Postscript

Postscript is both a programming language and a graphical page description language.

Page Description

Important elements of the Postscript imaging model include the current page, the current path, and the clipping path. There is an extensive set of graphic environment settings that influence how marks are actually made on the page.

The current page is independent of the page on which the image will actually be drawn. The drawing is specified in a user coordinate system with the origin (0,0) at the lower left corner. The units of the user coordinate system represent approximately 1/72 inch (1 point) unless scaled. User coordinates are mapped to device coordinates when the image is drawn.

The current path is a set of points, lines, and curves that defines a path through user coordinate space. Basic path operators include **newpath**, **closepath**, **moveto**, and **rmoveto**. Operators that create potentially visible line segments include **lineto**, **rlineto**, **arc**, **arcn**, **arct**, **arcto**, **curveto**, and **rcurveto**.

Another way to define the current path is with the text operators. The simplest method is to set a current font (**findfont**, **scalefont**, **setfont** or **selectfont**), provide a string defining the characters to display, and then **show** the characters in the given font. The **show** operator causes the characters to be displayed. There are numerous operations that can be done to manipulate the character outlines and use them the way you would any other path.

Basic inking operations can use the path to **stroke** the path or **fill** in the areas that it defines.

The clipping path is another path that defines the boundary of the area that may be drawn upon. It may be set to any path and can be used to create a wide variety of effects.

The operators **gsave** and **grestore** are used to take and retrieve a snapshot of the current graphics state. The graphics state includes the current path and numerous state variables such as gray value, line width, and user coordinate transformation.

Programming Language

Postscript is a stack oriented, interpreted, post-fix language. The operators of the language implement the standard capabilities of most programming languages, and hence Postscript can be used to implement a wide variety of programs. Generally, the programming flexibility of Postscript is used to support its page description role in life.

The language uses stacks in several contexts. The most obvious usage is the operand stack. Any operator that needs arguments gets them from the top of the stack. Any operator that returns results does so by pushing them on the stack. The push and pop operations are generally implicit

in the operator, although the pop operator can be used to clear the top element from the stack. There are several operators to move the top few elements around and change their order.

Postscript data types include numerics (real, integer), booleans, strings, and arrays. Procedures are executable arrays of operands and operators.

Operators

There are many predefined operators in Postscript. Common arithmetic operators that you should be familiar with include **add**, **div**, **mul**, **sub**, **abs**, **cos**, and **sin**. Common stack operators include **dup**, **exch**, **pop**, and **roll**. The comparison operators are **eq**, **ne**, **gt**, **lt**, **ge**, and **le**. The logical operators are **not**, **and**, **or**, **xor**, **true**, and **false**.

There are also some interactive operators for use while debugging. == pops the top of the operand stack and prints it on the screen. **pstack** prints the contents of the entire stack but does not change the stack itself.

Dictionaries

Postscript stores the definitions of variables, operators, and procedures in dictionaries. A dictionary is a hashmap, just like the hashmaps in Java. Each entry has a key and a value. When Postscript encounters a word while interpreting a program it looks up the word in a dictionary and uses the associated definition.

Entries in a dictionary are made with the **def** operator. The syntax is /key value **def**. The value can be any valid Postscript type.

There are always at least three dictionaries defined: user dictionary, global dictionary, and the system dictionary. Program defined symbols go in the current dictionary, which is usually the user dictionary. The system dictionary is read-only and contains the predefined operators. References to the dictionaries are kept on the dictionary stack, with the user dictionary on top. Postscript searches the dictionaries from the top down when looking for a reference.

Arrays

Arrays are 1-dimensional collections of objects. Indexes start at 0. Postscript arrays are similar to Java arrays of Objects in that they can hold any type of reference object, but they also can hold simple objects like integers and reals directly. Arrays can be created by giving the elements of the array in square brackets. For example: [1 2 3] creates an array with three elements. Arrays can also be created with the "*n* **array**" operator which creates an empty array of n elements. The **get** and **put** operators can be used to retrieve and store data in an array.

Virtual Memory

The Postscript environment includes stacks and virtual memory. The operand stack contains simple objects (eg, integers) and references to composite objects (eg, strings, arrays). Virtual memory (VM) is a storage pool for the values of all composite objects.

Local VM with **save**/**restore** pairs is used to encapsulate information whose lifetime conforms to a hierarchical structure like a page. The **save** operator makes a snapshot of local VM. The **restore** operator throws away the current local VM and restores the state from the last save.

Procedures

As described above, symbols are associated with names using the **def** operator. This works for procedures as well as variables. The value of the procedure is the contents of an executable array. The operations to be included in the procedure body are specified between opening and closing curly brackets. To use the procedure, you push the arguments it will need on to the operand stack, then give the name of the procedure. Postscript looks up the definition of the procedure in the dictionary, then executes the body. Procedure calls can be recursive.

Control flow

Executable arrays, specified just as they are for procedure definitions, are the basic building block of the flow control operators in Postscript.

The **if** and **ifelse** operators take a boolean and one (or two) executable arrays as operands. If the boolean is true the first block is executed, if it is false then the second block is executed (for **ifelse**).

The **repeat** operator executes a block a given number of times. The **for** operator takes four values on the stack: *initial increment limit body*, and executes *body* until *increment*ing *initial* causes it to be greater than *limit*. The current loop control value is pushed on the stack before each iteration.

The **loop** and **exit** operators work together. **loop** takes one value on the stack, an executable array. The body of the array is executed until an **exit** statement is encountered.