CPDS, Interpress and PostScript - A Comparison of Page Description Languages

Klaus Daube
Oerlikon Bührle Rechenzentrum AG, Zürich Switzerland (S216)

Abstract

Recent development of cheap highly functional laser printers and raster image processors **stimulated interest in languages for interfacing to these devices. This paper compares** CPDS **from IBM, Inter-press from Xerox and PostScript from Adobe. Areas covered are design** goal, **functionality, coding (data representation) and font metrics.**

Introduction

There have always been languages for communicating page information to typesetters and **printers. Until recently the capabilities of computer output have been very limited, so their input languages have been simple ASCII formats modified by escape codes. Typesetters on** the **other hand have quite complex input languages with lots of commands.**

With the recent advent of laser printers and raster typesetters the issue of PDL's (Page **De**-scription Languages) has received much attention, and a number of languages have been invented. Since document contents can be defined at several layers, they seem to break most **naturally in three layers:**

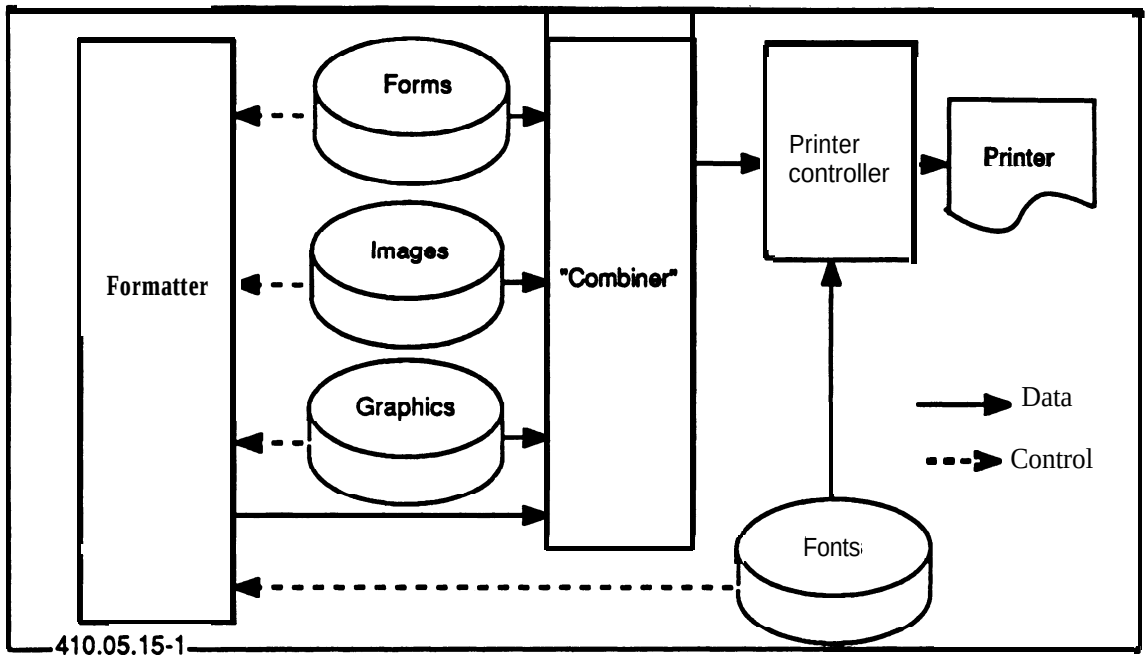
- **On** the highest level we have abstract languages, which are often described as "declarative" like SGML (Standard GML), GML (Generalized Markup Language) or TBL (Table description language in the UNIX environment).
- On the medium layer are the procedural languages, which tend to have a command **and parameter structure: Script,** TROFF etc.
- The lowest layer contains the slave languages. All the abstract information has disappeared. All decisions like line breaking or footnote placing have been made. Only general coordinate transformations may be done. For this level the Virtual Device Interface of TEX is an example.

The most appropriate layer for PDL's is the slave layer with maximum provision for interfacing. However, the boundaries between these layers are somewhat floating. So some PDL's are **procedural also.**

This text freely uses trademarks and product names without special notice.

Processing the Information

The picture below shows the flow of information from the formatter to the printer. **To form** the page of a "compound document" various sources of information are necessary.



Donald Knuth desired for his series typography to re-

(VDI) interface device independent TEX device in 1984. This PDL was very rudimentary.

In **1981** the Imagen Corporation began deliveries of a laser printer based on the Canon LBP-10 with 240 dots/inch resolution. Their PDL imPRESS was heavily influenced by TEX's page model. At first the image processor was a modified SUN workstation [2]

In the meantime a device independent "Press format" was developed at Xerox PARC by Sproull and Newman. This language did not become available in marketed products, but the definitions had been distributed widely enough, so, that by 1982 researchers at the EPFL at Lausanne could present their own implementation. A Motorola 68000 microprocessor was used in the controller for a Canon LBP-10 printing machine [12]

In 1982 Sproull, Warnock and others created a new page description language called Interpress, which evolved from both "Press format" and JAM (device independent language for graphics devices of Evans and Sutherland). This language was not available outside Xerox earlyer than 1984, when it was made public. The full language with three levels of functionality is not yet standardized or published.

Wamock and Geschke left Xerox to form Adobe Systems, which designed and market the PostScript language, which was heavily influenced by Interpress. The first PostScript printer was produced by Apple Corporation (LaserWriter).

IBM's page description language CPDS is defined as input to a specific program, the Print Service Facility, which is used as a "driver" for the 3800 family of APA (all points addressable) printers. It became available after the 3800-3 [4] in 1984.

This year (1986) Imagen corporation launched a new PDL called DDL for Document Description Language. Hewlett Packard has announced an agreement to adopt this language [1]

Several other manufacturers of laser printer controllers developed proprietary page description languages. QMS (Quality Micro Systems) established QUIC for their Magnum controllers. It is clearly a lowest level device, merely a special form of "escape sequences" with the ability to use forms, build loops, set half-tone patterns and blow up pixel patterns. Now they also offer PostScript versions of their printer controllers.

Xerox also uses a form of escape sequences in their 2700 and 4045 printers. Interpress is now available for Xerox' whole product line of printers.

Procedural versus nonprocedural

Non procedural languages can be generated more easyly by an application program.

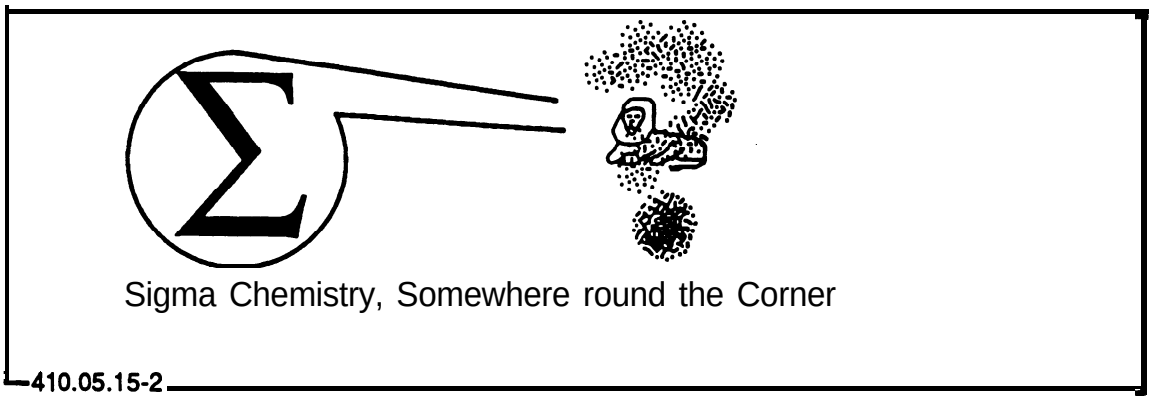
Procedural descriptions are more compact for images containing regularities. Consider a procedural description of a grid compared to a static description. Procedural descriptions can be used for abstraction and modular design. Both PostScript and Interpress files start with a prologue defining verbs (actions, functions, procedures) which are used later on. Hence application programs like one to generate business graphics can define appropriate procedures to shorten the image description, e.g. a procedure to generate pies in a chart.

The real power of a procedural PDL comes from the ability to execute code conditionally, to call functions and perform arbitrary computations. Redefinition of built-in functions is not necessary but often convenient.

As with procedural languages in general, there is no provision against abuse of the programming features. Hence an interpreter must deal with things like infinite loops.

Example

The following picture inside the box consists of text of two sizes (c, Sigma . ..) a small graph ic (the retort) and a small bit map (the smoke). It is used to show differences in the PDL's discussed here.



CPDS is designed to be generated by programs only. PostScript as well as Interpress (character form only) can be written by a humans like any programming language. But they also are intended to be generated by programs. This is the normal case.

CPDS

CPDS is short for Composed Page Data Stream . This datastream consists of the following entities:

- The print data set (file), which contains data to be printed and which is created by an application program (mostly a formatter).
- Resource objects, which are collections of printing instructions and data to be printed. They are stored on libraries and referenced in the print data set. An example is a page segment, which can be a combination of image and text. Resources are built by specialized utility programs like Print Management Facility or PPFA **[14].**

The format of the print data set determines the type of the data stream. There are three types:

- Composed-text data stream (we will concentrate on this only).
- Line format data stream. This is very convenient to enhance existing line printer applications.
- Mixed format data stream.

This composed data stream is processed by Print **Service** Facility which in turn generates the printer commands. The printer itself has limited capabilities concerning storage of fonts etc.

At the most basic level, CPDS is used to direct the placement of text on a page. This text with imbedded placement information is called composed text. CPDS as a "language" belongs to the family of IBM "architectures" which are based on "structured fields". These are definitions for functions and services. They all are of the form ***length, contents.*** **Hence** the contents may use all bit combinations of a byte. These definitions may be nested. Since the length declaration for the outermost "envelope" can only be determined after closing all the nestings, a pretty large storage will be necessary. This may be circumvented by limiting the lengthes of some elements. That is e.g. done by the VM3812 product [5].

CPDS is clearly a nonprocedural static description of the contents of a printed page. Compared with the other PDL's presented here CPDS' merely aims at an enhancement of "common" line printer applications. Various mechanisms for predefined elements are more important here than a graphics approach:

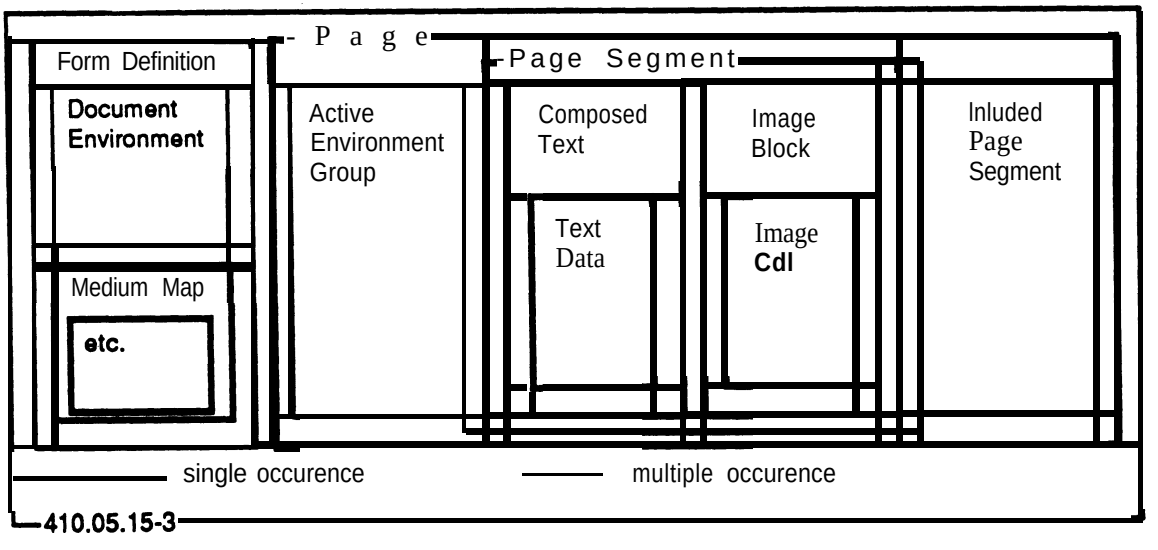
- overlays, which may consist of page segments, text, rules and images
- images
- text, which may be in line printer format
- page segments, which may consist of both text and images

CPDS is not really device independent, since object properties and font definitions depend on the printer type. The definitions within the datastream are completed or superseded by information from job control.

Functions

Related structured fields are arranged in a hierarchical fashion to represent CPDS objects. External objects such as a document or a page segment, are made up of structured fields and internal objects, such as a composed-text. These are made up of structured fields and-lesser objects etc. At the lowest level, objects are made up of a sequence of structured fields.

Comments may be inserted in the form of NOP (no operation) fields.



A document is built up by (composed text) pages. Every **page consists at** least of an "active environment group" and may contain composed text blocks, included page segments and image **blocks. The active environment group specifies properties of the page:**

- **Size of the page or overlay (if used in that definition)**
- **Control** for composed text blocks (size etc.)
- Page segments to be loaded into the printer (e.g. logos)
- Coded fonts to be loaded into the printer

Besides other special things a "document environment group" may be defined for a document, which will specify:

- Identification of overlays
- Identification of text strings to be suppressed
- Positioning of the page on the form (the paper)
- Medium descriptor, e.g. to specify measurement units

Object properties (size, coordinates) are measured in pels (1/240"). Thus **CPDL** is not **device independent.**

Text Objects

A composed text block can be used within an overlay (an object to be superimposed on one or more pages), a page segment (an object, which can be placed anywhere on one or more pages) or within a page itself.

Such a block specifies the text by special controls and the text itself. The controls define such things like

- Origin of the text block on the page - this is printer specific.
- Text orientation is defined by inline direction and baseline direction whereas font orientation is defined by inline direction and character rotation (relatively to baseline). This specification depends on the printer. Only orientations of n x 90° are possible.
- **A** string that can be marked for possible suppression (used in copy modifications)
- Rules, both horizontal and vertical.
- Font references. **For simple cases** this information may also come from the **JCL.**

Graphic Objects

Besides the rules mentioned in the text objects, CPDS does not support any graphical objects. All graphics must be transformed to images.

Image Objects

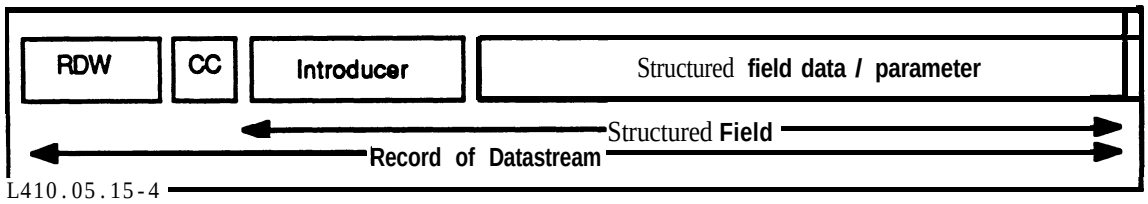
Image blocks may occur within a composed page, a page segment or an overlay. It specifies the contents of an image and its placement in a page segment or overlay. An image can be simple (one image block describing the total image) or complex. A complex image is built up of image cells. One reason for this is avoiding of blank space (savings both in storage and transmission).

Image objects can be overlayed (ORed) with text. Hence **in our example the retort with the Σ** may be two **images.**

Data Representation

Data records, other than line data, are contained in structured fields. Some **of** them **contain both the data to be printed and formatting instructions.**

Since these fields are of variable length, they are part of variable **length records, which start** with an RDW (record descriptor word), followed by a carriage control x'5a'.



A **structured field** can be up to 32'751 bytes long. It must have an introducer and usually has parameters (control information and/or data to be printed. It also can be padded.

Some structured fields have printer dependent versions. The introducer is built up of the following elements:

length	2 bytes giving the length **of** the structured field (8...32'751)
identifier	3 bytes, the first of which is always x' D3 '. These are not mnemonics.
flags	1 byte: extension indicator, padding indicator
sequence number	2 bytes: numbers the structured field to help find them in the data stream. Some rules should be observed.
extension length	*1* byte
extension	up to 254 bytes of user defined data can be included.

Miscellaneous

Error Handling

CPDS does not contain any mechanisms for error handling. Since the program interpreting the data stream is on the host, error handling is done there. This program (Printer Service Facility or **VM** service machine) also handles the error stati of the attached printer. There is no spooling system between these programs and the printer.

Font Specifics

Codepages associate binary code points (from the text using the code) with the character in a font. They also define which of the 256 possible 8-bit codes are valid and give the name of the **character (itself often a** 1 **byte code)** that will be printed. Codepages are necessary for environments with varying structure of the fonts. Coded font objects may specify single byte fonts or double byte fonts.

Fonts for CPDS are some kipd of bitmap. The font definitions for the 3800-3 (or 8) and 3820 are not identical. There are differences both in structure and pattern maps. Also the orientation of the font is different. So one must maintain different font libraries for these two types of printers.

Printing Instructions

CPDS can contain information about special operations to he done at the printer' like duplex/simplex printing or the positioning of the page on the paper. The number of copies can be specified for each page or the whole document. Also modifications to the copies can be specified. Some definitions may be done in several ways, so a precedence is defined for them.

Sample Definitions

The structure of the CPDS file to produce the mentioned example is shown here by indentation rather than nested boxes. The mnemonics define the type of "structured field". The length is not shown explicitly. Meta comments are enclosed in { }. xxxx shows the presence of data/parameters of the structured fields.

```
BDT                 (begin document)
  BDG                 (begin document environment group}
    PGP xxx             (page position)
    MDD xxx             (medium descriptor: units of measure)
  EDG                 (end document environment group}
  BCF                 (begin coded font}
    CFC xxx             (coded font control: constant data)
    CFI xxxxxxxxxx      (coded font index: name codepage, name font)
  ECF                 (end coded font}
  BPG                 (begin page}
    BAG                 (begin active environment group}
      MCF xxxxx           (map coded font}
      PGD xxx             (page descriptor: size of page}
      CTC xxx             (composed text control: constant data}
      CTD xxx             (composed text descriptor: size of text block)
    EAG                 (end active environment group)
    BCT                 (begin text block)
      CTX xxxxxxxxxx      (composed text data}
    ECT                 (end text block}
    BIM                 (begin image block}
      IOC xxx             (image output control: origin, scale factor,
      IID xxx             (image input descriptor: image size, ..}
      ICP xxx             (image cell position}
      IRD xxxxxxxxxxx     (image raster data: the retort)
      ICP xxx             (image cell position)
      IRD xxxxxxxxxxx     (image raster data: the smoke}
    EIM                 (end image block}
  EPG                 {end page}
EDT                 (end document}
```

Software and Hardware

Examles of software generating CPDS files:

DCF-III	Document Composition Facility, version 3.
PMF	Print Management Facility, a utility to generate forms and page segments.
PPFA	Page Printer Formatting Aid [14]
GDDM	Graphics Data Display Manager. Used in applications like ICU (Interactive Chart Utility) to generate images.

As far as I know there is no hardware supporting CPDS. The datastream is converted to printer commands by Print Service Facility [4] or VM3812 [5]. The electro erosion printer (IBM 4250) is not supported by PSF.

Inerpress

This PDL is an outgrow of the Xerox "Star" project. **The workstation philosophy developed** at Xerox PARC **called for "intelligent printing".** Interpress is a major remake of the Press format [12], which was never implemented in a commercial product. The weaknesses of **Press** format especially in the font area were overcome **by new** concepts. Until 1984 all aspects of Inter-press were proprietary. Today information **is only available from Xerox Corporation directly.**

Interpress is designed for high performance and efficiency for **the specific** purpose of electronics printing. It is not indended to be a general **purpose editable format, a composition language, or to be created** by people. **The Interpress model assumes a user creates Interpress masters (the file) via any number of document editing and composition systems.**

Interpress is a procedural language defined in terms of keywords (like Algol). Compared to PostScript it is defined more abstractly, since the encoding is defined "generic". For the human reader a form using a stream of characters is at hand. This is intended for debugging purpose only (developing of formatters etc.)

Both Interpress and PostScript don't use much syntax. Both are token-oriented languages. Each "token" is "executed" as soon as it is identified. Both languages assume that a printer (controller) contains an interpreter for the executable language. A page is thus printed as a side effect of executing the page description. Whereas CPDS is static, Interpress and PostScript are dynamic.

The function of full Interpress is divided into three levels (sets):

- **Commercial Set: text, rules, forms, bit-maps, 90 rotation. This set is comparable to CPDS.**
- **Publication Set: curves, vectors, dashed lines, rectangular clipping, functional colour (shades)**
- **Professional Graphics Set: gray scale pixels, arbitrary rotation and clipping, process colour**

Only this full Interpress is comparable to PostScript, but not yet defined in a standard.

Xerox tries to launch Interpress as a PDL standard for ISO/OSI networks, especially **due** to **its set of printing control.**

With an "Inter-press Transfer **Architecture" Xerox will set up** a bridge between **IBM's Advanced Function Printing and Interpress printing. The role of "Xerox Document Presentation Service" is similar to that of PSF** [4] and it may coexist with it.

Further discussion is based on Interpress version 2.1 [13].

Functions

Interpress is defined in terms **of byte codes which can be seen as instruction codes, which were** compiled from the character form (mnemonics). An Interpress file **has a static structure** (some call it "lexical structure"). It is defined by a sequence of "bodies". Each body is a se**quence of operators and parameters.**

The Interpress file is called master and consists of an arbitrary number of bodies linked to**gether** in a skeleton (a syntactic construct).

BEGIN **(preamble)** (page body 1) {page body 2) * i (page body_n END

The preamble may be preceded by a header with printer instructions. **Page bodies contain re**dundant information, so that pages become independent. These pages can be **printed independently (e.g. in any order).**

Storage in Inter-press is accessed by address using subscripts **(index) into the procedures** frame. "Variable names" are reduced to memory locations. So an Interpress file looks like ob**ject code. Of** course this approach is efficient.

Bodies can be translated into operators thus featuring programmability. Operators working on bodies are prefix, whereas operators working on tokens are postfix.

All objects can be coloured (shaded), since they are all "**masks**" **in the "inking"** process. By **default the properties of** objects **(size, coordinates) are measured in meters. Of** course this can be adjusted to the purpose of the master.

Text Objects

No character set standard is needed, but if existent, it can be accomodated (e.g. **ISO** 6937). **Xerox' development in this** area covers both character codes and rendering codes (e.g. for ligatures). Also codepages for mapping character codes can be set up.

If Text objects try to use undefined fonts, a fall back mechanism allows use of default fonts. Spacing may be corrected in these cases. Transformations (scaling, rotation, slanting, texturizing) may be applied depending to the implemented level of Interpress.

Graphic Objects

- Strokes are masks of uniform width (line or rule). Ending and intermediate points may use various shapes (round, butt or square). A trajectory may use an arbitrary number of points.
- Filled outlines generate masks by filling the region "inside" a closed trajectory. An outline is formed from one or more closed trajectories. To determine holes the "winding number rule" is used.

Intferpress 2.1[13] contains no operations for creating curved strokes. Curves can be approximated only using short strokes.

Image Objects

Image objects may be defined in several encoding schemes, including **CCITT-4** and canonical (plain bitmap). The decompression operators are not defined in the Interpress standard. Instead there is an operator to retrieve it from the environment.

```
xpixels ypixels 1 1 1 m
[vector of compressed data] [name of algorithm] FINDDECOMPRESSOR DO
MAKEPIXELARRAY
```

The two-dimensional array of pixels can be transformed in any way by the transformation machinery of Inter-press. So any scanning order can be handled.

Data Representation

The master is encoded by a header which identifies the encoding, followed by a sequence of tokens. Each token corresponds to:

- A single Interpress literal (not a body). Each such literal can be encoded by a single token.
- One of the symbols BEGIN END PAGEINSTRUCTIONS { }
- An encoding-notation, which stands for some sequence of Interpress literals.

The tokens appear in the same order as the corresponding literals or symbols, except that a body operator token precedes its body. The tokens are of different sizes; each one is a sequence of bytes.

Tokens use one of 5 formats (short/long operation, short number, short/long sequence). For example a primitive operator or symbol is assigned an integer **0.3191** called its ***encoding*** value. It is represented by a two byte long operation token. Strings or vectors are encoded by ***simple, runlength*** or ***extended*** mode.

These various encoding schemes provide for very compact data representation. All encoding schemes use 8 bits of a byte. The value x **' FF'** is heavily used as an "escape" function. There are utilities available to convert an encoded master to a human readable form and vice versa.

Miscellaneous

Error Handling

As Interpress masters are printed, various errors may be encountered. The implementation of the **Interpress** interpretation specifies what should happen. The first few errors should be reported in a status sheet at the end of printing the job. Errors are classified according to their severity.

There are marks on the stack where recovery from errors may start. Errors cause **differnt** types of marks.

Font Specifics

Interpress can work with any character code, but preferable with the **Xerox character code standard** which uses two bytes for every alphabet. Compactness is **achieved by encoding schemes. The standard also provides for** extension of the code (more than two bytes needed).

For each symbol in a font there are properties defined (center coordinates, left/ right extent, super/subscript location, x-height etc.). Some of these properties are vectors (kerning is based on various successor characters, ligature substitution also).

An easy property vector suggests symbol sizes and orientations that will be printed with **greatest** fidelity. Also a mechanism to correct spacing is provided, since width differences can arise when the imager can only approximate the font requested by the master.

A character ***operator*** **makes a graphic image of a character. This operator can be defined by basic operations (computation, line, spline, area fill, bit-map etc.). Symbols** may be **nonspacing (flying accents).**

Colour

Inter-press sets the current colour to any shade of gray with the SETGRAY operator. For those printers that can obtain colours other than grays, a FINDCOLOR operator can set any desired colour like "blue-green" or "light brown". Also arbitrary textures may be set using pixel arrays throughout the page.

Priority

When more then one colour (shade) is used on a page and masks of two **different colours** overlap, there is a possible ambiguity about the visibility of either. Each **object** may be **assumed to** have a numeric priority. When two objects overlap, the one with greatest **priority is visible.**

Printer Instructions

Since Interpress is intended to work in a network it offers a rich set of instructions for a printer or server. Interpress is designed for machine to machine communication. So these instructions enable an up-front determination of the ability of a given printer to print a document. Printer instructions are located in the first (optional) ***body:***

- information about the document (name, creation date/time, creator, sender, receiver, accounting, **job** priority, password etc.)
- name of functional set to be used in this document (assure the printer can print the document)
- environment (masters can be routed within a network to a printer with the needed environment like colour printing)
- finishing, stacking, simplex/duplex printing, image shift, medium description (paper stack).

Some of these instructions may conflict with a transport mechanism like DIA (DIA (Document Interchange Architecture).

Sample Definitions

The human readable form of Interpress for our example is not very different to the PostScript form. The following is a "pencil and paper" unchecked version. Comments are enclosed in --. Unit of measure is meter (default).

```
BEGIN
{
  [Xerox, xc82-0-0, Helvetica] FINDFONT
  0.004233 SCALE MODIFYFONT 0 FSET  -- 12 point Helvetica          --
  [Xerox, xc82-0-0, Symbol] FINDFONT
  0.0127 SCALE MODIFYFONT 1 FSET    -- 36 point Symbol font        --
  0 SETFONT                         -- first use Helvetica         --
  0.015 0.009 SETXY                 -- position the text           --
  <Sigma Chemistry, Somewhere round the Corner> SHOW
  1 SETFONT                         -- next use Symbol font        --
  0.022 0.019 SETXY                 -- position the text           --
  <S SHOW

  -- --------- draw the retort ---------------------------------  --
  0.0006 15 ISET                    -- stroke width 0.6mm          --
  0.074 0.033 0.043 0.035 MASKVECTOR -- lower line of nozzle       --
  0.074 0.038 0.030 0.044 MASKVECTOR -- upper line of nozzle       --

  -- I dont's show here a large number of vectors creating the arc --

  -- --------- "produce" smoke ---------------------------------  --
  64     95     --     xpixels,          ypixels                  --
  111    --    no                 transformation                  --
  0.0000847    SCALE   --  pixel   size   1/300   inch            --
  0.080 0.016 TRANSLATE CONCAT -- coordinates of image            --

  [  0  , 0 , 0 , 0  ,o  , 0 , 0  IO  ,
     0  , 0 , 1  , 0  , 0 , 128, 0 ,  0  ,
     0  , 0 , 0  ,    16,    0  , 8, 0 , 0 ,
     0 , 0,  32,   128,   16,   64, 0 , 0 ,
     --   and so on, a lot of numbers    --
     0    , 0  , 0 , 0 ,  32,   0 , 0 ,  0 ,
     0   , 0 , 0 , 0 , 0 ,  0 , 0 ,  0   1
  [unpacks] FINDDECOMPRESSOR DO
  MAKEPIXELARRY
  MASKPIXEL
}
END
```

Software and Hardware

Xerox literature mentions some software generating Interpress files:

- IBM **Advanced Function Printing to Interpress**
- Unix Interpress toolkit
- TROFF, TEX, SCRIBE
- Transpress **& Micropress**
- **GEM from Digital Research**

It is not to my knowledge which subset of Interpress is executed by the printers mentioned below. Currently no full implementation of Interpress exists.

Printer/controller	Resolution (dots/inch)	Page size	Marking engine
Telecopier 495-l	200	approx. A4	**Xerox**
3700 Laser Printing System	300	A4, A3	**Xerox**
4045 Professional Graphics	300	A4, A3	**Xerox**
8700 and 9700Laser Printing System	300	A4	**Xerox**

Among those companies which announced Interpress support are **Allied-Linotype (Linotype** imagesetters), Compugraphic (imagesetters), Dataproducts, **DEC, Interleaf and Siemens.**

PostScript

PostScript claims to be a simple interpretive programming language with powerful graphics primitives. The primary application of PostScript is to describe the appearance of text, images and graphic material on printed pages. PostScript may be **sayed** to admit no distinction between text and graphics.

Normally, PostScript source code is generated by word processing programs, CAD programs and other composition programs. Programmers may write PostScript directly when setting up applications. So PostScript also defines a standard, extensible print **file** format.

A program that generates a PostScript source file neeed not be complicated or maintain a large amount of state information. A program can stream PostScript source incrementally to a file. This attribute of the language allows even small machines to generate complex PostScript sources. Compared to Interpress the interpretation of PostScript needs more resources. In the long run, Adobe is counting on increased computer processing power. They decided early that it would **emphasize** full functionality and the highest possible output quality, even if this means taking somewhat of a "hit" on performance.

Adobe also feels that it is important that any PostScript printer be able to print any PostScript page - no matter how complex. The argument is that the user would rather have the page print slowly than not to be able print it at all.

Because PostScript is a programming language, it its possible to write very inefficient or very efficient PostScript output drivers. So substantial improvements to the performance of (current) PostScript raster image processors has to be made.

Functions

The PostScript imaging model is a unified view of two-dimensional graphics. An image is built up by placing ink on a page in selected areas. The ink may be in the form of letter shapes, general filled shapes, lines, or halftone representations of photographs. The ink itself may be in colour or in black, white or any shade of gray. Any of these elements may be cropped to within any shape as they are placed onto the page.

PostScript has no fixed lexical structure, it is just a stream of tokens to be processed by the interpreter. It prints a page whenever the **showpage** operator is executed. If this operator is located in a loop of 10 cycles then 10 pages are produced.

The imaging operators are **fill** (mark an area on paper), stroke to produce lines, image to paint a halftone gray-scale scanned image onto the current page, and show to paint character shapes.

Path operators set up polygonial or curved pathes, which act on the imaging operators. Generally speaking a PostScript program contain many instances of the following pattern:

- build a path using path operators
- set any **implicit arguments**
- perform an imaging operation

Beside that PostScript maintains a ***current dipping path*** that outlines the area of the current page that may be imaged upon. Initially this is the total imageable area of the page. This area can be modified by a **cl**ip operator.

PostScript provides a much richer set of general purpose processing capablities than Interpress. It provides a programmer with a rich set of arithmetic, control, looping, string processing, conversion between object types, signalling etc. operations. Use of these capabilities can impose heavy processing loads on the printer.

PostScript supports colour images. On a seminar at US (8/86) Adobe has shown the first colour picture imaged on the Linotronic 300 typesetter. It was a 120-line-screen, four-colour picture and colour pattern with unconventional screen angles to avoid moire patterns.

A PostScript program is able to generate an array of PostScript operators and have it executed just like any other PostScript operator.

So called ***late binding*** of shapes permits a defined operator to be used for varying effects in varying contexts. This virtue can be used to isolate pages by "enveloping" their definitions. So a PostScript file may become an inner part of another file. Combining pages on a sheet of paper is common use of this technique.

By default object properties are measured in 1/72 inch (approx a printers point). Of course this can be adjusted to any other unit (see example).

Text Objects

If Text **objects try to use** undefined fonts, a fall back mechanism allows use of default fonts. Spacing may be corrected in these cases, as long as words or characters are not positioned explicitely on the page.

All transformations (scaling, rotation, slanting, texturizing) may be applied to text elements, since text is only a special form of graphic.

PostScript has mechanisms to "format" lines of text (justify). Inherent restrictions of these mechanisms can be overcome by explicitly placing every word or symbol on the page.

Graphic Objects

Besides straight lines and polygons PostScript supports circles and arcs as well as Bezier curves (splines) and conic curves. Pathes (trajectories) may be constructed using all of these. These elements can be represented both in solid or dashed, they can be viewed through masks, filled with gray scale. Ending and joining points can be of various shapes (more kinds than in Interpress).

Filled areas are created by filling a closed path of any complexity. Filling may be by ***winding*** rule or by ***even-odd*** rule.

Image Objects

PostScript has more powerful facilities for halftones and images than any of the other languages described here. These include multiple bits per screen ("printer's screen"), sophisticated control of the screen shape and size, colour and gray scale. Not every PostScript printer is guaranteed to support all of these features, but even the low-end LaserWriter does a quite good job.

Bit images depend on the resolution at which they are digitized. However the power of PostScript allows arbitrary transformations. In our example the 'smoke' is a scaled bitmap.

Data Representation

PostScript is entirely human readable. All input comes from the printable 'I-bit ASCII character set plus the **newline** marker (LF) and can probably be transmitted by any communication medium. Characters outside this range must be encoded as hexadecimal or octal constants, or a few special **constans** escaped with a backslash in the style of the C language:

```
(foo \t bar \n)print
```

\t stands for a TAB, **\n** for a LF. As seen in this example, strings of characters are enclosed in parantheses. A parenthese itself is represented by **\)** and \ is represented by ****.

PostScript is defined in terms of character sequences. The character tokens are separated by white space characters. A comment facility provides for preprocessing and debugging.

Miscellaneous

Error Handling

When a PostScript error occurs, an error operator is executed. There is a set of built-in error operators provided as part of PostScript. A PostScript user can change this error handling by changing the dictionary entry for the relevant error operator. Depending on the relative position of that redefinition with respect to save and restore the redefinition will have a certain lifetime.

Fon t Specifics

PostScript's default font treatment is its most controversial feature. **PostScript interpreters** rasterize fonts on demand from outlines known to the interpreter. But **normally a printer has oftenly** used fonts in some sizes ready in read only memory. There are **two levels of compres**sion for the bitmaps.

For the latin alphabet the fonts hold a common set of graphics (characters, ligatures, **special** symbols). Within PostScript all graphics have a unique name, so codepages **can be set up. Al**so existing shapes within fonts can be selectively replaced.

Printer Instructions

PostScript does not define printer instructions in a standard way. However an implementation may introduce special operators and values in the status dictionary for that purpose. For example the LaserWriter supports multiple copies, select paper bin etc.

PostScript comments according to file structuring conventions can be used for preprocessing the PostScript file. This can serve for printing of selected pages, combining pages on a sheet etc.

Sample Definitions

The following PostScript file contains computations which normally would be performed by a formatter. Comments are introduced by the **%** sign. It also tries to conform to the **'PostScript** File Structuring Conventions' described in Appendix J of **[7]**. The comments starting with **%%** (and **%!** for the first line) have been inserted for that purpose.

```
%!PS-Adobe-1.0
%%Title: PostScript drawing example: Retortl.psc
%%DocumentFonts: Symbol Helvetica
%%Creator: A.Hoppler
%%CreationDate: 29-Sep-86
%%Pages : 1
%%For: OBRZ/DTA
%%EndComments
```

```
%------------------------------------------------ PostScript Prolog.

% Define the Jobname to the Apple LaserWriter
  statusdict /jobname (PostScript Example) put

% Define scale factors
  /inch { 72 mul } def
  /mm { 2.83465 mul } def

%%EndProlog

%%Page i 1
% 'i' is document-defined page no, 1 is sequence number

% global offset of lower left corner of picture
  1.2 inch 2 inch translate

% write Text
  /Helvetica findfont 12 scalefont setfont          % Helvetica-12
  15 mm 9 mm moveto                                 % lower left corner
  (Sigma Chemistry, Somewhere round the Corner) show

  /Symbol findfont 1.2 inch scalefont setfont       % large 'Symbol'
  30 mm 20 mm moveto                                % lower center
  (S)                                               % greek sigma
  dup stringwidth                                   % char width
  -0.5 mul exch -0.5 mul exch                       % calc. half
  rmoveto                                           % adjust
  show                                              % output character

% define 'vector length' operator
  /vlen {
    dup mul                 % calculate y**2
    exch dup mu1            % calculate x**2
    add                     % add
    sqrt                    % sqrt (x**2+y**2)
  } def
```

```
% draw retort
  0.6 mm setlinewidth          % line width (of course)
  1 setlinecap                 % round line caps
  1 setlinejoin                % joins ditto
  newpath                      % start a new 'path'
  74 mm 33 mm moveto           % move to first point of path
  43 mm 35 mm lineto           % line to 2nd point
  /savedmatrix matrix currentmatrix def % save coord. system
%   translate the coordinate system so that center of arc will be
%   at [0,0]. This simplifies the subsequent calculations.
    30 mm 30 mm translate      % center
      0 0                      % center (transformed)
      currentpoint vlen        % radius
      currentpoint exch atan   % angle of start point
      90                       % angle of endpoint
    arcn                       % arc (clockwise)
  savedmatrix setmatrix        % get back coordinate system
  73 mm 37 mm lineto           % last line segment

  stroke                       % outputs the entire path built above

% draw bitmap
  /imgstr 8 string def         % define work string
  gsave                        % save coord. system, etc.
%   transform coord. system so unit square will be mapped to
%   desired output aerea.
    80 mm 16 mm translate
    18 mm 32 mm scale
    /filemask {                % define proc to transfer bitmap from file
      {                        % reader proc
        currentfile imgstr
        readhexstring pop
      } imagemask
    } def

      64 95                    % scanlength, scanlines
      true                     % invert
      [64 0 0 -95 0 95]        % matrix
```

```
%   generate image. The following hex data may contain any
%   number of spaces and newlines, but NO commments. Also it
%   needen't be arranged corresponding to the rasterlines.

    filemask

      0000 0000 0000 0000
      0000 0100 0080 0000
      0000 0010 0008 0000
      0000 2080 1040 0000
      .... etc  ......... giving a total of 95 scanlines
      0000 0067 7200 0000
      0000 0012 4C40 0000
      0000 0008 8100 0000
      0000 0002 8800 0000
      0000 0000 2000 0000
      0000 0000 0000 0000

  grestore

  showpage                          % output the page
%
%%Trailer
%-------------------------------------------------PostScript Epilog

% (no epilog commands)
```

Software and Hardware

Best **known software to produce PostScript files are Macintosh applications like MS-Word** or **MacDraw.** In fact they produce an intermediate file in 'Quick Draw' which in turn is converted to PostScript by **Mac's** basic services. **A** font editor "Fontographer" produces PostScript directly. **As for Interpress there are also numerous announcements** to **suppert** PostScript, e.g.:

WIPS +	**DEC's** wokstation and host based document processing software
Interleaf TPS	Technical Publishing Software from Interleaf **corp**.
Interleaf WPS	Workstation Publishing Software from Interleaf **corp**. running on several workstations.
SUSI	Text **formatter of OBRZ**

An encreasing number of hardware supporting PostScript is available. Since PostScript is a language without subsets defined, all of the mentioned printers can print very complex pages.

Printer/controller	Resolution (dots/inch)	Page size	Marking engine
Apple Laserwriter	300	approx. A4	Canon
Dataproducts LZR-2660	300	approx. A4	Toshiba
Dataproducts LZR-2665	300	up to A3	Toshiba
Linotronic 100	up to 1448	297 × 575 mm	photo typesetter
Linotronic 300	up to 2540	305 × 655 mm	photo typesetter
Mergenthaler P101	up to 1270	???	photo typesetter
Mergenthaler P300	up to 2540	???	photo typesetter
Ricoh 4020	300	approx. A4	Ricoh
QMS PS 800	300	approx. A4	Canon
QMS PS 1200	300	approx. A4	Xerox
QMS PS 2400	300	approx. A4	Xerox

Comparison

Language Aspects

	CPDS	Interpress 2.1	PostScript	**imPRESS**	DDL
Programmability	no	limited	full	limited	full
Representation	binary	binary	ASCII	binary	ASCII, binary
Storage management	no	no	manual	no	automatic
File access	no	limited	**yes**	no	**yes**
Printing instructions	**yes**	**yes**	no	**??**	**yes**
Device independence	partial	**yes**	**yes**	partial	**yes**

Graphical Aspects

Arbitrary transformations	no	**yes**	**yes**	**???**	**yes**
Line	rules only	yes	**yes**	**yes**	**yes**
Area	images only	**yes**	**yes**	**yes**	**yes**
Curve	no	no	**yes**	**yes**	**yes**
Fonts	bitmap	bitmap, outline, stroke	bitmap, outline	bitmap	???
Texturizing mode	????	**transpar-** ent, opaque	opaque	???	transpar- ent, opaque, compliment
Priority important	**n/a**	on **| off**	on	???	on
Sampled objects	**yes**	**yes**	**yes**	???	**yes**
Arbitrary clipping	**n/a**	**rect.** only	yes	**rect.** only	**yes**
Colour	no	**yes**	**yes**	no	**yes**
Bitmap scaling	limited	no	**yes**	no	**yes**
Compressed images	no	**yes**	possible	no	???
Composite objects	**yes**	no	no	no	**yes**
Document layout	no	limited	no	no	**yes**

Performance

Compactness	**yes**	**yes**	no	**yes**	**yes**
Object **cacheing**	**n/a**	fonts only	fonts only	no	full
Independent objects	**pages**	**pages**	not guaran- teed	no	sections

Literature

[1] DDL: A Language to Describe Documents, Imagen Corporation 8/86

[2] imPRESS Programmers Manual version 2.1, Imagen Corporation, August 84

[3] 3800 All Points addressable Printing Technology (1984), IBM publication number SH35-0090-0, ISBN o-9331-04-5

[4] Print Services Facility Data Stream Reference (february 1985), IBM publication number SH35-0073-1

[5] VM3812 - IBM 3812 Pageprinter VM Support, Application Programmers Guide. IBM publication number SH20-6732-0

[6] Heather Brown, From Text Formatter to Printer, Protext I Conference proceedings, Boole Press, 1984

[7] PostScript Language Reference Manual, Adobe Systems Inc., Addison-Wesley, 1985

[8] Robert A. Morris, Page description Languages, An Introduction to Text Processing Systems, Boole Press, 1985

[9] David J. Harris, An Approach to the Design of a Page Description Language, Proceedings of the international conference on "Text Processing and Document Manipulation" at the University of Nottingham, Cambridge University Press 1986, ISBN O-52 1-32592-7

[10] Brian K. Reid, Procedural Page Description Languages; Proceedings of the international conference on "Text Processing and Document Manipulation" at the University of Nottingham, Cambridge University Press 1986, ISBN O-521-32592-7

[11] Brian K. Reid, PostScript and Interpress: a comparison, ARPANET Laserlovers distribution, March 1 1985

[12] Roger D. Hersch, P. Fäh: The formatted document description, rapport interne LAMI 1983, EPFL, Lausanne.

[13] Xerox Corporation, XNS Standards Interpress Electronic Printing Standard, El Segundo, CA, Document number XSIG 048404.

[14] Page Printer Formatting Aid, Users Guide and Reference, IBM publication number G544-3181