The layout of the saved form, is shown in figure 4.

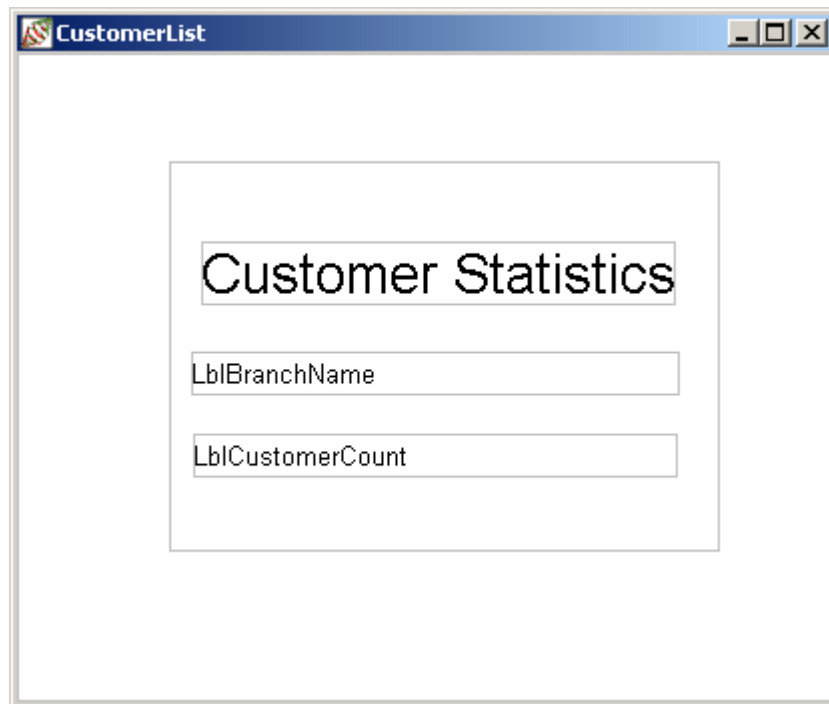


Figure 4 – Form as seen in the JADE Painter.

We are now ready to generate the report. We will be creating a **JadeScript** method to generate our report. We will need one local variable of the type of our report and one to reference the branch we want the statistics generated for, as shown below.

```
vars
  rpt : CustomerList;
  br  : Branch;
```

First we must create the report form. There are two ways of doing this. The first is the same way you would normally create a form; that is, using the create command, as follows.

```
create rpt;
```

Alternatively, use the **GUIClass** class **createPrintForm** method, as follows.

```
rpt := CustomerList.createPrintForm;
```

Your choice will make no difference in the output produced or what you can do. However, the second option will use less system resources and is therefore the preferred option.

Now generate the data to be printed

```
br := Company.firstInstance.allBranches.first;
rpt.lblBranchName.caption := "Branch Name : "
    & br.location;
rpt.lblCustomerCount.caption :=
    "Customer count : "
    & br.allCustomers.size.String;
```

Printing commands are issued against an object that is an instance of the JADE **Printer** class. You may create as many instances of this class as you like (they can only be transient). However, in most cases you can use the JADE-created instance that is accessible from the **Application** class from the **printer** property.

We can now issue the **print** command followed by a **close** command, which will send the output to the printer.

```
app.printer.print(rpt.fraBody);
app.printer.close();
```

During development and also when working with this example, it can be useful to preview our printed output. This allows us to view the report without physically printing it until we are happy with its layout. The preview form displayed has a **Cancel** button, which, when pressed, will stop the physical printing and therefore save a tree or two while we are learning about report functionality. (Alternatively, the **Print** button will send the previewed document to the physical printer.) We will have a closer look at print previewing later in this article. Print previewing is enabled by setting the **printPreview** property of our printer object to **true**.

For completeness, the **JadeScript** method that generates the report shown in Figure 2 is reproduced here (with previewing) in full.

```
customerStatsReport ();

vars
    rpt : CustomerList;
    br  : Branch;

begin
    // Create report form and set for printPreview
    rpt := CustomerList.createPrintForm;
    app.printer.printPreview := true;

    // Generate data to output
    br := Company.firstInstance.allBranches.first;
    rpt.lblBranchName.caption := "Branch Name : "
        & br.location;
    rpt.lblCustomerCount.caption :=
        "Customer count : "
        & br.allCustomer.size.String;
```

```
// Generate the report
app.printer.print(rpt.fraBody);

// Send to printer or preview
app.printer.close();

end;
```

Multiple page report

The **Printer::newPage** method can be used to force a new page to be generated (unless we are already positioned at the top of a new page). Using this method and multiple prints of our print frame, we can now modify our report so that all branches are reported on, each one on a new page.

```
multiPageCustomerStatsReport();

vars
  rpt : CustomerList;
  br  : Branch;

begin
  // Create report form and set for printPreview
  rpt := CustomerList.createPrintForm;
  app.printer.printPreview := true;

  foreach br in Company.firstInstance.allBranches do

    // Generate data to output
    rpt.lblBranchName.caption := "Branch Name : "
      & br.location;
    rpt.lblCustomerCount.caption :=
      "Customer count : "
      & br.allCustomers.size.String;

    // Generate the report on new page
    app.printer.newPage();
    app.printer.print(rpt.fraBody);

  endforeach;

  // Send to printer or preview
  app.printer.close();

end;
```

After running this method, the Print Preview form will be displayed. You can use the page buttons (**Next**, **Previous**, **Last**, **First**, and **Specific**) to browse through the pages of the report.

Headers and footers

When printing on multiple pages, it often becomes a requirement to print a *header*, a *footer*, or both. These are small areas at the top (header) and bottom (footer) of each printed page that may include such details as page number, date, report name, and company logos.

Headers and footers are created in the same way as the body of a report is created; that is, through the JADE Painter. Returning to the form originally painted, we add another two frames (named **fraHeader**, and **fraFooter**). In the header frame, add a label and from the Properties dialog set the **alignment** property to *right, middle* and its **formatOut** property to **=date**. For the footer frame, add another label and again in the Properties dialog set the **alignment** property to *center, middle* and its **formatOut** property to **=pagenofm**.

The **formatOut** property of **TextBox** and **Label** controls can be set to special values that will be determined at run time. These values are listed in the following table.

Option	Action
=date	Prints the current date as specified in Control Panel.
=direct	Sends the text of the control formatted in the font of the control directly to the printer. This provides you with the ability to send commands to the print driver; for example, the facsimile (fax) number when printing to a fax device.
=formatdate	Prints the date in the format supplied, as shown in the following example: =formatdate dd/MM/yyyy
=longdate	Prints the current date in the long date format.
=page	Prints the current page number.
=pagenofm	Prints the current page number of the total number of pages in the document (for example, 2 of 8).
=shortdate	Prints the current date in the short date format.
=time	Prints the current time (in hh:mm:ss am / pm format).
=totalpages	Prints the total number of pages in the document (for example, 8).

Our painted form should now look something like that shown below (Figure 5). Note the position of the label controls inside the header and footer frames. Also notice that the frame is higher than its child label controls. This will have the effect of reserving space between the header and footer text and the body of the report.

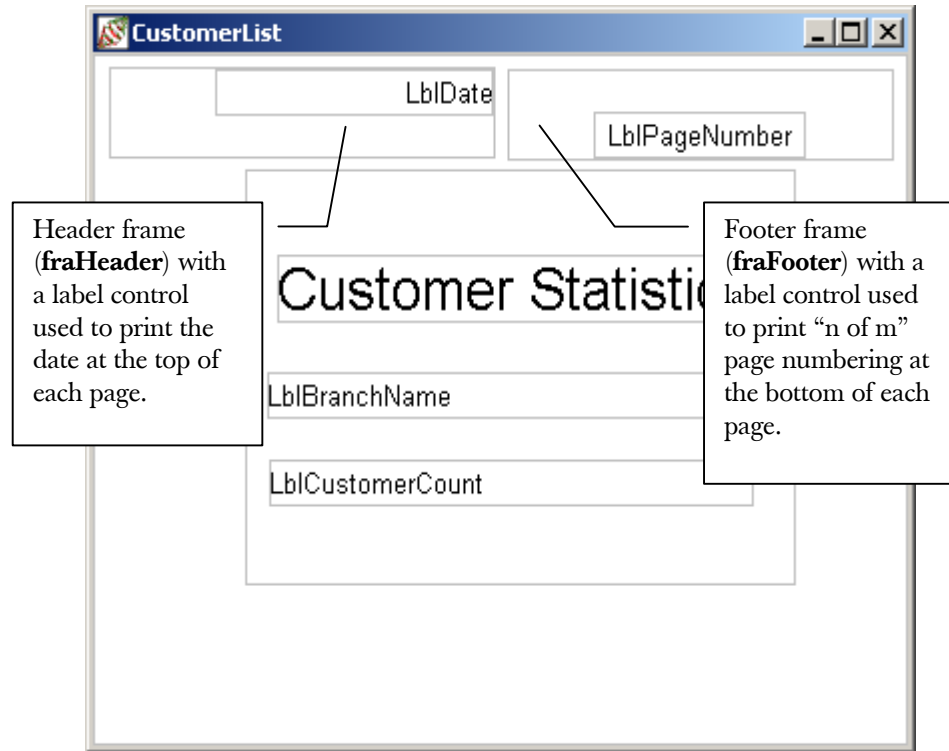


Figure 5 – Showing header and footer frames added in painter.

Having created the frames we intend to use as a header and footer, we must tell the printer class which is which. This is done by logic, using the **setHeader** and **setFooter** methods of the **Printer** class.

```
app.printer.setHeader(rpt.fraHeader);
app.printer.setFooter(rpt.fraFooter);
```

Some adjustment of the position of the frames may be necessary. During painting, we positioned the labels in the frames and specified the actual caption position within those labels. However, the final print position will also depend on the position of the frame. A frame's vertical position is determined at run time (more on this later in this article), however the horizontal position is that specified by the **left** property of the frame. We want our page number footer to be centered at the bottom of the printed page and the header up against the right side of the page. For the footer, we can use the **centreFrame** method. We can also use this method to center the report body within the page. For the header, we must calculate the frame position (that is, a **left** value) to get it positioned up against the right side of the form. The logic required to do this is

```
rpt.fraHeader.left := app.printer.pageWidth
                    - rpt.fraHeader.width;
app.printer.centreFrame(rpt.fraFooter);
app.printer.centreFrame(rpt.fraBody);
```

Add the above header and footer logic to the **JadeScript** method and run the report to display the print preview form. You will notice the *n* of *m* pages printed at the bottom of each page and a date at the top right side of each page.

Multiple frames per page

There is no limit on the number of frames that can be printed on a page. You can print many of the same frame (changing the data each time!) or a number of different frames.

Back in the JADE Painter, add another frame (name=**fraCustomer**) and to that a label (name=**lblCustomerName**). Make the label large enough to hold a customer name, but shrink the frame down so that it is just a little higher than the label and remove its caption. This frame is going to be used to print the customer names for each customer in each branch. The following logic is then used to generate the list of customer names.

```
foreach cust in br.allCustomers do
    rpt.lblCustomerName.caption := cust.name;
    app.printer.print(rpt.fraCustomer);
endforeach;
```

Once again, run the report and view the preview. There are two things you may notice. First, each frame is printed immediately after the previous frame and so filling up the page. Secondly, (assuming you have a branch that has more than 30+ customers) you will notice that, once there is no room for another frame to be printed, an automatic page throw is performed. This means the footer on the current page is generated and a new page is started (with headers).

Frame positioning

The horizontal positioning of a frame is based on the frame's **left** property. This can of course be overridden in logic. We have already seen this, and also used the **centreFrame** method to position the frame mid-way between the left and right margins of the paper.

The vertical print position is held internally by the printer object and is automatically updated as each frame is printed. By default, each printed frame will be positioned after the previous frame. This default vertical position can be queried by calling the **getPrintPosition** method and overridden by using the **setPrintPosition** method. These methods set and get the next print position as a zero-relative offset to the top margin of the page. When setting the print position, you can specify any position on page, including positions that have previously been printed to. This would allow, for example, one frame to be printed on the left side of the page and another on the right side of the page. The range of print positions is from zero through to that returned by the **pageHeight** property of the printer object.

The **frameFits** method tells us whether a frame will fit in the remaining area of a page (determined from the current print position and the amount of space required to print any page footers).

Returning to our form in the Painter, add another frame (name=**fraMore**) and to that add a label (name=**lblMore**) with its caption set to **More....**

We are going to use this frame to print **More...** just above the footer on the right side of the page if we are still processing customers of a specific branch and there is no room left on the current page for that customer. The vertical position of the frame is calculated by the following.

```
moreVertPos := (app.printer.pageHeight
                - rpt.fraFooter.height
                - rpt.fraMore.height).Integer;
```

This is the value used with the **setPrintPosition** method when we detect the need to print the “More...” label. We can now update our customer name printing logic to look like the following.

```
foreach cust in br.allCustomers do
    rpt.lblCustomerName.caption := cust.name;

    // Can this customer fit on the page? If
    // it won't fit, output "more..."
    if not app.printer.frameFits(rpt.fraCustomer) then

        app.printer.setPrintPosition(moreVertPos);
        app.printer.print(rpt.fraMore);
        app.printer.newPage();
    endif;

    app.printer.print(rpt.fraCustomer);
endforeach;
```

Free-format printing

Printing doesn't have to be done through frames, although this is the easiest way to format and therefore the most common technique. It is possible to write direct to the printed page using a set of graphics drawing properties and methods similar to those defined on the **Window** class. The following tables list those graphic drawing methods and properties defined on the **Printer** class.

Property	Description
drawFillColor	Contains the color used to fill in shapes drawn with the printer graphics methods
drawFillStyle	Contains the pattern used to fill the shapes drawn using the printer graphics methods
drawFontBold	Used when constructing the font used for drawing text
drawFontItalic	Used when constructing the font used for drawing text.
drawFontName	Used when constructing the font used for drawing text
drawFontSize	Used when constructing the font used for drawing text
drawFontStrikethru	Used when constructing the font used for drawing text
drawFontUnderline	Used when constructing the font used for drawing text
drawStyle	Defines the line style for output from printer graphics
drawTextAlign	Contains the alignment used when outputting text on the printer using the drawTextAt and drawTextIn methods
drawTextCharRotation	Specifies the angle in degrees between each character's base line and the x axis of the device
drawTextRotation	Specifies the angle in degrees between the base line of the text output and the x axis of the page
drawWidth	Contains the line width for output from printer graphics methods

Method	Description
drawArc	Draws an elliptical arc on the printer page
drawChord	Draws a chord on the printer page (that is, an arc with the end points joined and the interior filled)
drawEllipse	Draws an ellipse on the printed page
drawFilledRectangle	Draws a filled rectangle on the printed page
drawLine	Draws a line on the printed page
drawPie	Draws a pie-shaped wedge on the printed page
drawRectangle	Draws the border of a rectangle on the printed page
drawRoundRectangle	Draws a rectangle with rounded corners on the printed page
drawSolidRectangle	Draws a rectangle filled with the same color as the border on the printed page
drawTextAt	Draws a text string on the printer page
drawTextIn	Draws a text string with a bounded rectangle on the printer page
drawTextSize	Returns the size of the text on the print page, using the current drawFont property values
drawTextSizeIn	Returns the size of the text in a bounding rectangle on the print page, using the current drawFont property values

This free-format functionality can be mixed with frame printing. We will use free-formatting to mark the end of our report by printing an underlined **END** (complete with shadowing) at the end of our report. Figure 6 shows the underlined text we will output.



Figure 6 – The graphics we want printed to end our report.

First we need to select the font to use by the following logic

```
app.printer.setFontSize := 100;
app.printer.setFontName := "Arial";
```

We need to determine if we have enough space between the last printed frame and the page footer. The **getPrintPosition** method tells us the position to which

printing has already been done, therefore the value of the **pageHeight** property less the **getPrintPosition** method results in the space remaining. The height we require is equal to the sum of the text height, the gap between the underline and text, the shadow offset, and the line width of the shadow. The text height and width can be retrieved by the **drawTextSize** method, as follows.

```
txt := "END";
txtWdth := app.printer.drawTextSize(txt, txtHght);
rqdSpace := txtHght // text height
           + 10 // gap between text & underline
           + 4 // offset of line shadow
           + 6; // shadow line width
```

If the required space is less than the free space on the page, then we throw a new page as follows.

```
if app.printer.pageHeight -
    app.printer.getPrintPosition < rqdSpace then
    app.printer.newPage;
endif;
```

The horizontal and vertical print positions are determined by the following logic.

```
printAtPosY := app.printer.getPrintPosition;
printAtPosX := (app.printer.pageWidth - txtWdth) div 2;
```

We are now ready to print the text. We print the shadow first to get the required shadow effect.

```
app.printer.drawTextAt(txt, printAtPosX+4,
                      printAtPosY+4, Gray);
app.printer.drawTextAt(txt, printAtPosX,
                      printAtPosY, Black);
```

The underline is produced by similar logic except we are using line drawing instead of text drawing. Note that we are not using **getPrintPosition** to determine the vertical print position, as the value returned is not updated by free-format printing.

```
app.printer.drawStyle := Printer.DrawStyle_Solid;
app.printer.drawWidth := 6;
app.printAtPosY := printAtPosY + txtHght + 10;
app.printer.drawLine(printAtPosX+4,
                    printAtPosY+4,
                    printAtPosX+txtWdth+4,
                    printAtPosY+4,
                    Gray);
app.printer.drawLine(printAtPosX,
                    printAtPosY,
                    printAtPosX+txtWdth,
                    printAtPosY,
                    Black);
```

For completeness, a **JadeScript** method is reproduced here that will generate the required output.

```
freeFormatPrinting();

vars
```

```
printer : Printer;
printAtPosX, printAtPosY : Integer;
txtWdth, txtHght : Integer;
txt : String;
rqdSpace : Integer;

begin
  printer := app.printer;

  // Get current print position
  printAtPosY := printer.getPrintPosition;

  // Set graphics font
  printer.drawFontSize := 100;
  printer.drawFontName := "Arial";

  // Calculate text height & width
  // and determine space required
  txt := "END";
  txtWdth := printer.drawTextSize(txt, txtHght);
  rqdSpace := txtHght // text height
             + 10 // gap between text & underline
             + 4 // offset of line shadow
             + 6; // shadowline width

  // New page if text won't fit
  if printer.pageHeight - printer.getPrintPosition()
     < rqdSpace then
    printer.newPage();
    printAtPosY := printer.getPrintPosition();
  endif;

  // Print text (centered on page)
  // Do the shadow first (offset by a little)
  printAtPosX := (printer.pageWidth - txtWdth) div 2;
  printer.drawTextAt(txt, printAtPosX+4,
                    printAtPosY+4, Gray);
  printer.drawTextAt(txt, printAtPosX,
                    printAtPosY, Black);

  // Draw the underline - 10 pixels below text
  // Do the shadow first
  printer.drawStyle := Printer.DrawStyle_Solid;
  printer.drawWidth := 6;
  printAtPosY := printAtPosY + txtHght + 10;
  printer.drawLine(printAtPosX+4,
                  printAtPosY+4,
                  printAtPosX+txtWdth+4,
                  printAtPosY+4,
                  Gray);
  printer.drawLine(printAtPosX,
                  printAtPosY,
                  printAtPosX+txtWdth,
                  printAtPosY,
                  Black);

end;
```

In the JADE *Encyclopaedia of Classes and Types*, under the **Printer** class you will find a method reproduced that will generate a calendar page for a specific month and year. The printing is done entirely by free-format printing.

Conclusion

That completes our tutorial on building a report in JADE. Reproduced here is the final logic for our report generation method.

```
customerList();

vars
  rpt : CustomerListWithMore;
  br  : Branch;
  cust : Customer;
  moreVertPos : Integer;

begin
  // Create report form and set for printPreview
  rpt := CustomerListWithMore.createPrintForm;
  app.printer.printPreview := true;

  // Set Header & Footer
  app.printer.setHeader(rpt.fraHeader);
  app.printer.setFooter(rpt.fraFooter);

  // Position frames
  rpt.fraHeader.left := app.printer.pageWidth
                      - rpt.fraHeader.width;
  app.printer.centreFrame(rpt.fraFooter);
  app.printer.centreFrame(rpt.fraBody);
  rpt.fraCustomer.left := 20;
  rpt.fraMore.left := app.printer.pageWidth
                    - rpt.fraMore.width;
  moreVertPos := (app.printer.pageHeight
                - rpt.fraFooter.height
                - rpt.fraMore.height).Integer;

  foreach br in Company.firstInstance.allBranches do

    // Generate branch data to output
    rpt.lblBranchName.caption := "Branch Name : "
                                & br.location;
    rpt.lblCustomerCount.caption :=
                                "Customer count : "
                                & br.allCustomers.size.String;

    // Print the branch heading on new page
    app.printer.newPage();
    app.printer.print(rpt.fraBody);

    // List each customer within the branch
    foreach cust in br.allCustomers do
      rpt.lblCustomerName.caption := cust.name;

      // Can this customer fit on the page? If
      // it won't fit, output "more..."
      if not app.printer.frameFits(rpt.fraCustomer) then
        app.printer.setPrintPosition(moreVertPos);
```

```
        app.printer.print(rpt.fraMore);
        app.printer.newPage();
    endif;

    app.printer.print(rpt.fraCustomer);
endforeach;

endforeach;

// End report with free format printing
freeFormatPrinting();

// Send to printer or preview
app.printer.close();

epilog
    delete rpt;

end;
```

Previewing your Output

Print preview allows you to view your output as it would appear on the printed page if it were actually printed. Print previewing is invoked by setting the **printPreview** property of the **Printer** class to **true** as we did in the example earlier in this article.

An example of the print preview screen is shown in Figure 7

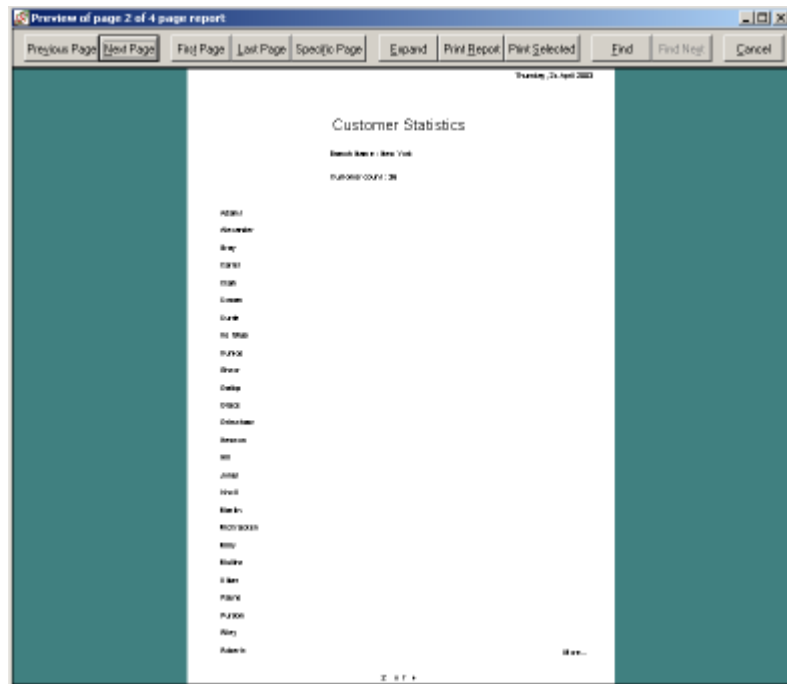


Figure 7 – Report output as seen from the Print Preview form.

From the print preview we can display any page of the report. At any time, we can print the report (via the **Print** button) or select a subset (via **Print Selected** button) of the report to be printed.

Other functionality provided by the print preview form includes

- Searching for specified text
- Zooming in on a page

Three of the **Printer** class properties can be used to control the previewing functionality. These are listed in the following table.

Property	Description
printPreviewAllowPrint	Specifies whether previewed output can be directed to the printer. Default value is true .
printPreviewAllowSelect	Specifies whether the Print Selected button is displayed during print preview. Default value is true .
printPreviewReduce	Specifies whether previewed output is reduced to display a full page on the screen. Default value is true .

The Printing Classes

Here we look at the purpose of the classes defined in JADE relating to printing.

Printer

This class is a transient-only class that handles printing. A transient instance of this is automatically created at run time and is available through the **printer** property of the **Application** class. In your logic, this is likely to be accessed by the following

```
app.printer
```

You can create additional instances of this class if you want to do things such as output to multiple printers or simultaneously create multiple print jobs.

If you have worked through the example earlier in this paper, you will now be familiar with the purpose and some of the functionality of this class.

JadeReport

An instance of this class is automatically created to hold a description of the report output data. This is the data that is used in the print preview or sent to the printer for printing.

Normally you do not see the **JadeReport**. However, you can use the **setReport** method of the **Printer** class to capture it for storage, or manipulation. This **setReport** method must be called before any output is created. Once set, and if **printPreview** of the **Printer** class is **false**, JADE assumes that you want to capture your report output and therefore will not send it to the printer. (If **printPreview** is **true**, output will go to the printer if the **Print** button on the preview form is pressed.)

Each page of output is held as a *meta* file. This is a set of commands that can be replayed to generate the output. Each page of the meta file is held as either an instance of **JadePrintPage** or **JadePrintDirect**. The pages are referenced from the **JadeReport** instance by the **printArray** property.

To copy your transient **JadeReport** data to persistent storage, you require logic similar to that shown below.

```
vars
    saved, tmp : JadeReport;
    pg : JadePrintData;
    :
begin
    // Set JadeReport before any printing
    create tmp transient;
    app.printer.setReport(tmp);
    :    // Your report generation logic
    :
    // Clone report data to persistent storage
```

```
beginTransaction;  
saved := tmp.copySelf(false);  
foreach pg in tmp.printArray do  
    savedJadeReport.printArray.add(pg.copySelf(false));  
endforeach;  
commitTransaction;  
:  
:  
epilog  
    delete tmp;  
end;
```

JadePrintData

This class is an abstract superclass of **JadePrintDirect** and **JadePrintPage**. The **JadeReport** class has a **printArray** property, that is an array of this class.

JadePrintDirect

This class holds the commands that are sent direct to the printer. These commands are created in JADE when the **formatOut** property of **TextBox** or **Label** is set to **=direct**. There can be any number of these associated with an instance of **JadeReport**.

JadePrintPage

A transient instance of this class exists for each page of print output. This output is held as a meta file that contains all the commands necessary to reproduce the page image. There can be any number of these associated with an instance of **JadeReport**.

Printer Setup

Setting up the printer ready for printing is done from the common dialog **CMDPrint**. This dialog is used for both printing and printer setup. The printer setup option is activated by setting the **CMDPrint::printerSetup** property to **true** (the default value).

In printer setup mode, you can select the required printer, number of copies, page size, orientation, and so on. These values will be used by subsequent print requests unless overwritten by your logic.

The initial values for properties of the setup dialog come from your workstation configuration. On printer close (that is, issuing the close command on your printer object), any values altered by the setup will be discarded. However, they can be saved by setting the **retainCMDValues** property of the **Printer** class to **true**. The default value is **false**, meaning property values should revert to their default values.

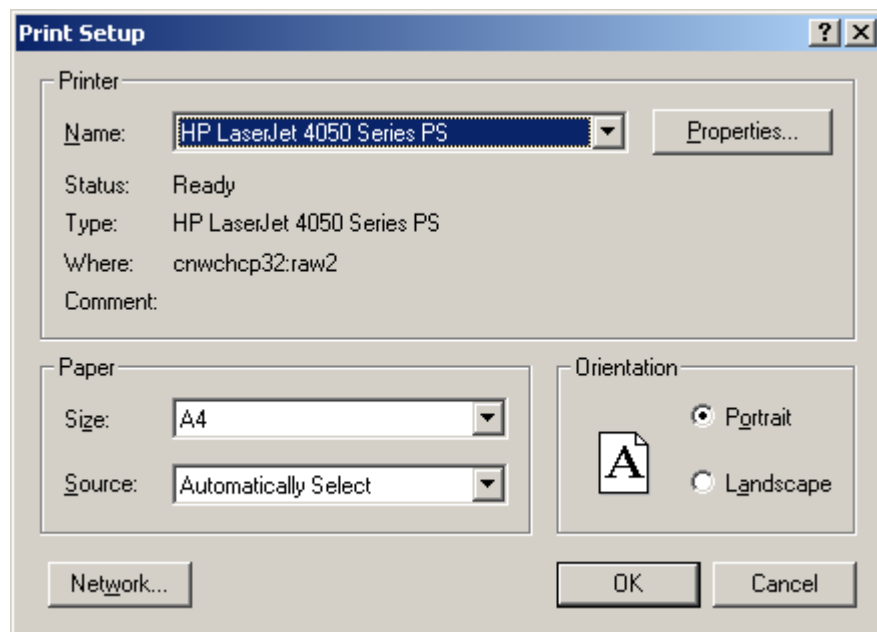


Figure 8 – Example of a Printer Setup dialog form.

Progress Dialog

The Print Progress dialog is displayed on the first call to the **print** method of the **Printer** class. This dialog displays the page number of the page currently being generated, a title, and buttons to allow the user to interrupt the printing.

The Print Progress dialog can be suppressed by setting the **suppressDialog** property of the **Printer** class to **true** (the default value is **false**).

The title shown in the dialog can be set by the **Printer::title**. The application name is used if the title is not specified.

There are two buttons on the dialog for interrupting the report generation.

- The **Cancel** button cancels the print operation and discards the output generated so far. The **print** method returns **Print_Cancelled** in this case.
- The **Stop** button cancels the print operation but does not discard the output. The **print** method returns **Print_Stopped** in this case.

To correctly handle cancelled or stopped printing, your JADE logic should take the appropriate action based on the value returned by the **print** method.

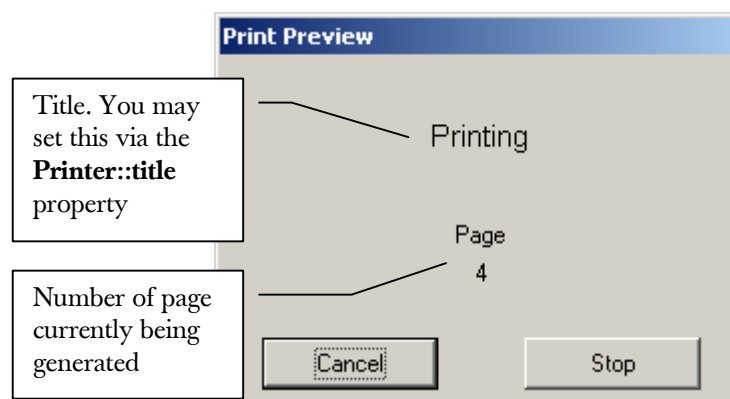


Figure 9 - Example of the Printing Progress dialog form.

Multiple Page Fields

Printing will only print the visible contents of a control. For example, if you have a **TextBox** that has text that extends beyond the bottom of the control, then only the text falling within the control's boundaries will be seen. Sometimes you may want to print a frame that is a variable length and may even span multiple physical pages.

To overcome the problems we will need to break these large blocks of text into smaller pieces. We present here two options on how to divide the contents of a control into printable blocks.

Multiple page TextBox

There are many ways to break up a large block of text into printable pieces. The first method we will show builds up the text to print, line by line, until we detect that adding a new line will mean the text will not fit in the area available in the control. (Assuming the text control is the same size as its containing frame, which in turn has been resized to fit in the available area of the page.) At this point, we print the frame (excluding the last line read), clear the text, and start building up the next page's frame (starting with the last line read).

The example we will build here takes a method and prints its source code. It will have a header showing the method name on the left side and current date on the right side. A footer will show the page number centered at the bottom of the page.

In the Painter, we create the new form. Next we add three frames (**fraHeader**, **fraFooter**, and **fraBody**), deleting the caption of each frame. The **alignChildren** property of the body frame (**fraBody**) is set to *all*. In the header frame, add two labels (**lblHeaderMethodName** and **lblHeaderDate**). From the Properties dialog set the **formatOut** property of **lblHeaderDate** to **=date** and its alignment to *right, middle*. To the footer frame, add the label **lblFooter**, setting its **formatOut** to **=pagenofm** and its alignment to *center, middle*. A text box control (**txtSource**) is added to the body frame. This will receive the logic of the method. A text box control has been used, as it has better handling of the tab character (which is used in the method source) than a label control.

The report form, as seen in the Painter, is shown in Figure 10.

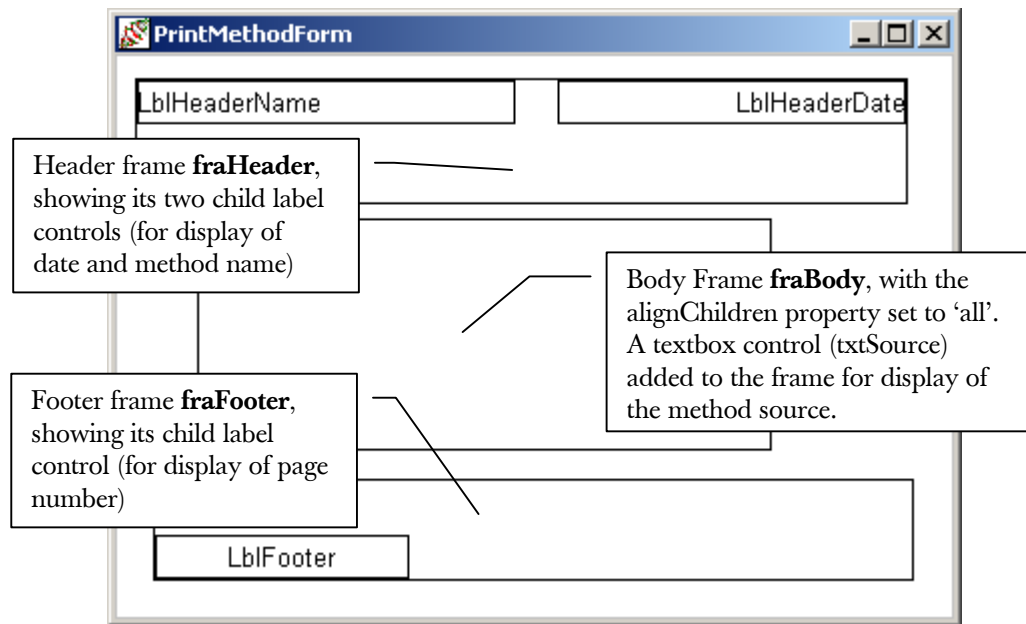


Figure 10 – Painter view of form for method printing example.

In our **JadeScript** method, the frame that is going to display the method source (**fraBody**) is first resized to the space available between the header and footer. The logic of the method is loaded into a string line by line. As each line is read, the text height is measured by the **Window** method **getTextHeightForWidth**. If the height of the text exceeds the height of the text box, the current string contents, without the last line read, is transferred to the text box control and a print performed. This process is repeated until all of the source code has been printed.

The **JadeScript** method to generate the report is also listed here. The method includes comments that describe its workings.

```
methodPrint (mth:Method) ;

vars
    rpt : PrintMethodForm;
    source : String;
    printString : String;
    char : Character;
    rqdHeight : Integer;
    idx : Integer;
    aLine : String;

begin

    rpt := PrintMethodForm.createPrintForm();
    app.printer.printPreview := true;

    // Centre Footer
    app.printer.centreFrame(rpt.fraFooter);
    app.printer.setFooter(rpt.fraFooter);
```

```
// Resize Header to match page width
// and set position of date label to RHS.
// Also set lblHeaderName label to method name
rpt.fraHeader.left := 0;
rpt.fraHeader.width := app.printer.pageWidth;
rpt.lblHeaderDate.left := rpt.fraHeader.width
                        - rpt.lblHeaderDate.width;
rpt.lblHeaderName.caption := mth.getSchemaType.name
                            & "::" & mth.name;
app.printer.setHeader(rpt.fraHeader);

// Resize Body frame to match page width and height
// to be available space between header and footer.
// As fraBody has 'align children = all' then
// source label automatically resized as well.
rpt.fraBody.width := app.printer.pageWidth;
rpt.fraBody.height := app.printer.pageHeight
                    - rpt.fraFooter.height
                    - rpt.fraHeader.height;

// Get method source
source := mth.source;

// Scan the method source line by line building
// up a print string until no more will fit in the
// available space on the page.
idx := 1;
while idx > 0 do
    // Get the next line
    aLine := source.scanUntil(CrLf, idx);

    // If there is another CrLf, add to line just read
    // and set index to the start of the next line.
    if idx > 0 then
        aLine := aLine & CrLf;
        idx := idx + CrLf.length;
    endif;

    // Determine space required for lines read so far
    rqdHeight := rpt.txtSource.getTextHeightForWidth(
                printString & aLine,
                rpt.txtSource.width.Integer);

    // If the last read line causes the height
    // requirement to exceed the available space then
    // transfer the string to the source control and
    // print the frame and reset the print string to
    // the line just read. Otherwise add this current
    // line to the lines for the next print.
    if rqdHeight >= rpt.fraBody.height then
        rpt.txtSource.text := printString;
        app.printer.print(rpt.fraBody);
        printString := aLine;
    else
        printString := printString & aLine;
    endif;

endwhile;
```

```

        // Print remaining text
        rpt.txtSource.text := printString;
        app.printer.print(rpt.fraBody);

        app.printer.close;

    epilog
        delete rpt;

end;

```

Multiple page JadeRichText

Our second example is very similar to the first. Here we are going to print a RTF (Rich Text Format) document over a number of pages. The Painter form is very similar to that of the previous example, the only difference being the use of a **JadeRichText** control instead of a **TextBox** control on the frame **fraBody**.

Once again, the method is reproduced here complete with comments to describe how it works. The point to note is that each line of an RTF file (or control) can be a different height. We use the **JadeRichText::getLine** method to measure a line's height (but don't actually do anything with the text returned by this method).

```

rtfPrint();

vars
    dialog : CMDFileOpen;
    rpt : PrintRTFForm;
    charIndex, lineLen, lineHght : Integer;
    result : Integer;
    lineNum : Integer;
    totalHght : Integer;

begin

    // Ask for RTF file to print
    create dialog;
    dialog.dialogTitle := "RTF file to print";
    dialog.filter := "RTF (*.rtf)|*.rtf|All Files (*.*)|*.*";
    result := dialog.open();
    if result <> 0 then
        return;
    endif;

    // Create report form and specify preview required
    rpt := PrintRTFForm.createPrintForm();

    app.printer.printPreview := true;

    // Centre Footer
    app.printer.centreFrame(rpt.fraFooter);
    app.printer.setFooter(rpt.fraFooter);

    // Resize Header to match page width
    // and set position of date label to RHS.

```

```
// Also set lblHeaderName label to method name
rpt.fraHeader.left := 0;
rpt.fraHeader.width := app.printer.pageWidth;
rpt.lblHeaderDate.left := rpt.fraHeader.width
                        - rpt.lblHeaderDate.width;
rpt.lblHeaderName.caption := dialog.fileName;
app.printer.setHeader(rpt.fraHeader);

// Resize Body frame to match page width and height
// to be available space between header and footer.
// Since fraBody has 'align children = all' then the
// rtf control will be automatically resized as well
rpt.fraBody.width := app.printer.pageWidth;
rpt.fraBody.height := app.printer.pageHeight
                    - rpt.fraFooter.height
                    - rpt.fraHeader.height;

// Load RTF file into RTF control.
rpt.rtfBody.loadFromFile(dialog.fileName,
                        JadeRichText.LoadFromFile_ReplaceAll,
                        JadeRichText.LoadFromFile_RTF);

// Read in lines until no more will fit in available
// space, when that happens, print the frame, delete
// text already printed, and repeat process until
// all the text is printed.
lineNum := 1;
while lineNum <= rpt.rtfBody.lineCount do
    rpt.rtfBody.getLine(lineNum,
                      JadeRichText.GetLine_PlainText,
                      charIndex,
                      lineLen,
                      lineHght);

    // If the line just read, wont fit into the current
    // frame, print the frame and start a new frame
    // beginning with the line just read.
    if totalHght + lineHght >= rpt.fraBody.height then
        app.printer.print(rpt.fraBody);

        // Delete printed text, this will move
        // remaining text 'up'
        rpt.rtfBody.selStart := 0;
        rpt.rtfBody.selLength := charIndex-1;
        rpt.rtfBody.selText := "";

        // We want to start printing at the 'new' line 1.
        lineNum := 1;
        totalHght := 0;
    else
        // This line fits, so remember total height.
        lineNum := lineNum + 1;
        totalHght := totalHght + lineHght;
    endif;

endwhile;

// Print last frame
app.printer.print(rpt.fraBody);
```

```
        app.printer.close;  
epilog  
    delete rpt;  
end;
```

Thin Client Considerations

JADE provides client-side printing only. A page of print output is constructed and can be previewed on the presentation client. The printers that can be accessed are those that are available from the presentation client.

If your logic calls the **Printer::setReport** method to indicate that the report output is to be saved or manipulated in some way, each page of output must be sent from the presentation client to the application server. This normally occurs on a page by page basis as each page is completed. However, if the **formatOut** property of any **TextBox** or **Label** control has been set to **=pagenofm** or **=totalpages**, the report can only be transferred after it has been completed (as the page count is not known until the report is completed).