Speaking In Tongues: Sorting Out Variable Data Printing Languages

By Eliot Harper

As the digital color print market continues to grow, adoption of variable data printing is increasing rapidly. Personalization has become commonplace. In this overview, we look at the proliferation of digital file formats devised to cope with the practical side of variable print communication.

Variable data printing has long since been opened out of it's flat-pack carton and assembled to join the rest of the industry acronym furniture, along with UCR, GCR, CtP, JDF, CIP4 and a few forgotten armchairs. Today, VDP accounts for a healthy share of print volume; 37% of graphic arts firms (printers and trade shops) produce some sort of VDP jobs in-house, up from 28% one year ago*. The variety of VDP applications in use today ranges from simple business correspondence with name, address and basic information changing for each recipient, through to direct mail applications where graphical and text elements are switched based on a set of business rules to produce a unique composition, customized to each recipient.

Furthermore, VDP is no longer limited to print. The term has been adopted as "variable data publishing" to include other media channels. A multichannel marketing campaign can incorporate personalized content in a printed piece and offer supporting content and response channels through Web, e-mail and mobile devices.

Although VDP has been used for over a decade, personalized digital printing is hardly new. Essential mail (bills, statements, etc.) has been produced on digital printers ever since Xerox introduced the first 9700 laser printer in 1977. These essential documents were printed from mainframe environments using optimized data stream languages such as Metacode and LCDS, which are still widely used in transactional printing environments.

When digital color printing emerged in the early 1990s, these legacy data stream languages were not suitable for personalized color documents, and as a result many printer and RIP vendors developed their own variable information (VI) languages.

Today, VDP software products and RIPs support a host of different VI languages, and selecting an appropriate language can be a somewhat daunting task. In this article we identify and describe all of the variable information languages so that you can make an informed decision when you choose an appropriate language for your VDP work.

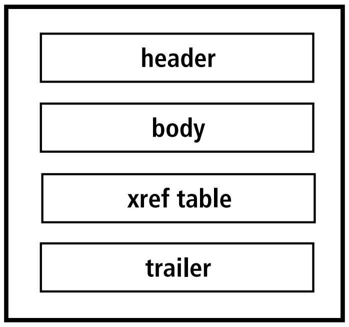
Most operating system print drivers create a page description language, or PDL (typically PostScript), by processing each page of a document individually. If such print drivers are used to create a print-ready file for VDP applications, the resulting files would contain a separate page (or pages) for each record. This can result in very large file sizes and might require a considerable amount of time for RIPs or print controllers to interpret and process the data.

To address this issue, several variable information languages have been developed specifically to contain page description information for VDP applications that overcome the limitations of traditional page description languages. These VI languages enable print files to be created using various optimization techniques, including object caching and custom page instructions that can be interpreted by supported RIPs.

Optimized Portable Document Format (PDF)

Adobe Systems' portable document format (PDF), developed in 1993, represents documents within a device- and display resolution-independent fixed-layout document format. Built on a subset of PostScript, PDF files encapsulate a complete description of all objects within a document, including text, fonts, graphics and vector objects. PDF files also include a structured storage system to bundle all page objects and other content into a single file, using data compression where appropriate.

The general structure of a PDF file consists of a header, body, cross-reference (xref) table and trailer. The trailer contains pointers to the xref table and to key objects contained in the trailer dictionary. The xref table contains pointers to all the objects included in the PDF file. It identifies how many objects are in the table, where the object begins (the offset) and its length in bytes. The body contains all the object information: fonts, images, text and other object types.



Since PDF was developed as a fixed-layout document format, a PDF file contains separate pages for each page to be printed. As a result, when used for variable data printing applications, the PDF file will contain a separate page (or pages) for each record, which can result in a PDF file several thousand pages long. However, PDF enables file optimization through its ability to reuse common objects in the document — both text and images. This not only results in a smaller file size, but also enables faster file processing at the RIP, as reused objects are cached at the RIP. This type of file optimization is commonly referred to as "thin" or "optimized" PDF.

As a result, the difference in file size between a PDF with 100 records and the same application with 1,000 records could be fairly minor, as common elements are stored and reused across multiple pages.

However, an optimized PDF file cannot be created by just any PDF driver, since the driver needs to identify repeating objects in a file and format them as reused content. One common method for creating an optimized PDF file is to use Adobe's PDF driver (installed with Acrobat) or create an optimized PostScript file, then create a PDF file from the PostScript file using Adobe Acrobat Distiller.

Optimized PostScript

Adobe Systems developed PostScript in 1984 for the desktop publishing market. PostScript and PDF share many similarities, as both file formats describe text and graphics. The main difference is that PostScript is a page description language and also a programming language that is processed by an interpreter to generate an image, whereas PDF is a file format and not a programming language.

As noted earlier, operating system print drivers create a PostScript file by writing each page individually, including all of the objects on every page. Therefore, creating a VDP application for many records can result in an extremely large file. This typically occurs when creating VDP applications using entry-level data merge utilities, as they rely on the operating system driver to create the print file, such as the built-in data merge functionality in Microsoft Word or Adobe InDesign. This type of PostScript file is often referred to as "fat" PostScript.

Similar to the PDF imaging model, PostScript also supports reusable content (or "form caching") of repeating objects by using the PostScript Level 2 form caching environment. This enables the creation of "thin" or "optimized" PostScript files, where repeating objects are only included once in the document. However, the software or driver used to create the PostScript file must support form caching to take advantage of this type of optimization.

Because PostScript is a programming language, PostScript files can include programmatic commands to draw page objects from data. For example, pie charts, line graphs or bar charts can be drawn from included data. Text content can also be written programmatically in PostScript so that words or blocks of text can change depending on defined rules. Since these commands are instructions in the PostScript file, pages can be composed by the PostScript interpreter (the raster image processor, or RIP) instead of using VDP software products for file composition.

Output device commands can also be included as variable instructions in PostScript. This enables the PostScript interpreter to call supported page tray and finishing features such as stapling, collating and folding. A programmed business rule could be defined in the PostScript file to delineate page tray-pull instructions for each record based on a value in the data. Letters for "regular" members might be printed on plain paper and letters for "premium" members on high-quality paper, for example.

The difference in file size between a PDF with 100 records and the same application with 1,000 records could be fairly minor, as common elements are stored and reused across multiple pages.

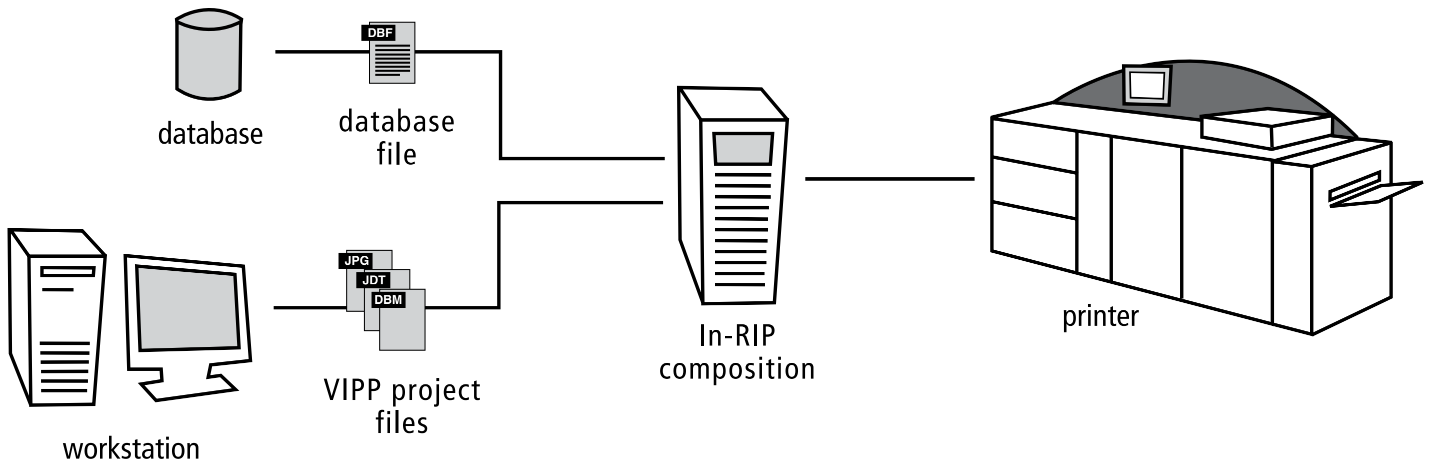
Although many VDP software products offer "optimized PostScript" output, only a few support RIP-level composition or "dynamic document composition." Most VDP software products use their own composition engine or page layout software (Adobe InDesign or QuarkXPress). This is not necessarily a disadvantage, as PostScript does not have the same level of typographic and graphic control as page layout software. However, RIP-side composition can present significant performance benefits since documents are composed and rasterized at the same time instead of a two-stage process in which documents are composed (as a PostScript file) on a desktop or server before being rasterized at the RIP.

Printer Command Language (PCL)

Printer Command Language (PCL) is a printer protocol originally developed by Hewlett Packard in the 1980s for early inkjet printers. PCL has been released in varying levels over the past 20 years and is now supported across a range of digital printing technologies.

PCL levels 1 through 5e/5c are command-based languages using control sequences that are processed and interpreted in the order in which they are received. In 1995, HP introduced PCL 6, a language very different from earlier PCL versions because it provided a stack-based, object-oriented protocol similar to PostScript.

Although PCL is not a variable information language, its file structure can store common page elements, so repeating page objects (such as images) only need to be stored in the PCL file once. PCL is supported by a few VDP software products.



Variable-data Intelligent PostScript Printware (VIPP)

Developed by Xerox, VIPP is a PostScript-based language designed to take advantage of the powerful programming features of PostScript, as well as address the limitations of PostScript in VDP applications.

With VIPP commands, VDP applications can remain independent of PostScript, since VIPP can use higher-level PostScript operators. They provide support for common VDP application requirements, including data-driven graphics and text commands for text highlighting and reflow across multiple frames, including pages.

VIPP can also process native data streams from legacy (line data) to XML, enabling independent data production and VDP application design. As a result, VIPP application resource files can be packaged as VIPP project container (VPC) files and loaded on the RIP, where the raw data file can be sent to the RIP to trigger document composition and production.

Although many VDP software products offer support for VIPP, only a few products take advantage of VIPP's intepreter-level composition model and "just send the data" workflow. Most variable data printing software products use their own composition engine or page layout software to compose the document and only use a few VIPP commands (such as form object caching). As indicated earlier, this is not necessarily a disadvantage, since page layout software such as InDesign includes powerful page layout features, drawing tools, typography control and more. As a result, document composition using page layout software might be more suited for design-intensive applications.

Variable Print Specification (VPS)

VPS is a PostScript-based language developed by Scitex (now Print On-Demand Solutions, a Kodak company). The VPS imaging model is constructed of pages, and each page is constructed of elements. There are two types of elements: reusable and non-reusable or "inline" elements. In this respect VPS is similar to other VI languages that use reusable element models, such as PDF, PostScript, PPML and VIPP. VPS is supported across a number of Creo/Kodak and EFI OEMed RIPs.

Personalized Print Markup Language (PPML)

PPML is an XML-based variable information language defined and developed by the Digital Print Initiative (PODi), a not-for-profit industry consortium of vendor companies that fosters digital printing growth through market and standards development activities. The PPML framework is built on two core methods, object-level granularity and reusable content.

Object-level granularity describes content objects on one page instead of individual pages. Reusable content refers to the ability to temporarily or permanently save page content to the RIP memory and use it throughout the composition of the VDP document or other VDP documents.

Reusable content can include fonts, graphics, images and other digital assets. Similar to VIPP and VPS, PPML content can reside locally on the RIP or it can be retrieved from another device by "referencing" remote resources via URLs, eliminating the need to send all the resources with the print job.

As PPML is an XML- (text) based language, it can't contain binary data. As a result, all internal graphical content in the PPML file (such as non-referenced content) has to be encoded. This encoding can result in significantly larger file sizes than comparable languages (like VPS or VIPP). As a result, it is good practice to use referenced content when creating PPML.

Like several other VI languages, PPML can dynamically merge objects (text and images) on a template at the RIP from the supplied dataset and digital assets.

Due to the large number of elements within the PPML definition and varying levels of PPML implementation across vendors, a Graphic Arts Conformance Specification has been created to define required PPML elements to ensure interoperability across different VDP software and RIPs. This specification defines how device-dependent colors are handled, which digital asset formats are supported and other related requirements.

Personalized Print Markup Language/Variable Data Exchange (PPML/VDX)

The American National Standards Institute (ANSI, **www.ansi.org**) approved the PPML/VDX standard, developed by the Committee for Graphic Arts Technologies Standards (CGATS), early in 2002. Formerly known as VDX, PPML/VDX is based on a subset of the PPML specification.

A PDF-based standard, PPML/VDX uses a subset of PPML to define the reusable content within the PDF file. A PPML/VDX file can consist of one or more files. A collection of one or more files is referred to as a "PPML/VDX Instance." In its most basic form, an instance will always contain a PPML/VDX layout file.

The layout file is a PDF file that functions as a container for the VDP template and variable elements (text and images). Although the layout file is a PDF, it uses the filename extension .vdx to signify that it is not a regular PDF file.

FreeForm 2, an extension of FreeForm, offers all the functionality of FreeForm while also providing full support for page picking and greater flexibility for database integration.

The layout file includes PPML information that defines the layout of the document, the structure of pages and the variable elements (text and images) used in the VDP document and how elements are joined to pages, while the PDF pages contain the variable elements.

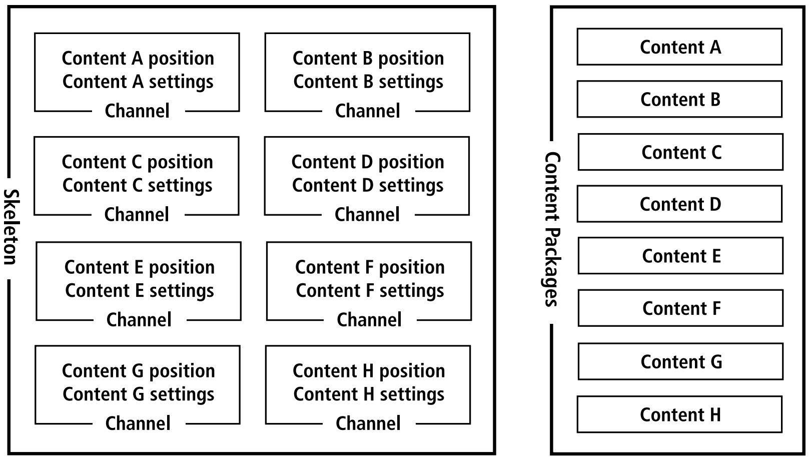
The layout file can also include job definition format (JDF) job ticketing instructions. The PPML information, variable elements and JDF instructions can either be contained within the layout file or referenced from the layout file and organized as individual files within the instance.

The layout file also contains a content table, or "checklist," that can verify that all required elements for the VDP application have been included in the instance.

FreeForm

Developed by Electronics for Imaging (EFI), FreeForm was one of the first color VI languages in the industry and is standard on most of EFI's Fiery branded RIPs.

The FreeForm imaging model uses two layers. A master layer (or template file) contains all static data (static images and PDF pages), while the variable layer contains all variable data (variable images and text). The layers, or files, are sent to the RIP separately. The template file is loaded onto the FreeForm-enabled RIP, where it is rasterized and assigned a user-defined numeric ID.



Once the template file is loaded on the RIP, the variable data layer (or job) is referenced to the corresponding template ID and is sent to the RIP. The RIP rasterizes the variable data job and overlays the rasterized master data with the rasterized data of the variable data job.

Using this template-based approach to VDP, the same master template can be reused for different or versioned VDP documents. In addition, FreeForm supports image caching, where repeating variable images are stored in reusable forms within the variable data layer.

FreeForm is an ideal variable information language for entry-level VDP documents and applications, but it offers only basic VI functionality and has limited support for advanced VI commands such as page picking (the ability to define individual pages in a document) and conditionally skipping layouts.

FreeForm 2

FreeForm 2, an extension of FreeForm, offers all the functionality of FreeForm while also providing full support for page picking and greater flexibility for database integration.

In addition, the master "template ID" in FreeForm 2 consists of a user-defined name rather than a number, which enables easier template management.

Job Layout (JLT)

Developed by Indigo (now HP), the job layout (JLT) language is a proprietary job description language used by HP Indigo Digital Presses.

JLT is not just a VI language, it is also a file format that defines document job structure and includes basic job ticket information. This proprietary file format enables integration of the print job with the software and hardware architecture of the HP Indigo press.

A JLT file contains two parts, a "skeleton" consisting of channels and "content packages." Channels in the skeleton define the position of the content (page objects) and settings that define the transformation of each object — for example image scaling and rotation. For variable data printing documents, these channels also define the link to variable content.

Content packages include any static content in the file (text and image page objects, for instance). Variable content is not included in content packages but is loaded on the RIP separately. However, using an optional "rich mode" setting, variable objects can be included as a PostScript file.

HP Indigo requires this unique workflow of separating the file structure from the content, as all page objects (static and variable) need to be rasterized to an internal format (Indigo Compressed Format). The ICF objects in the document are then assembled at the RIP according to the JLT structure or database file (for VDP documents).

One of the core differences in IJPDS compared to other VI languages is its support for built-in parallel processing of a single file.

Variable Data File (VDF)

Originally developed by Agfa for ChromaPress, the variable data file format was intended for the Agfa IntelliStream (now Xeikon) digital front end.

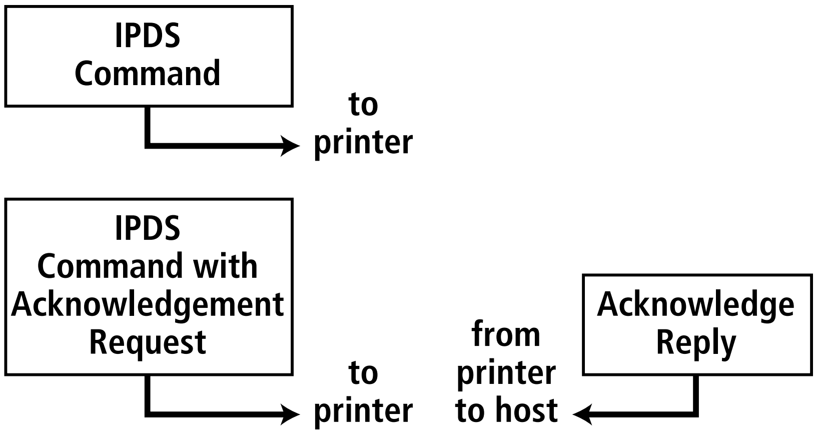
Variable data file is a PostScript-based language and the VDF workflow uses a document template that is saved as a PostScript file. Variable page objects are then saved as individual variable data files and a separate VDF file is created for each variable object. In addition, setup (STP) files can be created that contain information about each variable object, with a separate STP file created for each object. When STP files are available, the IntelliStream DFE can use its IntelliCache feature to cache variable objects.

Xeikon replaced the IntelliStream DFE (digital front end) in 2004 with a newer DFE architecture, the X-800, to support the current models of Xeikon print engines. In its new DFE, Xeikon has moved away from VDF support to PPML and PPML/VDX.

Intelligent Printer Data Streams (IPDS)

Developed by IBM for mainframe printing environments, intelligent printer data stream (IPDS) is part of IBM's advanced function presentation (AFP) architecture. The IPDS language contains the information necessary to identify, monitor and control the functions of certain kinds of printers that are used in mainframe environments. This information includes the characteristics of the printer, its resolution, what resources it has, whether it has sufficient memory and whether it can receive and print a job.

The IPDS architecture enables both spooled data (such as fonts, text, images) and print job management controls (like resolution, paper tray handling, media jams) to flow bidirectionally between both the print server (or print driver) and the printer controller. IPDS data streams are only used to carry print instructions and data from the print server to the printer in structured fields. The print controller processes IPDS commands, then returns an acknowledgment back to the print server.



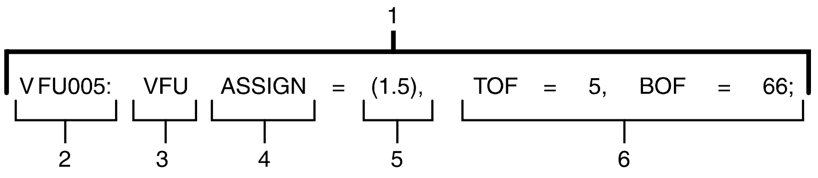
IPDS also provides support for media finishing using printer-attached devices or preprocessing and postprocessing devices. In addition to traditional printer-controlled finishing, constructs are also provided to enable IPDS data streams to be used within universal printer pre- and post-processing interface (UP3I) environments.

Inkjet Printer Data Stream (IJPDS)

Developed by Scitex, IJPDS is a proprietary file format for Scitex (now Kodak) Versamark printers built on a simple binary file format consisting of block-formatted page components.

IJPDS was developed primarily for printing variable text. Fonts are defined in the header file as bitmaps and page layout is based on lines of text. IJPDS support for images is limited and only images that are placed in line with text are supported. Furthermore, images have to be defined as a bitmap font glyph. For full-color images, four separate bitmap images are required (C, M, Y, K), which are placed over one another.

One of the core differences in IJPDS compared to other VI languages is its support for built-in parallel processing of a single file. For example, the IJPDS file could include instructions to process records across eight interpreters (or CPUs). These instructions are defined in the IJPDS file by a job control record that



1. Command statement
2. Identifier
3. Command keyword
4. Parameter keyword
5. Parameter option
6. Additional parameter keywords and options

indicates which interpreter the following records will refer to. The disadvantage of this parallel-processing approach is that the file becomes RIP-dependent, so it isn't possible to send a IJPDS file created for eight interpreters to a different RIP with two interpreters.

Line Conditioned Data Stream (LCDS)

Xerox developed LCDS to allow simple line data to take advantage of Xerox's 9700 and 4000 series printers. LCDS is a set of printing system commands that defines printer properties such as the appearance, output destination and paper feed source of a print job.

LCDS allowed easy migration from earlier impact-based printers by enabling page composition to be performed on the printer controller, where enabled forms, fonts and images were stored directly on the printer controller.

Printing system commands are entered together in a job source library (JSL) file. The JSL file is then compiled as an object file called a job descriptor library (JDL) file that the printing system can read. The printing system then responds to the commands contained in the JDL file and prints the job as it is defined to appear. LCDS does not support color printing.

Metacode

Developed by Xerox, Metacode is a machine code variant of LCDS used to describe text and graphics. Like LCDS, it was developed as a low-level control language for Xerox 9700 and 4000 series printers. It provides greater flexibility than LCDS through its own proprietary metalanguage.

The Metacode language uses hexidecimal character codes to describe the position of data (text and graphics) on a page. This code can also be used for page orientation control and font selection and can enable highlight color support.

Summary

To a large extent, your production environment will govern your choice of variable information languages. Your RIP will determine which VI languages you can support and your data environment will likely dictate which languages you can use (particularly in legacy LCDS, IPDS or Metacode data-stream environments). Whenever you do have a choice, the vast assortment of VI languages available today can make selecting an appropriate language a difficult task.

VI language support varies across different VDP software and RIPs. In creating a VI file, the supported features and composition performance of one VDP software product might be different from another VDP software product. For example, a VIPP project created from XMPie uDirect software will differ from one created by Lytrod Designer software. XMPie uDirect uses InDesign for document design and composition, whereas Lytrod Designer uses its own design environment and packages the application resource files together, and the document is composed at the RIP. This RIP-side composition can offer significant performance benefits over desktop or server-side composition, as the document is composed and interpreted simultaneously. Text, images and data-driven graphics such as pie chart or line graphs are composed by the PostScript interpreter. Desktop or server-side composition software needs to first generate an output file (using page layout software such as InDesign or their own composition engine) before it can be interpreted by the RIP.

The open standards discussion does not really have merit in VDP; if a proprietary file format can offer greater performance and VI feature support than a standards-based format, is the standards-based format better?

Furthermore, VI performance will vary across different RIPs. For example the processing time of a PPML file for one RIP could differ considerably when processed on a RIP by a different vendor, even if the RIPs are running on similar hardware. Different vendors use different approaches to interpreting and rendering variable information, and like with performance-enhancement drugs, "results may vary."

When possible, you should run some benchmark tests on your RIP using selected VDP software products (trial versions are available for most of them) to gauge performance differences among different software, VI languages and application types.

When selecting a VI language, base your choice on your own research and experience and not on market direction or opinions. One popular objection to vendor file formats is that they are regarded as proprietary technologies, not open standards. The open standards discussion does not really have merit in VDP; if a proprietary file format can offer greater performance and VI feature support than a standards-based format, is the standards-based format better?

Another consideration is performance, but this does not need to be the deciding factor. Certain VI formats can be interpreted by RIPs at a rate of several thousands of pages per minute, but can your printer print at thousands of pages per minute? The key to performance is ensuring that you can run your print engine at rated-speed.

Choosing a VI language is a little like buying a new pair of shoes; there are many varieties, each with its own style and benefits, and one size does not fit all. You need to find the pair that fits you. **TSR**

* source: "*Variable Data Printing 2006: Growth and Changes in the Marketplace*" TrendWatch Report

Eliot Harper is workflow marketing manager at Fuji Xerox Australia. He can be contacted at eliot.harper@aus.fujixerox.com.