Chapter 1. Printing from Java applications

Initially, Java was used to create platform independent applications for the World Wide Web. However, as with any tool that gains general acceptance, more and more programmers are using Java to develop business applications. These applications require an interface to generate printed output that is easy to use in a program, is flexible, robust, and yet provides a means to generate complex business documents.

The new LineDataRecordWriter class delivers these features. This document describes this new class as well as other print functions that are provided as part of the IBM Toolbox for Java.

For a complete description of all classes, methods, and parameters, refer to the Java information available from the iSeries Information Assistant on the Internet at: `http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/index.htm`

From the navigation bar, select **Programming->Java->IBM Toolbox for Java**. For general information about the classes, select **Access Classes**. For specific programming information, select **Javadocs for IBM Toolbox for Java classes**. The various classes described in this chapter can be found under `com.ibm.as400.access`.

It is assumed that the reader has some experience with Java programming and understands the basic concepts of classes and methods.

1.1 Class LineDataRecordWriter

The LineDataRecordWriter is a new Class object that is introduced in IBM Toolbox for Java with V5R1.

By using this class, the Java application programmer is mainly only concerned with providing the data fields to the class. The layout of the data on the page is defined external to the application with a form definition and a page definition.

The LineDataRecordWriter class writes a record in line data format, with the name of the record format inserted into positions 1 through 10 of the line data. The record format name corresponds to a Layout command in the page definition. This requires that a page definition using the new record format line data processing be used to format the spooled file that is generated. Record Layout page definitions are described in the Using Form Definitions and Page Definitions chapter.

The records may be written in one of two formats. The FIXED_LAYOUT_LENGTH format is equivalent to standard output generated by most applications. Here, each field has a specified position and length in the record. Data may be aligned left or right within the designated length.

The VARIABLE_LAYOUT_LENGTH format uses a delimiter character to separate fields. This corresponds to the delimiter used in the LAYOUT command in the page definition.

Java applications usually work using the ASCII data streams. The LineDataRecordWriter translates the characters into the Coded Character Set

Identifier (CCSID) of the iSeries server. The CCSID of the iSeries server is stored in the system value QCCSID.

The printer file being generated must have specific attributes. In Java terms, you must set the following parameters:

- ATTR_CONTROL_CHARACTER: Forms Control Character set to *NONE
- ATTR_CONVERT_LINE_DATA: Convert Line Data set to *YES (this not required if you are printing to a device that is configured with AFP(*YES))
- ATTR_FORM_DEFINITION: Form definition integrated file system name
- ATTR_PAGE_DEFINITION: Page definition integrated file system name
- ATTR_PRTDEVTYPE: Printer device type set to *LINE

1.1.1 Sample Java program description

The listing in the next section is a subset of a program from the iSeries Information Center on the Web.

Since this program is a self contained example used for illustrating the programming techniques, the data to be printed is generated within the program and only one record format is used. In a production environment, the data to be printed is a combination of fields entered interactively by a user, extracted from other data bases, or calculated with the Java program. A variety of formats would likely be used.

The program illustrates the following Java instructions necessary to set up the environment and to use the LineDataRecordWriter class:

1. The RecordFormat qcustcdt is defined as RecordFormat().
2. Fields CUSNUM, LSTNAM, BALDUE, and CDTDUE are defined within the record format.
3. The print format of the fields is set by defining the length and alignment.
4. The record format ID is set to string CUSTRECID. This prints in the first 10 bytes of the output record as required by record format line data processing.
5. Comments are added to the program to show how a variable layout length record would be defined with a delimiter.
6. This example uses fixed layout length records.
7. The fields are added to the output record format.
8. Sign on to the iSeries 400.
9. Get the value for ccsid. This is equivalent to the iSeries server value QCCSID. This is used to convert the ASCII data used in the Java environment to EBCDIC, which is used for the LINE data.
10. Define the output queue and printer file parameters.
11. Create a new output spooled file.
12. Create a new instance of the LineDataRecordWriter class called `ldw`.
13. Write the record information into ldw.
14. Close the spooled file.

1.1.2 Sample Java program using the LineDataRecordWriter Class

To find the full listing of this program, look under the in iSeries Information Center under Javadoc for the RecordFormat class. There is another, simpler sample program that can be found by selecting **IBM Toolbox for Java->Access classes ->Data conversion and data description classes->LineDataRecordWriter**.

```
///////////////////////////////////////////////////////////////////////
//
// LineDataRecordWriter example.  This program uses the line data
// record writer access class to create a line data spooled file
// on the AS/400.
//
// The Source code for this program is not published or otherwise
// divested of its trade secrets, irrespective of what has been
// deposited with the U.S. Copyright Office
//
// This source is an example of using the IBM Toolbox for Java
// "LineDataRecordWriter" class.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantablility and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Toolbox for Java
// (C) Copyright IBM Corp. 1999
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import java.io.*;
import java.math.BigDecimal;

public class TestA {

    //Private
    private static int ccsid_           = -1;       // local ccsid variable
    private static AS400 system_        = null;     // the AS/400 system
    private static SequentialFile file_ = null;     // the file

    /**
     **  Create the record field descriptions and record format.
     **/
    public static RecordFormat initializeRecordFormat()
    {
[1]     // Create the record format.
        RecordFormat qcustcdt = new RecordFormat();

        // Create record field descriptions for the record format.
[2]    ZonedDecimalFieldDescription customerNumber =
                        new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,0),
                                                         "CUSNUM");
        CharacterFieldDescription lastName =
                        new CharacterFieldDescription(new AS400Text(8, ccsid_, system_),
                                                      "LSTNAM");

        ZonedDecimalFieldDescription balanceDue =
                        new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,2),
                                                         "BALDUE");
        ZonedDecimalFieldDescription creditDue =
                        new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,2),
                                                         "CDTDUE");

        // assign constants from FieldDescription class
        int justLeft = FieldDescription.ALIGN_LEFT;
        int justRight = FieldDescription.ALIGN_RIGHT;
```

```
[3]     // set the length and alignment attributes for writing the fields
        // The length indicates how many characters the field is, and
        // justification indicates where in the layout field the data
        // should be placed.
        customerNumber.setLayoutAttributes(10,justLeft);
        lastName.setLayoutAttributes(10,justLeft);
        balanceDue.setLayoutAttributes(10,justRight);
        creditDue.setLayoutAttributes(10,justRight);

[4]     // set the record format ID
        String d = "CUSTRECID";
        qcustcdt.setRecordFormatID(d);

[5]     // if this were a variable field length record,
        // we would set the type and delimiter accordingly.  We
        // also would not have needed to specify layoutLength and
        // layoutAlignment values.
        // qcustcdt.setRecordFormatType(RecordFormat.VARIABLE_LAYOUT_LENGTH);
        // qcustcdt.setDelimiter(';');

 [6]     // set the record type to fixed field length
        qcustcdt.setRecordFormatType(RecordFormat.FIXED_LAYOUT_LENGTH);

 [7]    // add the field descriptions to the record format.
        qcustcdt.addFieldDescription(customerNumber);
        qcustcdt.addFieldDescription(lastName);
        qcustcdt.addFieldDescription(balanceDue);
        qcustcdt.addFieldDescription(creditDue);

        return qcustcdt;
    }


    /**
     ** Creates the actual record with data
     **/
    public static void createRecord(Record record)
    {
        record.setField("CUSNUM", new BigDecimal(323));
        record.setField("LSTNAM", "Johnson");
        record.setField("BALDUE", new BigDecimal(25.00));
        record.setField("CDTDUE", new BigDecimal(0.00));
    }


    public static void main(String[]args) {

[8]         // create an instance of the AS/400 system
        system_ = new AS400("SYSTEMA", "JOE", "PGMR");

[9]         // create a ccsid
        ccsid_ = system_.getCcsid();

[10]         // create output queue and specify spooled file data to be *LINE
        OutputQueue outQ = new OutputQueue(system_, "/QSYS.LIB/QUSRSYS.LIB/LDRW.OUTQ");
        PrintParameterList parms = new PrintParameterList();
        parms.setParameter(PrintObject.ATTR_PRTDEVTYPE, "*LINE");
        parms.setParameter(PrintObject.ATTR_PAGDFN,"/QSYS.LIB.QUSRSYS.LIB/LDRW.PAGDFN");
        parms.setParameter(PrintObject.ATTR_CONVERT_LINEDATA,"*YES");


        // initialize the record format for writing data
        RecordFormat recfmt = initializeRecordFormat();

        // create a record and assign data to be printed...
        Record record = new Record(recfmt);
        createRecord(record);

        SpooledFileOutputStream os = null;

[11]    try {
            // create the output spooled file to hold the record data
            os = new SpooledFileOutputStream(system_, parms, null, outQ);
        }

        if (os != null) { // Output stream was created successfully!
            LineDataRecordWriter ldw;
            try {
```

```
[12]            // create the line data record writer
                ldw = new LineDataRecordWriter(os, ccsid_, system_);

[13]            // write the record of data
                ldw.writeRecord(record);
            }
            catch (IOException e) {
                System.out.println("Error occurred writing record data");
            }

[14]        // close the output stream (spooled file)
            try {
                os.close();
            }
            catch (Exception e) {
                System.out.println("Error occurred closing output stream.");
            }
        }
    }
}
```

1.2 Java Report Builder

The Java report writer classes are a set of classes that Java applications can utilize to access and format application data from an EXtensible Markup Language (XML) source file or data produced by servlets or JavaServer Pages(). These classes are included in the licensed program for IBM Toolbox for Java, 5722-JC1, Version 5 Release 1 (V5R1). The Extensible Style Language (XSL) Formatting Objects are used as the language for defining how the application data is to be formatted into a document. The formatting objects are defined within an XSL stylesheet when formatting XML data. The formatting objects are contained within a Java servlet or JavaServer Page (JSP) when formatting data generated from a servlet or JSP. The report writer classes can output the formatted data in one of two document formats, including HP PCL and Adobe's Portable Document Format (PDF).

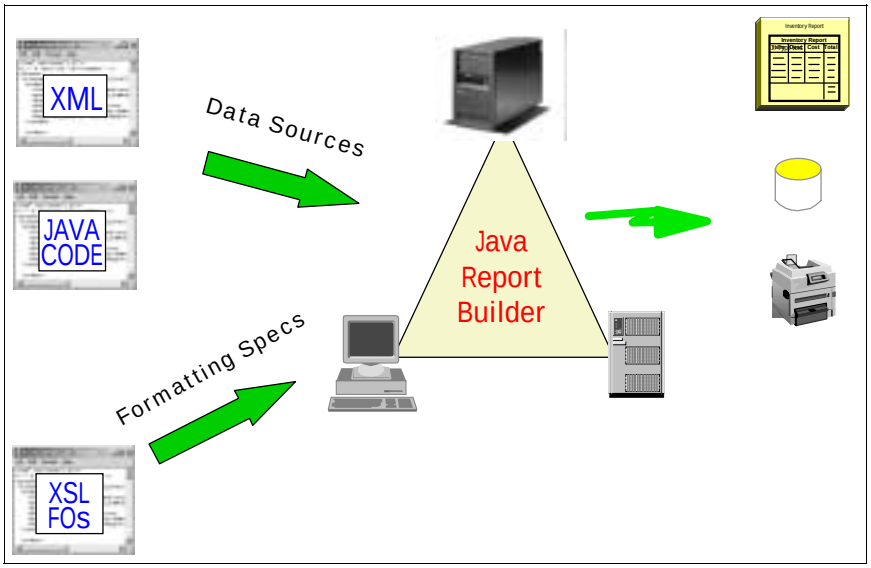

Figure 1. Java Report Builder

The report writer classes are contained in three different but related packages

- com.ibm.as400.util.reportwriter.pclwriter
- com.ibm.as400.util.reportwriter.pdfwriter

- com.ibm.as400.util.reportwriter.processor

The classes in the processor package are used by the application to specify the locations of the XML data, the XSL stylesheets, and the servlets/JSPs used within an application, and to initiate the actual formation of the output document. The XSLReportProcessor class in this package is used to process XML data with XSL stylesheets. The JSPReportProcess class is used to retrieve data from servlets and JSP pages and format the data.

The context classes (in the pclwriter and the pdfwriter packages) define methods that the ReportProcessor classes need to render XML and JSP data in the chosen format. The PCLContext class in combination with a ReportWriter class is used to generate a report in the Hewlett Packard Printer Control Language (PCL) format. The PDFContext class in combination with a ReportWriter class is used to generate a report in the Adobe Portable Document Format (PDF).

To create a formatted document from the application data, the report writer classes use the XSL formatting objects contained within the XSL stylesheet, the servlet, and the JSP. These formatting objects are defined in the second part of the XSL language, used for expressing stylesheets, as the XML vocabulary for specifying formatting semantics. A description of the formatting objects can be found in the Extensible Stylesheet Language (XSL) Version 1.0 specification at URL `http://www.w3.org/TR/xsl/`. An application programmer or document designer can create the XSL stylesheet or JSP, containing the XSL formatting objects, with any text editor, XSL stylesheet or JSP editor.

1.3 Creating SCS spooled files

There are a number of classes provided in the IBM Toolbox for Java that may be used to generate SCS spooled files. Each one supports slightly different print characteristics according to the actual device being written to, or emulated, ranging from SCS5256Writer, which is the simplest, to SCS3812Writer, which supports the widest range of function.

It is the programmer's responsibility to set all appropriate parameters for the data stream, including new lines and page eject codes.

For more information on using these classes, and for programming examples, see the Javadoc in the iSeries Information Center.

1.4 Creating a spooled file from stream data

If a fully composed data stream is available in a format that is available to a java program, such as in the IFS, the SpooledFileOutputStream class can be used to generate an iSeries spooled file.

Using this class would be synonymous with using the Print API QSPPUTSP that can be used from other high level languages.

It is not usually expected that a programmer would generate the data stream from scratch. However, if the print data stream was generated using some other method, this class can be used to create an iSeries spooled file.

1.5 Other related classes

There are a series of Print classes that allow the manipulation of iSeries print objects in the IBM Toolbox for Java. Print objects include spooled files, output queues, printers, printer files, writer jobs, and AFP resources. The AFP resources include fonts, form definitions, page definitions, overlays, and page segments.

The classes for print objects are organized in a base class (PrintObject) with subclasses for each of the six types of print objects. The Javadocs found on the iSeries Information Center provide details of the methods and attributes specific to each of these.

An example of using the AFPResource class to extract the contents of iSeries AFP resources can be found in the Extracting AFP Resource Contents Appendix.