XML PRINTER SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of priority from co-pending provisional patent application U.S. Serial No. 60/345,389, filed January 4, 2002, entitled XML Printer Technology, from co-pending provisional patent application U.S. Serial No. 60/341,427, filed December 17, 2001, entitled Bar Code Labeling Systems Having Machine Readable Standards, and from co-pending non-provisional application U.S. Serial No. 10/197,014, filed on July 17, 2002, entitled Native XML Printer. Provisional patent application Serial Nos. 60/345,389 and 60/341,427 are incorporated herein by reference in their entirety. This application is a continuation-in-part of application U.S. Serial No. 10/197,014, filed July 17, 2002, entitled Native XML Printer.

STATEMENT REGARDING COPYRIGHT RIGHTS

A portion of this disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office Patent files or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

The present invention relates generally to a method and apparatus for printing bar code labels and more specifically in one embodiment, to an XML (Extensible Mark-up Language) printer that prints bar code labels based on an XML data stream using existing ZPL-based (Zebra Programming Language) format template.

BACKGROUND

Printer systems for printing barcodes and for transmitting data to a barcode printer are known. However, many such systems use proprietary methods of data encoding, and therefore such methods cannot be used interchangeably with other barcode printers. Also, known data encoding methods typically render the underlying data unreadable by humans. While this presents no impediment to the computer systems, it may be burdensome to humans attempting

to review, debug or understand certain data appearing in the underlying barcode element names. In that regard, XML is an open standard that is being adopted by many business entities and is human-readable. Use of XML may avoid many of the problems and pitfalls associated with non-human readable methods.

Barcode labeling is used extensively in many facets of commerce. In particular, packages or merchandise shipped from one destination to another are identified by the shipper by a specific barcode label. Conversely, merchandise received may also be identified and entered into the receiver's inventory system by use of the barcode label. Often, the receiver of merchandise may dictate the form and content of the barcode applied by the shipper. This is referred to as "compliance labeling." Of course, merchandise need not be shipped to avail itself of the benefits of barcode labeling. For example, inventory control systems make extensive use of barcode labeling to track and monitor various goods within a facility or between facilities.

Compliance labeling is typically used by buyers of merchandise having relatively large market power or purchasing power. Because of their economic power, they may be able to dictate the form and content of the barcode labels applied to products provided to them by their suppliers or vendors. Although this may be burdensome to the supplier, if the supplier desires to do business with the buyer, they must comply with their demands with respect to labeling. For example, large retailers, such as Wal-Mart, Inc., not only have the ability and purchasing power to require that suppliers meet their compliance labeling demands, but may also fine suppliers who fail to comply with the labeling requirements.

Further, such barcode labeling requirements may change at the whim of the entity demanding compliance. Accordingly, the supplier must implement the new labeling requirements and test the modified barcode to insure that it meets all specifications. This is relatively inefficient and time consuming. It is also prone to errors, which may translate into monetary fines.

A need exists to permit an enterprise resource planning system (ERP) to format its data for transmission to a barcode printer system in XML, while additionally making use of existing ZPL format templates that govern the layout of the label and/or by making use of a pure XML format template that uses XSL (extensible stylesheet language) to govern the

layout of the label to be printed.

BRIEF DESCRIPTION OF THE DRAWINGS

The features of the present invention which are believed to be novel are set forth with particularity in the appended claims. The invention, together with further objects and advantages thereof, may best be understood by reference to the following description in conjunction with the accompanying drawings.

Fig. 1 is a high-level hardware block diagram of a specific embodiment of a printer system according to the present invention;

Fig. 2 is a high-level software block diagram of a specific embodiment of a printer system;

Fig. 3 is a combined high-level software block diagram and data flow diagram of a specific embodiment of a printer system;

Fig. 4 is a high-level software block diagram of a specific embodiment of a bitmap/barcode rendering engine;

Fig. 5 is a specific representation of a barcode label produced in accordance with the printer system of Figs. 1-4;

Fig. 6 is a specific example of an alternate embodiment of a printer system configured as a barcode rendering server;

Fig. 7 is a high-level software block diagram of a specific alternate embodiment of a printer system;

Fig. 8 is a combined high-level software block diagram and data flow diagram of a specific alternate embodiment of a printer system; and

Fig. 9 is a specific representation of a barcode label produced in accordance with the printer system of Figs. 7-8.

DETAILED DESCRIPTION

In this written description, the use of the disjunctive is intended to include the conjunctive. The use of definite or indefinite articles is not intended to indicate cardinality. In particular, a reference to "the" object or thing or "an" object or "a" thing is intended to also describe a plurality of such objects or things.

Referring now to Fig. 1, a specific embodiment of a high-level hardware block diagram

of an XML printer system 10 is shown generally. In the embodiment of Figs. 1-6, the native or natural "language" of the system is XML, thus the printer system 10 may be referred to as a "native XML printer." The XML printer system 10 includes a computer or processing system 12, which includes various hardware components, such as RAM 14, ROM 16, hard disk storage 18, cache memory 20, database storage 22, and the like (also referred to as "memory subsystem" 26), as is known in the art. The computer system 12 may include any suitable processing device 28, such as a computer, microprocessor, RISC processor (reduced instruction set computer), CISC processor (complex instruction set computer), mainframe computer, work station, single-chip computer, distributed processor, server, controller, micro-controller, discrete logic computer and the like, as is known in the art. For example, the processing device 28 may be an Intel Pentium® microprocessor, x86 compatible microprocessor, or equivalent device.

A user interface 30 may be coupled to the computer system 12 and may include various input devices 36, such as switches selectable by the user and/or a keyboard. The user interface also may include suitable output devices 40, such as an LCD display, a CRT, various LED indicators and/or a speech output device, as is known in the art.

To communicate between the computer system 12 and external sources, a communication interface 42 may be operatively coupled to the computer system. The communication interface 42 may be, for example, a local area network, as an Ethernet network, intranet, or other suitable network 43. The communication interface 42 may also be connected to a public switched telephone network (PSTN) 46 or POTS (plain old telephone system), which may facilitate communication via the Internet 44. Dedicated and remote networks may also be employed. Any suitable commercially available communication device or network may be used, as is known in the art.

The computer system 12 is further coupled to a printer system 50. The printer system 50 may include a media/paper control system 52, a printer driver 54 and a print head mechanism 56. Any suitable printer capable of printing barcode labels may be used, which may include various dot matrix, ink jet, laser and/or thermal printers. Of course, dot matrix printers are usually of lower quality and require closer monitoring of the label output. Preferably, the printer system 50 is a thermal transfer printer. Such suitable printers, for example, are

available from Zebra Technologies Corporation of Vernon Hills, Illinois, and may include the Model Xi series barcode printers (XiIII+, 90XiIII, 96XiIII, 140XiIII, 170XiIII, 220XiIII, etc.), the 2800 Series barcode printers, Model Z4M, Z6M, 105SL barcode printers, and others. Any suitable barcode label printer may be used.

Typically, such printers may include various motors, label cutters, ribbon handlers, sensors, and the like (not shown). Additionally, such printers may include various control inputs or sensors, such as a media sensor, print head temperature sensor, head open sensor, ribbon sensor, and the like (not shown), as is known in the art. The printer system 50 may include one or more additional processors 60, other than the processor 28 residing in the computer system 12. Alternatively, the processor 28 in the computer system 12, if sufficiently powerful, may control and handle the printer system 50 functions without the need for a separate processing device. Greater detail concerning the control of the print-head may be found in U.S. Patent No. 5,372,439 entitled "Thermal Transfer Printer With Controlled Ribbon Feed," issued December 13, 1994, and owned by the owner of the present invention. U.S. Patent No. 5,372,439 is hereby incorporated by reference in its entirety.

Preferably, the computer system 12 and the printer system 50 are located in a common enclosure, but need not necessarily be constructed in this manner. For example, the computer system 12 may be housed in an enclosure separate and apart from the printer system 50.

Referring now to Figs. 1-3, Fig. 2 illustrates a specific embodiment of a high-level software block diagram, while Fig. 3 illustrates a specific embodiment of a combined high-level software block diagram and data flow diagram. The software described below may be executed by the processor 28 of the computer system 12 of Fig. 1. Again, the processor 28 may perform functions common to both the computer system 12 and the printer system 50. There may be one or more processors, which may function in concert or which may function separately. It is not material to the scope of this invention whether the processing or processing functions are performed by or in the computer system or by or in the printer system.

The software blocks illustrated in Figs. 2-3 include an XML (extensible mark-up language) processor 70 (also referred to as the "XML parser"), an XSLT (extensible stylesheet language transformation) processor 74, an XSLFO (extensible stylesheet language

formatting object) processor 78, a barcode rendering engine 80 (also referred to as the "bitmap/barcode rendering engine"), and the printer driver 54 (Fig. 1). Note that the printer driver 54 is an example of a component described above whose function may be performed by either the processing device in the computer system 12 or the processing device 60 (Fig. 1) in the printer system 50, depending upon the physical location of the associated processing device. Again, a single processing device, if sufficiently powerful, may handle all functions for the XML printer system 10.

An XML schema repository 82 (schema repository) may provide input to the XML processor 70 while an XSLT stylesheet repository 84 (stylesheet repository) may provide input to the XSLT processor 74. Also shown is an enterprise resource planning (ERP) system 88, which may be, for example, a warehouse management system that transmits an XML input data stream 90 to the XML processor 70. The ERP system 88 essentially initiates the request to print the barcode label, and provides the XML data that forms the bar code and other variable label or element fields to be printed. Such variable label fields may include, for example, any or all of the human-readable text and/or characters printed on the label. Of course, any enterprise computer system may be used, and this invention is not limited to use with any specific type of enterprise computer system.

When referring to the XML data, two basic types shall be referred to herein, namely, the XML value data and the XML element name. The XML value data is the changeable data or the data that is desired to be printed on the barcode label, such as the data "1122 Green Street," which may be part of the XML value data corresponding to, for example, a shipping address. The XML element names are part of the XML language semantics where an arbitrary label or element name may be selected to represent the XML value data, the use of which is defined by the XML language. Typically, the element names appear between angled bracket ("<element name>").

As described above, known barcode label systems often use proprietary software encoding schemes. Additionally, such schemes are often prone to errors, and the underlying value data is usually unreadable by a non-technical individual. In known systems, if an error exists in the underlying value data sent from the enterprise system, or if the data is missing or otherwise incorrect, the barcode printer will print what it is instructed to do, which of course,

produces an error in the barcode label, rendering it inaccurate or useless.

Moreover, when dealing with compliance labeling, known systems require non-trivial changes in the data encoding when the form or content of the label changes in accordance with the compliance label demands. Such changes in the form or content of the barcode, again, are susceptible to errors, which in turn can lead to monetary fines by the entity demanding compliance. Business relationships may also be damaged by continued problems in the barcode labeling system, especially if such errors disrupt the business of the compliance demander.

The present XML printer system 10 utilizes an open format. In particular, the formatting requirements and the form of the barcode label are all defined in the XML language. Moreover, not only is XML well defined and available for all to use, but non-programmers can understand the data and commands in an XML data stream or file (or hard copy) with minimal training.

Various XML software blocks shown in Figs. 2-3 are commercially available. Several different commercially available XML processors 70 may be used interchangeably or with little modification. For example, the following commercially available XML processors 70 may be used: "XML for C++" available from IBM Corporation, "MSXML3" available from Microsoft Corporation, "Oracle XML Developers Kit for C" available from Oracle Corporation, "Expat" available from Thai Open Source Software Center, Ltd., or "Xerces-C++" available from the Apache Software Foundation. However, any suitable XML processor may be used.

Similarly, several different commercially available XSLT processors 74 may be used interchangeably or with little modification. For example, the following XSLT processors 74 may be used: "iXSLT" available from Infoteria Corporation, "MSXML3" available from Microsoft Corporation, and "Libxslt" available from Gnome. However, any suitable XSLT processor may be used.

Again, several different commercially available XSLFO processors 78 may be used interchangeably or with little modification. For example, the following XSLFO processors 78 may be used: "XEP" available from RenderX Corporation, "XSL Formatter" available from Antenna House Corporation, and "FOP" available from the Apache Software Foundation.

However, any suitable XSLFO processor may be used.

Still referring to Figs. 1-3, the XML processor 70 receives the XML input data stream 90 from an external source 88. For example, the external source may be the ERP system 88, such as the warehouse management system. The XML processor 70 is essentially parses and processes the XML input data stream 90 and generates a set of nodes, which may be in a "tree" structure, as is known in the art. Each of the software processing blocks shown in Figs. 2-3 act on the nodes of the "tree" to perform their required function. The underlying value data contained in the XML input data stream 90 from the ERP system 88 is processed and entered into a "label values node tree," 100 which holds the data.

The following is a brief overview of the operation of the various software components. First, note that the XML input data stream 90 includes text that identifies the name and location of other required XML documents or files. One such document is referred to as "XML schema" or "schema." The schema is used to validate the XML input data stream, including the underlying value data. If validation is successful, a stylesheet is applied, as will be described below. The name and location of the stylesheet is also specified in the XML input data stream 90. Application of the stylesheet is handled by the XSLT processor 74, which under the direction of the stylesheet, may transform the underlying XML element names and/or underlying value data. Next, the data is processed by the XSLFO processor 78, which handles formatting and "layout" of the underlying value data, which may include, for example, formatting the underlying value data in accordance with, for example, font type, font size, color, and the like. Next, the underlying value data is processed by the bitmap rendering engine 80, which creates a bitmap 92 of the barcode label corresponding to the transformed and formatted data. The bitmap rendering engine 80 may utilize an "instream foreign object" residing in the stylesheet to direct creation of the bitmap. The bitmap 92 is then sent to the printer driver 54 (Fig. 1) for subsequent printing of the barcode label by the barcode printer.

As described above, the schema functions to validate the entire input data stream 90, in particular, the underlying value data, where errors may be typically found. In practice, errors are often inadvertently introduced when changes are made to the form or content of the bar code label.

The name and location of the schema document is contained in the XML input data

stream 90, which XML input data stream corresponds to the request to print a barcode label. The XML processor 70 in conjunction with a schema validation module 110 validates the underlying value data. The use of schema is cost effective because it prevents errors and omissions with respect to the final output, namely, the bar code label or "shipping label."

If the XML input data stream 90 is rejected or flagged as having an error, an error message may be transmitted back to the source 88. This may flag or trigger human intervention to correct the error. For example, in this specific example, the source is the ERP system 88. In this way, the data is initially checked prior to processing to insure that it complies with all required label and barcode rules.

This may be particularly beneficial when dealing with compliance labeling. In known systems, the compliance demander would merely notify the supplier as to the changes in the compliance labeling requirements. If the supplier then makes an error in interpreting or implementing these changes or instructions, the labels produced and applied to products shipped to the compliance demander may have errors, which could jeopardize future business or cause monetary fines to be applied.

In the present invention, the compliance demander preferably makes the changes directly to the schema and/or the XSLT stylesheet. For example, if the physical layout of the label has been changed or if element names have been changed, the compliance demander will modify the XSLT style sheet. Similarly, if the underlying value data has been added or deleted or otherwise qualified (i.e., a new acceptable numerical range for a zip code), the compliance demander may modify the schema. In this way, the supplier need only modify the output of its ERP system 88 to ensure that it matches the modified XML input data stream 90. If only the physical layout of the label has changed, the supplier does not need to make any modifications at all.

For example, the compliance demander may now require that a nine digit zip code be used rather than the original five digit zip code. Accordingly, the compliance demander will modify the schema to require both a first and second zip code field, and the second field will also be limited to numerical digits within a certain range, perhaps 0000-9999. The compliance demander may also modify the stylesheet to accommodate that change. In response thereto, the supplier must insert the added zip code field in its ERP system so that it appears in the

XML input data stream 90 sent to the XML printer system 10. If such modification of the XML input data stream 90 is not performed correctly, the schema will cause an error to be reported back to the ERP system 88, and the label will not be printed.

Thus, the supplier need only access the modified schema and/or stylesheet from the repository 82, 84, which is automatically applied to the underlying value data when received. Essentially, minor changes, and significantly, major changes, to the form and content of the barcode label are transparent to the supplier, and such changes to the content of the barcode label are validated in accordance with the schema. Accordingly, the supplier need not incur costs to change the form or content of the barcode label dictated by the compliance demander, and cannot make any errors in implementing such changes. If there are any errors, such errors would have been inadvertently made by the compliance demander, who could not then blame the supplier.

The schema documents are preferably obtained from the XML schema repository 82. In one specific embodiment, the schema repository 82 may be external to the XML printer system 10 and the computer system 12, and may be accessed via the network, the Internet, or via any suitable network 43, 44 to which the computer system is coupled. The schema repository 82 may contain a plurality of schema documents. Thus, the XML input data streams 90 representing the various requests to create a barcode label may each specify the name and location of the corresponding schema in the repository 82. When the request is received by the XML processor 70, the corresponding schema may be retrieved from the schema repository 82.

In another embodiment, the schema obtained from the schema repository 82 via the network 42, 43 may be kept locally, and thus may temporarily reside in the memory subsystem 26 (Fig. 1), such as the hard disk 18 or database 22. In this way, if the same schema is used for multiple XML input data streams 90 or for subsequent barcode label requests, the XML processor 70 need not retrieve the same schema externally via the network 42, 44, but rather, may retrieve that schema from the memory subsystem 26, which may be more efficient. According to this embodiment, the compliance demander may change or modify the schema in the external repository 82 at only certain times. For example, the compliance demander may change the stylesheet only at 1:00 AM each day. Thus, the supplier need only update the

schema from the repository 82 into the memory subsystem 26 only once per day, for example, after the compliance demander has performed the schema update. The supplier would then know that the schema saved temporarily in the memory subsystem 26 is the most recent schema document, at least up until the time that the updating is scheduled to occur.

Regardless of the location from where the schema is obtained, the schema validation module 110 performs the checking and validation of the underlying data. Although the schema validation module 110 is shown as a separate block from the XML processor 70 in Fig. 2, it is shown in this location for purposes of illustration only so that it may be shown on the drawing adjacent to the label values node tree 100, which is the data upon which it acts. However, the schema validation module 110 may be part of and integrated into the XML processor 70, or it may be a separate and apart therefrom.

Of course, the schema is also an XML document, and thus it is also processed by the XML processor 70. Accordingly, the result of the processing of the schema is the XML schema node tree 114 shown in Fig. 3, which is the "memory representation" or working model of the schema that was processed. The XML schema node tree 114 may be in the form of a "document object model" (DOM), as is known in the art. Further, the XML schema node tree 114 may reside in cache memory for efficiency, as shown in an XML schema cache 116. The schema validation module 110 and/or the XML processor 70 operate on the data in the XML schema node tree 114 to perform its function of validating the underlying value data in accordance with the schema document.

As described above, if an error exists in the XML input data stream 90, as determined by application of the schema, an error message may be generated. If the XML input data stream 90 is validated, the data remains essentially "untouched." The data in the label value node tree 100 is then processed by the XSLT processor 74 using the XSLT stylesheets.

The stylesheet documents are preferably obtained from the XSLT stylesheet repository 84. In one specific embodiment, the stylesheet repository 84 may be external to the XML printer system 10 and the computer system 12, and may be accessed via the network, the Internet, or via any suitable network 43, 44 to which the computer system is coupled. The stylesheet repository 84 may contain a plurality of stylesheets. Thus, XML input data streams 90 representing the various requests to create a barcode label may each specify the name and

location of the corresponding stylesheet in the repository 84. When the request is received by the XML processor 70, the corresponding stylesheet may be retrieved from the stylesheet repository 84.

In another embodiment, the stylesheet obtained from the stylesheet repository 84 via the network 43, 44 may be kept locally, and thus may temporarily reside in the memory subsystem 26 (Fig. 1), such as the hard disk 18 or database 22. In this way, if the same stylesheet is used for multiple XML input data streams 90 or for subsequent barcode label requests, the XML processor 70 need not retrieve the same stylesheet externally via the network 43, 44, but rather, may retrieve that stylesheet from the memory subsystem 26, which may be more efficient.

According to this embodiment, the compliance demander may change or modify the stylesheet in the external stylesheet repository 84 at only certain times. For example, the compliance demander may change the stylesheet only at 1:00 AM each day. Thus, the supplier need only update the stylesheet from the stylesheet repository 84 into the memory subsystem 26 only once per day, for example, after the compliance demander has performed the stylesheet update. The supplier would then know that the stylesheet saved temporarily in the memory subsystem 26 is the most recent stylesheet, at least up until the time that the updating is scheduled to occur.

Of course, the stylesheet is also an XML document, and thus it is also processed by the XML processor 70. Accordingly, the result of the processing of the stylesheet is the XSLT stylesheet node tree 120 shown in Fig. 3, which is the "memory representation" or working model of the stylesheet that was processed. The XSLT stylesheet node tree 120 may be in the form of a "document object model" (DOM), as is known in the art. Further, the XSLT stylesheet node tree 120 may reside in cache memory for efficiency, as shown in an XSLT stylesheet cache 126. The XSLT processor 74 operates on the data in the XSLT stylesheet node tree 120 to perform its function of transforming the underlying value data or underlying element names in accordance with the stylesheet.

Note that although the XSLT style sheet is shown as an input to the XML processor 70 in Fig. 3, the XSLT processor 74 processes the stylesheet. It is initially provided to the XML processor 70 because all XML documents are first processed and placed into the

appropriate data structure for subsequent processing.

The XSLT processor 74 may modify, reposition, and rearrange the underlying value data or may add to the underlying value data or delete some of the underlying value data. For example, under direction of the stylesheet, the underlying value data may be rearranged into table format or into columns. In particular, the stylesheet may add XSLFO formatting elements and attributes.

After the underlying value data in the label value node tree 100 has been processed in accordance with the corresponding stylesheet, an XSLFO instance node tree 130 is produced. Again, the XSLFO instance node tree 130 may be in the form of a document object module, as is known in the art. The XSLFO instance node tree 130 contains XSLFO commands (layout instructions) that directs the XSLFO processor 78 with respect to formatting and layout. The XSLFO processor 78 then interprets the XSLFO commands and applies such commands to the underlying value data so as to properly format and layout the underlying value data. The XSLFO processor 78 produces the XSLFO area node tree 130, which represents the final output of formatting before rendering.

Turning now to Fig. 3 and a code segment shown immediately below entitled "code segment 1 for an XML input data stream," the code segment 1 illustrates an XML input data stream 90 in hard copy, which may, for example, be sent to the XML printer system 10 by the ERP or warehouse management system 88. Line numbering has been inserted for purposes of illustration only and is not part of the code.

Code Segment 1 For An XML Input Data Steam

```
1 <?xml version="1.0" encoding="UTF-8"?>
  <?xml-stylesheet type="text/xsl"
      href="D:\Projects\XML\Native\Docs\ShipLabels.xsl"?>
<labels
5     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation=
            "D:\Projects\XML\Native\Docs\ShipLabels.xsd">
      <label>
            <name>Albert Einstein</name>
10          <address>1234 Relative Way</address>
            <city>Princeton</city>
            <state>NJ</state>
            <zip>08540</zip>
      </label>
15    <label>
            <name>Steven Hawking</name>
            <address>5678 Black Hole Drive</address>
            <city>Los Angeles</city>
```

```
            <state>CA</state>
            <zip>90007</zip>
      </label>
      <label>
            <name>Richard Feynman</name>
            <address>90 Quantum Circle</address>
            <city>New York</city>
            <state>NY</state>
            <zip>10044</zip>
      </label>
</labels>
```

The XML input data stream identifies the schema document as "ShipLabels.xsd," and that schema document may be found, in this specific example, in a directory called "D:/Projects/XML/Native/Docs," as shown at line 7 in code segment 1. Further, the XML input data stream identifies the stylesheet document as "ShipLabels.xsl," and that stylesheet document may also be found in a directory called "D:/Projects/XML/Native/Docs," as shown at line 3 of the code segment 1. Of course, the schema document and the stylesheet document may be located anywhere, for example, as identified by an Internet address.

This specific example shows the underlying value data and element names for three shipping labels to be printed. Each shipping label contains an XML element name defined between angular brackets as follows: <name>, <address>, <city>, <state> and <zip>. The value of the first element name, <name>, is "Albert Einstein," the value of the second element name, <address>, is "1234 Relative Way," the value of the third element name, <city>, is "Princeton," the value of the forth element name, <state>, is "NJ" and the value of the fifth element name, <zip>, is "08540." This is the underlying value data.

Now turning to Fig. 3, code segment 1, and a code segment shown immediately below entitled "code segment 2 for XML schema," the code segment 2 illustrates a specific example of an XML document in the form of the XML schema document specified in the XML input data stream of code segment 1. Line numbering has been inserted for purposes of illustration only and is not part of the code.

Code Segment 2 For XML Schema

```
1 <?xml version="1.0" encoding="UTF-8"?>
  <xs:schema
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">
5     <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="label">
            <xs:complexType>
```

```
                <xs:sequence>
                      <xs:element ref="name"/>
                      <xs:element ref="address"/>
                      <xs:element ref="city"/>
                      <xs:element ref="state"/>
                      <xs:element ref="zip"/>
                </xs:sequence>
          </xs:complexType>
    </xs:element>
    <xs:element name="labels">
          <xs:complexType>
                <xs:sequence>
                      <xs:element ref="label" maxOccurs="unbounded"/>
                </xs:sequence>
          </xs:complexType>
    </xs:element>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="state">
          <xs:simpleType>
                <xs:restriction base="xs:string">
                      <xs:enumeration value="CA"/>
                      <xs:enumeration value="NJ"/>
                      <xs:enumeration value="NY"/>
                </xs:restriction>
          </xs:simpleType>
    </xs:element>
    <xs:element name="zip">
          <xs:simpleType>
                <xs:restriction base="xs:int">
                      <xs:minInclusive value="00000"/>
                      <xs:maxInclusive value="99999"/>
                </xs:restriction>
          </xs:simpleType>
    </xs:element>
</xs:schema>
```

As mentioned above, the schema is used to validate the underlying value data. This entails checking to determine that all required data is present, that no extraneous data is present, that the data present is within the specified ranges, and the like. Any suitable validation scheme may be specified in the schema, depending upon the application. The XML language is robust and code may be written to handle a vast multitude of requirements.

For example, the schema document shown in code segment 2 above specifies that the underlying value data corresponding to the element name, <address>, must be a string, as defined in the XML Schema specification, as shown by line 5 in the code segment 2. The schema document also specifies that the underlying value data corresponding to the element names of <name>, <address>, <city>, <state>, and <zip> must also be present in the sequence indicated, as shown by lines 9-15 in the code segment 2. Further, this specific schema document shown in the code segment 2 specifies that the underlying value data corresponding

to the element name, <state>, must be one of three states, namely, "CA," "NJ," or "NY." Of course, this is only an abbreviated example, and not all states have been included for purposes of illustration only. The schema document shown in code segment 2 also specifies that the underlying value data corresponding to the element name, <zip>, must be in the range from 00000 to 99999. If any of the above-mentioned schema criteria are not met by the data in the XML input data stream, the schema validation module 110 will reject it, and will preferably return an error message back to the source 88.

Now turning to Fig. 3, code segments 1-2, and a code segment shown immediately below entitled "code segment 3 for an XSLT stylesheet," the code segment 3 shows a specific example of an XML document in the form of the XSLT stylesheet document specified in the XML input data stream of the code segment 1. Line numbering has been inserted for purposes of illustration only and is not part of the code.

Code Segment 3 For An XSLT Stylesheet

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:transform version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:bo="http://www.zebra.com/2002/XSL/Barcode">
<xsl:output method="xml" version="1.0" indent="yes" />
  <xsl:template match="labels">
    <fo:root>
      <fo:layout-master-set>
          <fo:simple-page-master master-name="all-labels">
              <fo:region-body margin="1in" />
        </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-name="all-labels">
        <fo:flow flow-name="xsl-region-body" font="12pt
        Times">
          <xsl:apply-templates />
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
  <xsl:template match="label">
    <fo:block break-after="page">
      <xsl:apply-templates select="name" />
      <xsl:apply-templates select="address" />
      <fo:block font="bold 14pt Times" text-align="left">
        <xsl:apply-templates select="city" />
        <xsl:apply-templates select="state" />
      </fo:block>
      <xsl:apply-templates select="zip" />
```

```
      <fo:block>
        <fo:instream-foreign-object>
          <bo:barcode>
            <bo:postnet interpretation-line="none">
              <xsl:value-of select="zip" />
            </bo:postnet>
          </bo:barcode>
        </fo:instream-foreign-object>
      </fo:block>
    </fo:block>
  </xsl:template>
  <xsl:template match="name">
    <fo:block font="bold 14pt Times" text-align="left">
      <xsl:value-of select="." />
    </fo:block>
  </xsl:template>
  <xsl:template match="address">
    <fo:block font="bold 14pt Times" text-align="left">
      <xsl:value-of select="." />
    </fo:block>
  </xsl:template>
  <xsl:template match="city">
    <xsl:value-of select="." />
  </xsl:template>
  <xsl:template match="state">
    <xsl:text>,</xsl:text>
    <xsl:value-of select="." />
  </xsl:template>
  <xsl:template match="zip">
    <fo:block font="bold 14pt Times" text-align="left">
      <xsl:value-of select="." />
    </fo:block>
  </xsl:template>
</xsl:transform>
```

As mentioned above, the stylesheet is used to transform the data. In this specific example, the stylesheet of the code segment 3 defines two types of namespace, namely, a "bo" type object, which is selected to be a "barcode-type object," and an "fo" type object, which is selected to be a formatting object, which was described above with respect to the XSLFO processor 78. This permits differentiation between the different objects, as is known in the art. The stylesheet may cause some data to be transformed or even added. For example, the underlying value data from the XML input data stream of the code segment 1, namely "Einstein, "1234 Relative Way," and the like, is inserted, as indicated by lines 40-62.

This stylesheet also causes a variety of "fo" type elements to be generated, which are designated as formatting elements or XSLFO objects. The XSLFO objects contain layout type commands that the XSLFO processor 78 "understands" and processes. Note that all lines of

code having a prefix of "fo" are not interpreted by the XSLT processor 74, but rather, are passed on to the XSLFO processor 78.

For example, line 21 of the code segment 3 shows <xsl: template match= "label"> followed by <fo:block break-after = "page"> on line 22. This is interpreted by the XSLFO processor 78 to put a "page break" between the three shipping labels specified in the code segment 1. Essentially a new shipping label is created due to the page break, with each new shipping label containing the underlying value data specified in the code segment 1.

Further, this specific stylesheet specifies that an "instream foreign object" is inserted, as shown at line 31. Line 31 is shown as <fo:instream-foreign-object> in conjunction with lines 31-37, which define the instream foreign object. Note that the instream-foreign object is not processed by either the XSLT processor 74 or the XSLFO processor 78. Rather, the instream foreign object is passed to and is processed by the bitmap/barcode rendering engine 80. For example, lines 32-36 of the code segment 3 cause a <bo:barcode> element to be inserted that contains the definition of a "postnet" barcode element, which element will later be processed by the bitmap/barcode rendering engine 80. At line 34, the XSLT processor 74 causes the zip code from the XML input data stream to be inserted as the text of the <bo:postnet> element.

Now turning to Fig. 3, code segments 1-3, and a code segment shown immediately below entitled "code segment 4 for an XSLT instance node tree representation," the code segment 4 shows a specific example of a representation of an XSLFO instance node tree 130 output produced by the XSLT processor 74 in conjunction with the stylesheet of code segment 3, which output is a representation of the XSLFO instance node tree provided to the XSLFO processor 78. Line numbering has been inserted for purposes of illustration only and is not part of the code.

Code Segment 4 For An XSLFO Instance Node Tree Representation

```
1 <?xml version="1.0" encoding="UTF-16" ?>
   <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:bo="http://www.zebra.com/2002/XSL/Barcode">
     <fo:layout-master-set>
5     <fo:simple-page-master master-name="all-labels">
        <fo:region-body margin="1in" />
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-name="all-labels">
```

```
<fo:flow flow-name="xsl-region-body" font="12pt Times">
  <fo:block break-after="page">
    <fo:block font="bold 14pt Times" text-
      align="left">Albert Einstein</fo:block>
    <fo:block font="bold 14pt Times" text-
      align="left">1234
          Relative Way</fo:block>
    <fo:block font="bold 14pt Times" text-
      align="left">Princeton, NJ</fo:block>
    <fo:block font="bold 14pt Times" text-
      align="left">08540</fo:block>
    <fo:block>
      <fo:instream-foreign-object>
        <bo:barcode>
          <bo:postnet interpretation-
            line="none">08540</bo:postnet>
        </bo:barcode>
      </fo:instream-foreign-object>
    </fo:block>
  </fo:block>
  <fo:block break-after="page">
    <fo:block font="bold 14pt Times" text-
    align="left">Steven Hawking</fo:block>
    <fo:block font="bold 14pt Times" text-
      align="left">5678 Black Hole Drive</fo:block>
    <fo:block font="bold 14pt Times" text-
      align="left">Los
          Angeles, CA</fo:block>
    <fo:block font="bold 14pt Times" text-
      align="left">90007</fo:block>
    fo:block>
      <fo:instream-foreign-object>
        <bo:barcode>
          <bo:postnet interpretation-
            line="none">90007</bo:postnet>
        </bo:barcode>
      </fo:instream-foreign-object>
    </fo:block>
  </fo:block>
  <fo:block break-after="page">
    <fo:block font="bold 14pt Times" text-
      align="left">Richard Feynman</fo:block>
    <fo:block font="bold 14pt Times" text-
      align="left">90 Quantum Circle</fo:block>
    <fo:block font="bold 14pt Times" text-
      align="left">New York, NY</fo:block>
    <fo:block font="bold 14pt Times" text-
          align="left">10044</fo:block>
    <fo:block>
      <fo:instream-foreign-object>
        <bo:barcode>
          <bo:postnet interpretation-
                line="none">10044</bo:postnet>
        </bo:barcode>
```

```
            </fo:instream-foreign-object>
          </fo:block>
        </fo:block>
65    </fo:flow>
    </fo:page-sequence>
  </fo:root>
```

Two main portions of code segment 4 above are briefly described, namely, one portion beginning at line 4 with the code <fo:layout-master-set> and a second portion beginning at line 9 with the code <fo:page-sequence master-name="all-labels>. This portion of the code essentially defines what a barcode label will "look like" or how it will be "laid-out" and how the underlying value data will appear in that lay-out.

With respect to formatting the underlying value data shown in code segment 1, for example, the font attribute of "bold 14pt Times" and text-align = "left" shown in lines 12-13 will be applied to the underlying value data of "Albert Einstein" so that when the barcode label is printed, the name of "Albert Einstein" will be printed in 14 point Times font, and it will aligned with the left margin of the barcode label. As is known in the art, may different kinds of formatting instructions may be specified, which will be interpreted by the XSLFO processor 78.

Further, this specific example of the representation of the XSLFO instance node tree 130 contains an instream foreign object, as shown at lines 21-26 of the code segment 4, which instream foreign object may not be processed by the XSLFO processor 78. Rather, the insteam foreign object may be passed to the bitmap/barcode rendering engine 80 for processing. Note that at this point in the processing, the XSLFO processor 78 does not "know" how to print a barcode label because barcode element types are not included in the XSLFO specification. Rather, only the text information, such as the underlying value data, namely, "Einstein," "1234 Relative Way," and the like, as shown in code segments 1-2, could be printed on the barcode label as text. No actual barcode symbol would appear because the XSLFO processor 78 does not "know" how to produce a barcode symbol.

The result of processing by the XSLFO processor 78 is an XSLFO area node tree 140. Note that the instream foreign object shown in the code segment 4 is not processed by the XSLFO processor 78, but rather, is passed to the barcode/bitmap rendering engine 80 in one or more nodes of the XSLFO area node tree 140. The instream foreign object element is

referred to as <bo:barcode> and is shown at lines 21-26 of the code segment 4. The barcode/bitmap rendering engine 80 receives the <bo:barcode> element and processes the code contained in that block. For example, the rendering engine 80 interprets lines 23-24, namely, <bo:postnet interpretation-line="none">08540<bo:postnet> as defining this particular node as being a "postnet" barcode type of element. Many different "types" of barcode symbols may be defined, as is defined by corresponding industry standards, discussed below.

The underlying value data to be printed as a barcode symbol is the numerical data "08540," as shown in line 24 of code segment 4, which underlying value data is the zip code associated with Albert Einstein's address shown in the code segment 1. In this specific example, only the zip code is transformed into the barcode symbol. Of course, any and all information may be transformed into a barcode symbol, depending upon the application. Also, in this specific example, the instream foreign objects shown in the code segment 4 only relate to the three zip codes shown in the code segment 1. No other underlying value data is transformed into the barcode symbol in this example.

Turning now to Figs. 2-4 in conjunction with the code segment 4, additional detail concerning the barcode/bitmap rendering engine 80 and the processing of the instream foreign objects will now be set forth. As described above, the rendering engine 80 processes each node of the XSLFO area node tree 140 generated by the XSLFO processor 78. As is known in the art, the XSLFO area node tree 140 is represented in memory as a document object model. Accordingly, multiple nodes may exist. For example, one node may contain the instream foreign object, another node may contain associated element names, another node may contain associated text data, another node may contain associated attributes, and the like.

As shown in block 150 in Fig. 4, the nodes of the XSLFO area node tree 140 are processed. In particular, the rendering engine 80 "knows" how to render or produce the actual barcode symbol defined by the instream foreign object. Commercially available XSLFO processors do not "know" how to process such instream foreign objects representing barcode symbols, and thus are not able to produce or render barcode symbols. Commercially available XSLFO processors may, however, render text in an acceptable manner and may even render certain types of instream foreign objects.

If a particular node does not contain an instream foreign object, meaning that it may

contain, for example, text data, then this node may be processed in a "standard" way by utilizing the processing power of the XSLFO processor 78 rather than the rendering engine 80, as illustrated in a block 154. In one embodiment, the non-instream foreign object is provided "back" to the XSLFO processor 78, or the XSLFO processor is notified that a particular node in the XSLFO area node tree 140 is suitable for processing by it.

Processing of a node in the XSLFO area node tree 140 containing a non-instream foreign object may be processed by the XSLFO processor 78, as described above. Of course, different commercially available XSLFO processors render text, for example, into different formats. For example, the XSLFO processor referred to as "XSL Formatter" available from Antenna House Corporation, as described above, renders a node into a WINDOWS "device-independent bitmap" format. The XSLFO processor referred to as "XEP" available from RenderX Corporation, as described above, renders a node into "PDF" format, and the XSLFO processor referred to as "FOP" available from the Apache Software Foundation, as described above, may also render a node into "PDF" format.

Preferably, the XML printer system 10 utilizes the "FOP" XSLFO processor 78 available from the Apache Software Foundation because this XSLFO processor may be configured to "call" a subroutine or other processing function when it encounters an instream foreign object that it is not capable of processing. In particular, this XSLFO processor 78 may be set to call the rendering engine software block 80 described herein, to handle the instream foreign object that it is not capable of processing.

Next, as shown in a block 160, if the node contains an instream foreign object representing a barcode, which is not capable of being processed by the XSLFO processor 78, the rendering engine 80 checks to determine if the instream foreign object has a "namespace" specific to this XML printer system 10. For example, the rendering engine 80 may check to determine if the namespace associated with the instream foreign object corresponds to the "Zebra corporation namespace." The namespace notation is known in the art. This is shown in line 3 of the code segment 4 as "xmlns:bo="http://www.zebra.com/2002/XSL/Barcode">" Thus, as shown in the code segment 4, the namespace defined above is applied to any elements having a "bo" prefix. Such elements having a "bo" prefix are then only available for processing by the rendering engine 80, which "bo" prefix type elements correspond to a

barcode-type instream foreign object. The XSLFO processor 78 will not "know" how to process such elements.

If the instream foreign object is not associated with the proper namespace, it is passed back to the XSLFO processor 78, assuming that it is not a barcode type instream foreign object because the XSLFO processor may be able to process some limited number of instream foreign objects. This is shown by arrow 162 from the block 160 to the block 154. Generally, however, instream foreign objects exist or are created for the specific reason that the data associated with an instream foreign object is very unique in nature (such as a barcode), and thus, many commercially available or general purpose XSLFO processors can not process them.

Note that the commercially available XSLFO processors mentioned herein are not necessarily incapable of processing all instream foreign objects. They are, however, incapable of processing instream foreign objects corresponding to barcode symbols.

Assuming that instream foreign object has the appropriate namespace, a block 170 interprets the elements in the instream foreign object to determine barcode type. For example, the instream foreign object is shown at lines 21-26 the code segment 4. More specifically, the barcode element having a prefix of "bo," indicating the correct namespace corresponding to a barcode, is shown to have a barcode type of "postnet," as shown in line 23-24 of the code segment 4, which appears as <bo:postnet interpretation-line="none">08540</bo:postnet> . Note that lines 41-42 of the code segment 4 show a similar barcode element for another instream foreign object, namely, the next barcode label to be printed, which corresponds to the address of "Steven Hawking" shown in code segment 1.

Any barcode type may be specified in the instream foreign object depending upon the application. The barcode type is based upon known barcode standards. For example, the barcode type is not limited to a postnet type of barcode symbol, and may be, for example, Industrial 2 of 5," "PDF 417," "UPC," and the like. The barcode "type" specifies the basic structure of the barcode symbol to be printed, such as the thickness of the bars, the ratio of the thickness of the bars to the width of the space between bars, and the like. Each specific known barcode standard is concretely defined.

Next, after the barcode type has been ascertained from the instream foreign object, the attributes of the instream foreign object are interpreted to determine the barcode characteristics, as shown in a block 176. The characteristics may modify the way in which the barcode symbol is created. For example, one attribute of the "postnet" type barcode may be seen at lines 23-24 of the code segment 4 as 'interpretation-line="none" ' As is known in the art, 'interpretation-line="none" ' means that no corresponding human readable text is generated along with the specific barcode symbol. Another example of an attribute associated with the barcode element may be the height of the barcode symbol, which may also be specified in the instream foreign object as an attribute.

Using the barcode type and the attributes obtained from the instream foreign object, the rendering engine 80 then obtains the actual node text data to be converted into the barcode symbol, as shown in a block 180. In this example, the actual barcode data is the zip code "08540," as is shown between reverse angled brackets (>80540<) in line 24 of the code segment 4. The physical conversion from the numerical data "80540" into the bitmap representing the lines or bars of the barcode symbol (for a linear barcode, for example) may be performed by software routines contained in, for example, the barcode label printers commercially available from Zebra Corporation. Such suitable barcode printers may include, for example Models Xi series barcode printers (XiIII+, 90XiIII, 96XiIII, 140XiIII, 170XiIII, 220XiIII, etc.), the 2800 Series barcode printers, Model Z4M, Z6M, 105SL barcode printers, and the like. Such known software routines or algorithms contained in the barcode printers commercially available from Zebra Corporation are very efficient, and thus the barcodes are rendered quickly.

For linear barcodes, the bitmap is rendered by converting the text data to a series of dark and light bars whose relative widths conform to the industry specifications for the barcode symbology. The text data may have check characters added before encoding to allow verification of the barcode when it is read by a barcode scanner. The check characters may be required or may be optional depending upon the symbology. The bars may be grouped into modules of bars depending upon the requirements of the encoding of the symbology. The text data may be compressed during encoding depending upon the symbology. The bars may be preceded and/or followed by one or more "finder" bars as

defined by the symbology specifications, which allow a scanner to locate the barcode data more easily.

Other algorithms may be applied as needed for the proper encoding and proper physical dimensions required by the symbology. The height of the bars, the width of the narrow bar, the ratio of the wide bar to narrow bar, and other symbology specific parameters may be specified by attributes in the XML data. In addition, the barcode bitmap may be rotated (usually in ninety degree increments) as specified by attributes in the XML data.

For two-dimensional barcodes, the bitmap is rendered by converting the text data to a bit pattern according to the various unique symbology algorithms. Two-dimensional barcodes typically allow for encoding of more text data than linear barcodes. The encoding algorithm may include, for example, data compression, error checking and correction, varying levels of data redundancy, modularization, finder patterns, and other algorithms as defined by the symbology. The user-specifiable symbology specific parameters may be specified by attributes in the XML data. The bitmap generated by the symbology algorithm may then be magnified and/or rotated as specified by attributes in the XML data.

Once the bitmap representing the barcode symbol has been generated by the block 180, that bitmap is then placed into or merged with the "master" or "label" bitmap 184, which label bitmap represents the image of the entire shipping label, as shown in a block 182. The shipping label contains the human readable text, other text or characters, and the barcode symbol. Note that the aforementioned bitmaps may be separate, i.e., bitmaps for text and a bitmap for the barcode symbol, or one master bitmap may be used or "populated" as processing progresses. In that regard, note that the tree or data structure upon which the rendering engine 80 operates is referred to as the "XSLFO area node tree," meaning that it contains different "areas." Such areas may correspond to different or individual bitmaps or bitmap portions. However, any number of suitable bitmap structures may be used. The exact structure or configuration of the bitmap, or the number of bitmaps is not relevant to the scope of the present invention.

For example, the XSLFO processor 78 when generating text, may place "its" text in one particular bitmap. Such text may correspond to the name "Albert Einstein" discussed above. Because this text will also be printed on the shipping label along with the barcode

symbol, the "master" or "label" bitmap 184 will contain all of the component bitmaps. The final "master" bitmap 184 is then sent to the printer driver 54 (Fig. 1) so that the shipping label may be printed.

Turning now to Figs. 4-5, Fig. 5 is a representation 190 of the final output of the above-described processing for the first barcode shipping label specified in code segment 1. The text data is rendered and formatted in accordance with the corresponding stylesheet, while the zip code 192 is also rendered in the form of a barcode symbol by the rendering engine 80, as described above. Note that for purposes of illustration only, the font size or font type shown in Fig. 5 may not be the same as that specified in the corresponding stylesheet shown in the code segment 3.

Referring now to Fig. 6, an alternate embodiment is shown. Like reference numbers are used to denote like structures. In this specific embodiment, the XML printer system 10 need not include or specifically utilize the XML printer apparatus in all instances. In this embodiment, for example, an external requestor 194 may desire to have an XML input data stream rendered into a bitmap so that it can print the barcode label, which may be done remotely. Accordingly, the external requestor 194 may send a request to the XML server 200 over the Internet or other network 43, 44. The request may be in the same form as the XML input data stream described above with respect to Figs. 1-3 and code segments 1-4. The request may also be in the form of a protocol that contains XML data, such as SOAP (Simple Object Access Protocol) as is known in the art.

In response, the computer system 12 of the XML server 200 may process the XML input data stream. All of the same processing functions described above are carried out in this embodiment, except that the bitmap is not sent to the printer driver. Rather, the bitmap is converted to any suitable format, such as "JPEG" or "PNG" format, as is known in the art. The JPEG or PNG file is then sent back to the external requestor 194, who then formats the JPEG or PNG file for the appropriate printer driver, and prints and/or displays the barcode label. Accordingly, the present invention may function as a barcode rendering server and need not necessarily perform the printing function.

An alternate embodiment of the printer system of Fig. 1 is now shown in Figs. 7-8, which show a "hybrid" XML printer system 300, rather than the "native" XML printer system

10 shown in Figs. 2-6. Note that this embodiment may still be used with the computer system 12 and printer system 50 shown in Fig. 1, and like reference numerals will be used to denote like structure or components.

In this embodiment, the hybrid XML printer system 300 may receive XML data that is based on ZPL data templates, which templates conform to the ZPL language (Zebra Programming Language), such as ZPL I and/or ZPL II, and any variants thereof. The ZPL language is a formatting language that uses the "caret" character and a two letter command code to define a command. For example, the "^XA" command defines the start of a new label format, and the "^XZ" command defines the end of a label format.

The ZPL language also includes "tilde" commands, which represent control commands. For example, the command "~JR" is a printer reset command, which is acted upon immediately by the printer, and does not really have significant bearing on the actual formatting of a barcode label. ZPL I and ZPL II were developed by Zebra Corporation of Vernon Hills, Illinois, the owner of the present invention. Note that ZPL is used interchangeably with ZPL II herein.

Greater detail regarding the ZPL language specification can be found in documents entitled "ZPL II Programming Guide Volume One: Command Reference for X.10," Manufacturer Part No. 45541LB, and "ZPL II Programming Guide Volume Two: The X.10 Environment," Manufacturer Part No. 45542LB, which documents may be found and downloaded at the Zebra Corporation website at the following Internet web address "www.zebra.com/SS/manuals.htm." The aforementioned documents are hereby incorporated by reference in their entirety. In the embodiment of Figs. 7-9, the "language" of the printer system is a hybrid of both ZPL and XML, thus the printer system 10 may be referred to as a "hybrid XML printer."

As is known, ZPL-based printers can handle "on demand printing" and "stored label format." On-demand printing means that when a data stream containing label data to be printed is sent to the printer, it is immediately printed. Stored label format means that templates for the labels may be created in advance and downloaded to the printer. The templates may then be recalled internally within the printer to use in conjunction with a data stream transmitted at a later time that contains the label data to be printed.

As will become more clear in the description below, one difference between the native XML printer system embodiment of Figs. 2-6 and the hybrid XML printer system embodiment of Figs. 7-8 is that in the hybrid XML printer system embodiment, the data templates are predefined in ZPL and are first downloaded to the printer. Also note that the hybrid embodiment of Figs. 7-8 is not a combination of the native XML printer system of the embodiment of Figs. 2-6 with a known ZPL printer. It is not a "two-in-one" printer that merely switches between a native XML mode and a ZPL mode.

The hybrid XML printer system of Figs. 7-8 is different from the native XML printer system of Figs. 2-6 in which XSLT is used to define and layout the label in conjunction with XSL formatting elements embedded in the XSL template. To handle the layout and formatting of a label in the embodiment of the native XML printer system of Figs. 2-6, a company (or vendor) must prepare and define a label layout using XSL formatting objects, and in many situations, must also prepare XSL stylesheets and XSL schema, as set forth above in the native XML printer embodiment. If the business entity's ERP system 88 was able to transmit well-formed XML data using XSL formatting objects and the like (i.e., XSL style sheets and schema) in the first place, then there would be no need to use the ZPL templates, which would indicate that the business entity already made the complete switch to a fully compatible XML ERP system, thus obviating the need for any form of ZPL.

Some companies, however, may not be ready or willing to make the investment to such a "full conversion" of its system into "pure XML" because such companies may have a sizeable investment in the older ZPL templates for their labels to be printed. Many companies are building infrastructure to permit XML data to be easily passed from business process to business process because transmission of such data in XML compatible format is becoming the industry standard. Although many companies, and the industry as a whole, are moving in the direction of using XML format for data, such companies may not want to scrap their investment in their template database infrastructure just to make the switch to a fully compatible XML format.

Accordingly, the hybrid XML printer embodiment of Figs. 7-8 permits use of existing ZPL templates with data in XML format. The ERP system 88 may continue to use the existing ZPL templates because the "complete" switch to well-formed XML has not yet been made,

most likely for economic reasons. Thus, XSL stylesheets, schema and formatting objects have not yet been implemented at the ERP site, and the hybrid XML printer system provides a cost-effective solution in such environments.

Turning now to Fig. 7 and code segment 5, the code segment shown immediately below entitled "code segment 5 -- a ZPL downloadable template" shows a specific example of a ZPL format template that may be transmitted to the hybrid XML printer system 300 prior to the printing of an actual bar code label, and which may be saved in the memory of the printer system for future printing jobs. Line numbering has been inserted for purposes of illustration only and is not part of the code.

Code Segment 5 -- a ZPL downloadable template

```
1       ^XA
        ^DFE:SHIPLABL.ZPL^FS
        ^MCY
        ^LRN^FWN^CFD,24^LH0,0
5       ^CI0^PR2^MNY^MTT^MMT^MD0^JJ0,0^PON^PMN
        ^LT0
        ^A0N,58,58^FO188,169^CI0^FN999^FDname^FS
        ^A0N,58,58^FO188,244^CI0^FN998^FDaddress^FS
        ^A0N,58,58^FO188,319^CI0^FN997^FDcity^FS
10      ^A0N,58,58^FO657,319^CI0^FN996^FDstate^FS
        ^A0N,58,58^FO188,394^CI0^FN995^FDzip^FS
        ^BY4^FO188,488^BZN,35,N^FN995^FDzip^FS
        ^PQ1,0,1,Y
14      ^XZ
```

As is known in the art, the commands shown at lines 1 and 14 (^XA, ^XZ) define the beginning and end of the template, as is specified in the ZPL language. Line 2 indicates that the format template downloaded to the hybrid printer system 300 is stored in the memory of the hybrid printer at a location defined by "E:SHIPLABL.ZPL," where "E:" represents a storage device identifier, which may be, for example, a disk drive. However, any suitable storage device may be used. The inclusion of this command indicates that this format template is to be saved. Lines 3-6 represent some of the various printer setup and initialization commands provided for by the ZPL programming language.

Referring now to line 7, the "^A0" command with an associated parameter of "N" means that a text field is to be created using font number zero, and in a normal ("N") orientation (i.e., the font is not rotated), where "58, 58" represents the height and width (in dots) of the text that will be printed. As provided for in the ZPL language, the command "^FO188, 169" defines the field origin so that the text will be positioned at coordinates 188,

169 (i.e., 188 dots across and 169 dots down from the beginning of the label).

The command "^FN999" (FN command with the parameter of "999") defines a variable field, which will contain variable information to be downloaded to the printer at a later time. In that regard, this is somewhat different than in standard ZPL. In standard ZPL, when an "^FN" command in a template is received, no data is supplied along with the command because when the template is later recalled, a subsequent "^FN" command is sent that specifies the location of that variable data, followed by the data itself. In this hybrid embodiment, however, certain "association" data is provided immediately following the "^FN" command, as indicated by the "^FD" data field command. In this specific example, the data immediately following the "^FN" command, but preceded by the "^FD" data field command, is shown as "name," which is the data that will be matched with the XML data sent from the ERP or warehouse system at a later time. Thus, when the template shown in code segment 5 is received by the hybrid XML printer system 300, it is stored in memory of the printer, and additionally, the "association" between "name" and field "999" is stored.

Similarly, on line 8, the association between "address" and field 998 is stored, on line 9, the association between "city" and field 997 is stored, and so on. Note that the fields represented by, for example, "name," "address," "city," and the like, are not the actual data to be sent as XML data by the ERP system 88. Rather, "name," "address," and "city" are "tags" or "associations" that associate the data that will later be sent by the ERP system with the field location or identifier, namely, "999," "998," and "997," respectively.

Turning now to line 12, the "^BY" command defines default parameters for barcode commands which follow it. In this case the parameter of 4 defines the narrow bar width to be 4 dots. The "^FO" command defines that the barcode to be printed will be placed at coordinates 188, 488. The "^BZ" command defines a barcode, which is a postnet type of barcode format, with normal orientation (parameter of N), height of 35 dots (parameter of 35), and no human-readable interpretation line (parameter of N). Further, the "FN" command with a parameter of 995 specifies the association between the label field identified as "995" with name of the XML data to be downloaded, which is defined by the data command "^FD" having an attribute of "zip," which further specifies that "zip" is the name of the XML data to be downloaded to create the actual printed barcode. The actual barcode that is printed is the

zip code 304, as shown in Fig. 9.

Of course, text fields are printed on the barcode label 305 as well as the barcode itself 304, as shown in Fig. 9. Such fields, may include for example, a name 306, address 308, city 310, state 312, and text zip code 314.

Thus, in this embodiment, the template shown in code segment 5 is stored in the hybrid XML printer system 300, and no real "work" is performed (i.e., a label is not yet printed). In summary, lines 7-11 define text fields and line 12 defines a bar code that are defined along with associations that will be correlated with actual XML data sent from the ERP system 88 at a later time, along with other parameters, as set forth above. The template of code segment 5 may define which fields are static and which fields are variable. The variable fields have a variable name (stored in the ^FD command), which may be later correlated or "associated" with data to be downloaded at a later time, which data corresponds to the printed label.

Thus, in the hybrid embodiment of Figs. 7-8, the "^FD" command (i.e., followed by "name") determines the name of the variable that will be sent in the actual XML data that eventually follows the downloading of the template, rather than the actual data that will be printed on the barcode label.

To see how the format template of code segment 5 above is used in operation, code segment 6 is shown immediately below, and is entitled "code segment 6 -- downloaded XML data for use with a ZPL template." Line numbering has been inserted for purposes of illustration only and is not part of the code.

Code Segment 6 -- downloaded XML data for use with a ZPL template

```
1     <?xml version="1.0" encoding="UTF-8"?>
      <!DOCTYPE labels SYSTEM "label.dtd">
      <labels _FORMAT="E:SHIPLABL.ZPL" _QUANTITY="1" _PRINTERNAME="Printer
      1" _JOBNAME="LBL101">
5           <label>
                  <variable name="name">Albert Einstein</variable>
                  <variable name="address">1234 Relative Way</variable>
                  <variable name="city">Princeton</variable>
                  <variable name="state">NJ</variable>
10                <variable name="zip">08540</variable>
            </label>
            <label>
                  <variable name="name">Steven Hawking</variable>
                  <variable name="address">5678 Black Hole Drive</variable>
15                <variable name="city">Los Angeles</variable>
                  <variable name="state">CA</variable>
```

```
            <variable name="zip">90007</variable>
        </label>
        <label>
            <variable name="name">Richard Feynman</variable>
            <variable name="address">90 Quantum Circle</variable>
            <variable name="city">New York</variable>
            <variable name="state">NY</variable>
            <variable name="zip">10044</variable>
        </label>
    </labels>
```

Code segment 6 represents the actual XML data stream that the ERP system 88 or warehouse management system would send to the hybrid XML printer system 300, but after the format template of code segment 5 was first downloaded to the hybrid XML printer system. In the specific example of code segments 5 and 6, the warehouse management system that may be used, for example, is an Oracle warehouse management system (the "Oracle system" or "Oracle WMS"). Of course, any business system may be used, such as Oracle, SAP, and the like. In this specific example, the Oracle system gathers data from its database and formats the data in a particular XML format for transmission to the printer. Because this specific example uses the Oracle system, the template of code segment 6 uses Oracle XML semantics.

In that regard, the particular XML semantics used by the Oracle warehouse management system is defined in line 2 of code segment 6 by the "DTD" or document type definition, which is identified as "label.dtd," as is known in the art. Also, as is known in the art, the ERP system 88 or Oracle system has the capability of specifying the format in which the XML data is to be printed. By way of background, this capability may have been originally built into the Oracle system and other ERP systems because such systems were designed to interface to a custom "middleware" software system, which handled the format conversion between the Oracle or ERP system and the particular barcode printer in use. The present hybrid printer eliminates the need for such a "middleware" system.

However, the XML data transmitted by the Oracle system specifies on line 3 of code segment 6 that there is an XML element called "labels," which has an attribute called "_FORMAT." This attribute specifies the particular format template to use. In this specific example, the format template to use with the transmitted XML data is identified as "E:SHIPLABL.ZPL," which is the format template previously stored in the hybrid XML printer system 300, as shown in code segment 5, and which is a ZPL based format template.

Thus, the format template previously downloaded and stored in the memory of the hybrid XML printer system 300 may be recalled from a location specified by "E:SHIPLABL.ZPL."

Line 3 also contains some additional attributes of the "labels" element, such as "_QUANTITY," "_PRINTERNAME," and "_JOBNAME." The attribute "_QUANTITY," specifies the number of barcode labels to print, while the attributes "_PRINTERNAME," and "_JOBNAME" are legacy fields not needed by the hybrid XML printer system 300, but are nonetheless included in the Oracle data because such data was originally created to interface to the middleware software, as described above.

The code segment 6 contains three groups of "label" data, as shown at lines 5-11, 12-18, and 19-25, respectively. Note that the three groups of "label" data appear to be similar to code segment 1, namely lines 8-14, 15-21, and 22-28 respectively, because both code segments refer to "Albert Einstein," "Steven Hawking," and "Richard Feynman," respectively, and have associated fields for address, city, state, and zip code data. Accordingly, the data shown in code segment 6 corresponds to three individual labels to be printed. Note, however, that for each of the three labels to be printed, all of the addressee information is printed as text, but only the zip code field is printed as an actual barcode. This is because of the "^BZ" command defined in template of code segment 5, line 12.

Referring to code segments 5 and 6, the data transmitted by the Oracle system shown in code segment 6 is then associated with the corresponding format template shown in code segment 5. For example, in line 6 of code segment 6, the variable name of "name" corresponds to "Albert Einstein," which is the actual XML text data. Using the previously saved template of code segment 5, an association is made between "name" (the contents of which are "Albert Einstein") and the field number "999," as shown in line 7 of code segment 5. The XML data "Albert Einstein" is then saved in memory defined by the field number 999.

Similarly, the XML data "1234 Relative Way" is saved in memory defined by the field number 998, the XML data "Princeton" is saved in memory defined by the field number 997, the XML data "NJ" is saved in memory defined by the field number 996, and so on. Once all of the data contained in the XML data input stream have been associated with the corresponding template and have been loaded into memory, the label is printed.

Turning now to Figs. 7 and 8, Fig. 7 shows a high-level functional block diagram of a

specific embodiment of the hybrid XML printer system 300, while Fig. 8 illustrates a combined high-level software block diagram and data flow diagram of a specific embodiment of the hybrid XML printer system. The hybrid XML printer system 300 receives data from the enterprise resource planning (ERP) system 88, as described above with respect to the native XML printer system. The data may contain both XML components and ZPL components, such as XML data and/or the ZPL template data.

The high level blocks of Fig. 7 illustrate use of a SAX processor 320, which is a type of XML processor. By way of background, a SAX processor (Simple API for XML) does not use the DOM processing model (Document Object Model) used by the XML processors described above with respect to the native XML printer system of Figs. 2-6.

As described above, DOM-based XML processors create a node tree in memory in order to carry out its processing steps, as is known in the art. This is necessary when performing XSL type processing, as described above with respect to the embodiment of Figs. 2-6. Accordingly, all data is stored in memory, and thus may occupy a large amount of memory. A SAX processor, however, interprets XML syntax and data streams, and rather than building a node tree in memory, it creates "events" that are handled by event handlers, as is known in the art. SAX processors are very memory efficient, which is desirable when processing large amounts of XML data.

In this alternate embodiment of Figs. 7-8, however, XSL-type processing is preferably not performed, hence processing based on node trees in memory is not required. Accordingly, rather than using an XML processor that is strictly DOM-based, a SAX processor may be used, as shown in Fig. 7. The term "SAX" merely refers to a specification regarding how to handle data and syntax of the XML language. For example, a commercially available XML processor that may be used in this embodiment is the "Expat" XML processor available from Thai Open Source Software Center, Ltd. Although this specific model of XML processor has the capability to function as both a DOM-based XML processor or a non DOM-based SAX processor, for purposes of this embodiment, the processor is preferably configured as a non-DOM-based XML processor, and thus will be referred to as the SAX processor.

As shown in Fig. 7, in addition to the SAX processor 320, a ZPL processor 322 is also shown. The data paths from the SAX processor 320 and the ZPL processor 322 are then

routed to the block labeled Variable Data Integration 328. The Variable Data Integration block 328 combines the XML data received from the ERP system 88 and applies or "associates" the data with the fields that were previously defined by the ZPL template, as mentioned above with respect to the ZPL template of code segment 5. Essentially, in this step, the actual XML data, for example, the name "Albert Einstein," is saved in its associated location defined by 999 of the template of the code segment 5, and the like.

Next, the data is rendered by a Bitmap/Barcode Rendering block 330, which processing is similar to known ZPL bitmap rendering used in known commercially available ZPL printers. For example, the following known ZPL printers, which are commercially available from Zebra Technologies Corporation of Vernon Hills, Illinois, may include software, hardware, and/or firmware able to perform the ZPL bitmap/barcode processing as follows: Zebra Models XiIII+, 90XiIII, 96XiIII, 140XiIII, 170XiIII, 220XiIII, Model Z4M, Z6M, 105SL and any Model Xi series barcode printers from Zebra Corporation. Note that the Bitmap/Barcode Rendering block 330 is preferably performed without use of any XSLFO processing, which processing was required in the native XML printer system embodiment, and no XSLT stylesheets or schema is used.

The barcodes are rendered as described above for the native XML printer embodiment, but preferably with the exception that the barcode parameters are not specified in the XML data as attributes, but rather, are preferably specified by ZPL command parameters in the format template. For example, the ZPL command "^BY2,2,100" specifies, for a linear barcode, that the narrow bar width will be 2 dots, the ratio of the wide bar width to narrow bar width will be 2, and the height of the bars will be 100 dots.

After the bitmap is rendered, it is routed to the printer driver 54, which is also shown in Fig. 1. The printer driver 54 is an example of a component described above whose function may be performed by either the processing device in the computer system 12 or the processing device 60 (Fig. 1), depending upon the physical location of the associated processing device. Again, a single processing device, if sufficiently powerful, may handle all functions for the hybrid XML printer system 300.

Referring now to Figs. 1, 7 and 8, Fig. 8 illustrates a specific embodiment of a combined high-level software block diagram and data flow diagram. The software described

below may be executed by the processor 28 of the computer system 12 of Fig. 1. Again, the processor 28 may perform functions common to both the computer system 12 and the printer system 50. There may be one or more processors, which may function in concert or which may function separately. It is not material to the scope of this invention whether the processing or processing functions are performed by or in the computer system or by or in the printer system.

As will be described below, multiple "passes" through the processing steps shown in Fig. 8 may occur under control of the SAX processor 320. However, such multiple "passes" are not necessarily linear and thus, an exact path shown by specific lines or arrows in Fig. 8 may not necessarily be shown. An interative process is described that may not have an exact beginning and ending point shown on Fig. 8. Rather, an action or result generated by the pass may be shown, and thus an "alternative path" may be shown in Fig. 8 in dashed lines that returns processing control back to the SAX processor 320. This is a shorthand way of indicating that the path may be iterative and that the SAX processor 320 may loop or iterate many times before processing reaches its end.

For purposes of illustration only, in a first "pass" through the processing illustrated in Fig. 8, a ZPL template is first sent to the hybrid XML printer system 300, where the actual data to be printed is not yet sent. In a second "pass," as will also be described below, the XML data representing the barcode label to be printed is sent to the printer system, to be printed in accordance with the ZPL template previously saved.

As shown in Fig. 8, a data input stream 340 is received from the ERP system 88, and as described above, may contain XML data and/or ZPL template data. The SAX processing block 320 initially receives and parses the data input stream 340. Note that the SAX processing block 320 is actually performed by an XML processor because a SAX processor is an XML processor operating in a SAX mode. Thus, the phrase XML processor or XML processing block may be interchanged with the phrase SAX processor or SAX processing block. However, as described above, the XML processor preferably operates in SAX mode, and is thus referred to as the SAX processor 320.

First, the SAX processing block 320 determines whether the data input stream 340 is well-formed XML data by applying standard XML syntax rules to the data input stream. Note

that the SAX processing block 320 of Fig. 8 is the essentially the same as the SAX processor of Fig. 7 with Fig. 7 showing some hardware and Fig. 8 showing data flow. The terms SAX processor and SAX processing block will be used interchangeably herein.

If the SAX processing block 320 determines that the data input stream does not conform to XML syntax, then it is assumed that the data input stream is ZPL based. Accordingly, the SAX processing block 320 routes the data input stream 340 to the ZPL processing block 322 for further processing. Note that the ZPL processing block 322 of Fig. 8 is the essentially the same as the ZPL processor of Fig. 7 with Fig. 7 showing some hardware and Fig. 8 showing data flow. The terms ZPL processor and ZPL processing block will be used interchangeably herein.

The ZPL processing block 322 then processes or parses the data input stream 340, which represents the ZPL format template. Preferably, the ZPL processing block 322 receives a "beginning of label" command (^XA), which causes it to begin processing the format template. The format template is shown in the code segment 5.

Various ZPL commands govern the processing and define the fields of the label, essentially establishing and defining the data to be printed. Printer control and other commands may be received at this stage of processing. When all of the format template has been downloaded, the ZPL processing block 322 encounters an "end of label" command (^XZ), which causes the format template to be saved. Note that the ^DF command with the corresponding identifier "E:SHIPLABL.ZPL," (shown at line 2 of code segment 5) indicates to the ZPL processing block 322 that the format template is preferably saved internally in the printer system on the "E:" device in a file named "shiplabl.zpl." Of course, the format template need not be saved internally in the printer system and may be saved at any storage location, such as on a networked server or a remote location accessible through a communication network.

The ^DF (Download Format) ZPL command indicates that the label format of which it is part is to be stored rather than immediately printed. The save actually takes place when the ^XZ is encountered which indicates that all template data has been received.

Also note in Fig. 8, a block labeled "ZPL label formats" (352) without vertical lines completing the box is the symbol in the figure for a data store. This indicates that the template

data coming from the input stream is stored in a data store. The line exiting the data store 352 indicates that at some later time the data is retrieved from the data store for additional processing. The data flow path is from the block 360 to the store to the block 356, but this does not represent the actual "control flow," which is not a linear flow.

Although a block labeled "recall ZPL label format" 356 is shown connected to the block labeled "save ZPL label format" 360 as mentioned above, processing may not be linear, and thus data flow may not necessarily continue or flow from the "save ZPL label format" block 360 to the "recall ZPL label format" block 356. Rather, the entire process is iterative as described above, and in this initial "pass," after the ZPL label format is saved, as shown in the block 360, control of the input data stream may revert back to the SAX processing block 320 for continued processing, as shown by dashed lines 370. If there is no additional data in the data input stream 340 to be processed, processing terminates and awaits additional downloaded data.

In a second "pass" through the processing illustrated in Figs. 7-8, the ERP system 88 may transmit the actual XML data, which will be processed by the hybrid XML printer system 300 in accordance with the previously downloaded ZPL template. For example, the SAX processing block 320 may receive well-formed XML data, which may specify that label data is to follow. To process the actual data, the stored ZPL format is recalled, as shown in the block labeled "recall ZPL label format" 356, so that the recalled format template may be used in conjunction with the incoming XML data stream.

The SAX processing block 320 processes the XML syntax and determines various elements and attributes defined by the data input stream 340. In this embodiment upon receiving a "_FORMAT" attribute, the data input stream 340 is temporarily interrupted and input data comes from the format template stored in the printer system 300, and thus the ZPL commands embedded in the format template are interpreted by the ZPL processing block 322. Once the commands in the format template have been interpreted, the data input stream 340 again becomes the SAX processing block 320, which continues to process the XML data.

Additional passes through the processing under control of the SAX processor 320 may cause various "events" to be initiated, as described above. For example, a "variable data SAX event" 376 may be initiated so that the various fields in the format template are populated with

the corresponding or "associated" XML data being received. This is shown by the block labeled "Match XML variable with ZPL label format variable" 380. Essentially, the variables in the XML data input stream 340 are associated with the fields in the format template. For example, in this step, the XML variable, such as the actual XML data "Albert Einstein" (306 of Fig. 9) shown in code segment 6, is matched with the ZPL label format variable, such as the label variable "name" in the ZPL template of code segment 5.

The SAX processor 320 may initiate further events to handle the tasks described above, such as a "begin label SAX event" 384, and "other SAX events" 388. This pass or other passes through the processing of Fig. 8 under control of the SAX processor 320 may cause the generation of "other SAX events," which may be processed by a block labeled "Process event" 390. For example, such "other" events may include "set printer speed event," "set quantity of labels to print event," "select media event," "select ribbon to use event," "choose paper event," "use 300 dpi printer event," "use 600 dpi printer event," "variable data event," and the like.

The number of such events are virtually unlimited, and are governed by the type, configuration, and capability of the hardware printer system. Essentially, all aspects and functions of the printer may be controlled by use of codes sent in the data input steam 340. This also allows the printer to determine if it is capable or configured to print the label described by the XML data stream, and to optionally return error information if it cannot. Although in code segment 6 corresponding to the Oracle-base XML data, only a limited number of "other commands" are shown (namely, quantity=1 and printer name = printer 1, both shown on line 3), essentially, any function or feature that may be controlled in the printer may be specified in the XML input data stream, as mentioned above. Thus, there may be a close binding between the actual XML input data and printer control codes.

Also note that such "other" commands and corresponding "other events" generated by the SAX processing block 320 in response to the commands are not part of or specific to the Oracle system, meaning that the Oracle system does not necessarily "know" what these commands do or how they are used. It is only when such downloaded commands are matched during SAX processing that the commands and subsequent generated events take on meaning.

When an "end of label element" event (</label>) is received, the label is complete and

now may be printed. Control is then routed to the ZPL formatting engine processing 394, which causes the label to be printed. Preferably, the function of the ZPL formatting engine processing 394 includes the function of bitmap/barcode rendering. Thus, the block 394 will be referred to as the ZPL formatting engine. This combined function determines the bitmaps for each of the text, barcode, and other fields on the label, integrates the individual bitmaps with proper rotation and magnification into a single label bitmap 396, and sends the completed label bitmap to the print engine (or printer driver 54, Fig. 1) for physical rendering on media. As described above, the ZPL formatting engine processing 394 may be found in various commercially available ZPL printers, such as for example, in printers commercially available from Zebra Technologies Corporation of Vernon Hills, Illinois, which may include the following printer models: XiIII+, 90XiIII, 96XiIII, 140XiIII, 170XiIII, 220XiIII, Model Z4M, Z6M, 105SL and any Model Xi series barcode printers.

The ZPL formatting engine processing 394 essentially determines all of the fields contained in the label to be printed and determines what must be done for each field. For example, the ZPL formatting engine processing 394 may determine that the field containing the data "Albert Einstein" (306 of Fig. 9) shown in line 6 of code segment 6 is printed using font number zero and in a normal ("N") orientation, and is 58 dots in height and 58 dots in width, and begins at a field origin coordinate of 188, 169, as indicated by line 7 of the code segment 5.

Thus, the ZPL formatting engine processing 394 inspects all of the fields defined by the format template in conjunction with the corresponding XML data and essentially determines how to create dots (or bits) on the label that represent the text or barcode. Barcode fields are formatted and rendered by applying the appropriate bitmap generation algorithm to the field parameters based upon the barcode symbology specified for the field accounting for horizontal and vertical sizing of linear barcodes and magnification of two-dimensional barcodes.

Specific embodiments of an XML printer system, including a native XML printer system and a hybrid XML printer system, according to the present invention have been described for the purpose of illustrating the manner in which the invention may be made and used. It should be understood that implementation of other variations and modifications of the

invention and its various aspects will be apparent to those skilled in the art, and that the invention is not limited by the specific embodiments described. It is therefore contemplated to cover by the present invention any and all modifications, variations, or equivalents that fall within the true spirit and scope of the basic underlying principles disclosed and claimed herein.

CLAIMS

WHAT IS CLAIMED IS

1. An XML printer system adapted to print bar code labels based upon an extensible markup language (XML) input data stream, the XML printer system including a printer apparatus having a media control system, a print head assembly, and a print head driver, the XML printer system comprising:

a computer system operatively coupled to the printer apparatus;

the computer system further including

a memory subsystem;

a communication interface operatively coupled to a network;

an XML processor configured to receive and process the XML input data stream, and obtain schema identified in the XML data stream from a schema repository, the XML processor validating the XML data stream based upon the schema obtained;

an extensible stylesheet language transformation (XSLT) processor configured to obtain a stylesheet identified in the XML data stream from a stylesheet repository, the XSLT processor transforming data in the XML input data stream into transformed XML data based upon the stylesheet obtained;

an extensible stylesheet language formatting object (XSLFO) processor configured to format the transformed XML data into formatted XML data based upon XSLFO instructions contained in the stylesheet; and

a barcode rendering subsystem configured to receive the formatted XML data and generate a bit map representative of the bar code label.

2. The system according to claim 1 wherein the memory subsystem is selected from the group of memory components consisting of RAM, ROM, hard disk storage and database system.

3. The system according to claim 1 wherein the memory subsystem is remote from the computer system and accessible through a network.

4. The system according to claim 1 wherein the memory subsystem is incorporated into the computer system.

5. The system according to claim 1 wherein the computer system includes a

processor selected from the group consisting of a computer, microprocessor, RISC processor, CISC processor, mainframe computer, work station, single-chip computer, distributed processor, server, controller, micro-controller, discrete logic computer and internet computer.

6. The system according to claim 1 wherein the XML schema and the XSLT stylesheet are XML documents.

7. The system according to claim 1 wherein the computer system and the printer apparatus are housed in a common enclosure.

8. The system according to claim 1 wherein the computer system and the printer apparatus are housed in separate enclosures.

9. The system according to claim 1 wherein the computer system and the printer apparatus are housed in separate enclosures and communicate with each other through a network.

10. The system according to claim 1 wherein an enterprise resource planning (ERP) system sends the XML input data stream to the XML printer system to initiate printing of the barcode label.

11. The system according to claim 10 wherein the enterprise resource planning system is a warehouse management system.

12. The system according to claim 10 wherein the enterprise resource planning system is coupled to the XML printer system through the network.

13. The system according to claim 10 wherein the XML processor transmits an error indication to the enterprise resource planning system if the XML input data stream is not successfully validated in accordance with the schema.

14. The system according to claim 1 wherein the network is selected from the group of networks consisting of a local area network, remote network, dedicated network, Ethernet, Internet, public-switched telephone network (PSTN), and plain-old telephone system (POTS).

15. The system according to claim 1 wherein the stylesheet repository contains a plurality of stylesheets and is accessed remotely by the computer system through the network.

16. The system according to claim 1 wherein the schema repository contains a plurality of schema and is accessed remotely by the computer system through the network.

17. The system according to claim 1 wherein at least one of the stylesheets in the repository and at least one of the schema in the repository are updated periodically.

18. The system according to claim 1 wherein at least one of the stylesheets obtained from the stylesheet repository and at least one of the schema obtained from the schema repository are temporarily stored in the memory subsystem after retrieval from the respective repositories.

19. The system according to claim 1 wherein the barcode rendering subsystem processes an in-stream foreign object defined in the stylesheet.

20. The system according to claim 19 wherein the in-stream foreign object is a barcode-type object.

21. The system according to claim 20 wherein the XSLFO processor cannot process an in-stream foreign object of the barcode type.

22. The system according to claim 1 wherein the printer apparatus contains at least one algorithm used to generate a barcode symbol based upon a barcode type specified in the stylesheet.

23. The system according to claim 1 wherein the barcode rendering subsystem provides the bitmap to the print head driver to facilitate printing of the bar code label.

24. The system according to claim 1 wherein an external source transmits a request to the printer system to generate a barcode label bitmap.

25. The system according to claim 24 wherein the external source transmits the request over the network.

26. The system according to claim 24 wherein the external source transmits the XML input data stream to the printer system.

27. The system according to claim 1 wherein an external source transmits a request to the printer system to generate a barcode label bitmap, and in response to the request, the XML printer system processes the request in accordance with claim 1 and transmits the bitmap to the external source.

28. The system according to claim 1 wherein the computer system receives a request from an external source to render a barcode label, the external source transmitting the XML input data stream, and wherein the bitmap generated by the barcode rendering

subsystem is transmitted to the external source.

29. For use in a barcode printer apparatus, an XML processing system comprising:

a computer system operatively coupled to the barcode printer apparatus;

the computer system further including

an XML processor configured to receive and process an XML input data stream, and obtain schema identified in the XML data stream from a schema repository, the XML processor validating the XML data stream based upon the schema obtained;

an extensible stylesheet language transformation (XSLT) processor configured to obtain a stylesheet identified in the XML data stream from a stylesheet repository, the XSLT processor transforming data in the XML input data stream into transformed XML data based upon the stylesheet obtained;

extensible stylesheet language formatting object (XSLFO) processor configured to format the transformed XML data into formatted XML data based upon XSLFO instructions contained in the stylesheet; and

a barcode rendering subsystem configured to receive the formatted XML data and generate a bit map representative of the bar code label.

30. A method for generating a bar code label in an XML printer system adapted to print bar code labels based upon an extensible markup language (XML) input data stream, the XML printer system including a printer apparatus having a media control system, a print head assembly, and a print head driver, the method comprising:

obtaining schema identified in the XML data stream from a schema repository;

processing the XML input data stream using an XML processor;

validating the XML data stream based upon the schema obtained;

obtaining a stylesheet identified in the XML data stream from a stylesheet repository;

transforming data in the XML input data stream into transformed XML data using an extensible stylesheet language transformation (XSLT) processor based upon the stylesheet obtained;

formatting the transformed XML data into formatted XML data using an extensible stylesheet language formatting object (XSLFO) processor based upon XSLFO instructions contained in the stylesheet;

generating a bitmap representative of the bar code label using a barcode rendering subsystem based on the formatted XML data, the barcode rendering subsystem processing a barcode-type instream foreign object defined in the stylesheet; and

transmitting the bitmap to the print head driver to facilitate printing of the bar code label

31. For use in a barcode printer apparatus, an XML processing system comprising:

a computer system operatively coupled to the barcode printer apparatus;

the computer system further including

an XML processor configured to receive and process an XML input data stream and obtain schema identified in the XML data stream, the XML processor validating the XML data stream based upon the schema obtained;

an extensible stylesheet language transformation (XSLT) processor configured to obtain a stylesheet identified in the XML data stream, the XSLT processor transforming data in the XML input data stream into transformed XML data based upon the stylesheet obtained;

extensible stylesheet language formatting object (XSLFO) processor configured to format the transformed XML data into formatted XML data based upon XSLFO instructions contained in the stylesheet; and

a barcode rendering subsystem configured to receive the formatted XML data and generate a bit map representative of the bar code label.

32. An XML printer system adapted to print bar code labels based upon an extensible markup language (XML) input data stream and a format template, the printer system including a printer apparatus having a media control system, a print head assembly, and a print head driver, the printer system comprising:

a computer system operatively coupled to the printer apparatus;

the computer system further including

a memory subsystem;

a SAX processor configured to receive and process the XML input data stream;

a ZPL processor configured to receive and process the format template;

a variable data integrator configured to develop an association between the XML data contained in the XML input data stream and the format template;

a formatting engine configured to format the associated XML data according to a format governed by the format template; and

a barcode rendering engine configured to receive the associated and formatted XML data and generate a bit map representative of the bar code label.

33. The system according to claim 32 wherein the SAX processor generates a plurality of events based upon commands embedded in the format template.

34. The system according to claim 33 wherein the events are selected from the group consisting of a set printer speed event, set quantity of labels to print event, select media event, select ribbon to use event, choose paper event, use 300 dpi printer event, use 600 dpi printer event, and variable data event.

35. The system according to claim 32 wherein the format template is sent to the SAX processor and saved for use with downloaded XML data, the XML data download at a subsequent time.

36. The system according to claim 32 wherein the format template is a ZPL-based format template.