Stream & File Objects

Streams and Files are objects used for reading and writing text. Streams read and write onto strings. Files read and write onto files used for import and export.

Streams and Files are optimized for text, however, they can also be used to read raw bytes from strings and files.

»
The parent of File is Stream. Remember that all of the messages of stream are available to files.

Reading

s read-byte return the numeric value of the next byte

s read-char return the next character

These read the next byte from the stream and return either its numeric value in the first case, or the character in the second.

s read-field spacers w delimiters d
return the next field
This reads the next field from the stream. The text in the stream is read up to and including the next delimiter, which can be any of the characters in the string d. Before the read text is returned, the delimiter, and any white space at either end of the text (defined by characters in the string w) are removed. The arguments spacers and delimiters default to the strings " " and "\t\r\n" respectively. This defaults to reading tab delimited files nicely. If the delimiters includes both CR and LF ("\r\n"), then all three end-of-line standards are handled automatically (see write-line below).

s read-line return the next line

s read-word return the next word

s read-number spacers w delimiters d

return the next field converted to a number

These are variants of read-field. The first two default the arguments of field to read lines and words respectively. The third message is just like read-field, only that afterward, the value is converted to a number (using the scan message).

s read-string width n delimiters d
return the next string

This is similar to read-field, only the field is defined by the width argument, which is the number of characters in the field. If any of the delimiters are reached, then the string is returned, even if it has too few characters. The delimiters argument defaults to "\r\n".

Writing

s write-byte n
write a byte with numeric value n

s write-char c
write the character c

These write a single byte or a character to the stream.

s write-field t delimiter d
write s followed by d

s write-word t

write s followed by a space
s write-number n using f delimiter d

format n and then write it followed by d

The first message writes the text t to the stream followed by the delimiter character d. The second does the same, with a delimiter of a space. The last message, first formats the number n using the format f (which default to the normal number formatting), and then writes it followed by the delimiter.

s write-line t dos write t followed by a CR-LF sequence

s write-line t mac write t followed by a CR character

s write-line t unix write t followed by a LF character

s write-line t
defaults to dos

These messages write text out followed by one of the three standards for end-of-line. If the text value t is omitted, then just the end-of-line sequence is written.

s write-string t width n max m
write t out with size constraints

This writes out t to the stream. If t is shorter than n, then extra spaces are written out to make it n characters long. If t is longer than m, then

only the first m characters are written. Both width and max default to the size of t.

Control

s.position
get the current position

s.position := n
set the current position

These get or set the current position in the stream. This is the index of the next character from the stream to read. [There is a bug in that files are still zero based].

s.at-end
return true if there are no more characters

s skip n
move the current position by n

The skip value, n can be negative to move backwards.

s.size
return the size of the stream

[There is a bug in that this only works for files.]

Creation

stream over t
return a new stream, reading the string t

new file named n
open a file, create it if needed
new file named n create delete the file if it exists, the create it
new file named n exist generate an error if the file doesn't exist

These create a new file object on the file named n. You do not need to use this message for import and export as the file object is passed to your scripts as an argument.

The name determines where the file is stored in PenPoint's file structure, which is only visible to

programmers. If the file name is just a simple name (doesn't start with a backslash character) then the file will be created inside a directory called FILES in the document directory. If the file name begins with a single backslash, then it names a path on the current volume. If the file name begins with two backslashes, then it names a volume and a full file path.