NAME

rpg – Report Generator

SYNOPSIS

rpg <file>

<file> identifies the file containing the rpg source program to be executed. Data is read from the standard input file and the report is sent to the standard output file.

DESCRIPTION

RPG interprets the commands contained in the input file to produce reports. Input data on the standard input file is expected to be in the form of messages.

DEFINITIONS

Constant - octal, decimal, or hexidecimal integers (octal integers always begin with zero).

Expression - any statement contained within parenthesis. Statements consist of logical and arithmetic operation on constants, and bits (extracted from words or fields).

Field - string of ASCII characters delimited by spaces (040), tabs (011), or newlines (012). Multiple occurrences of spaces or tabs are treated as a single occurrence.

Line - string of ASCII characters terminated with a newline character (012).

Message - string of ASCII characters delimited by an End-of-Text character (003).

Word - a string of any ASCII characters on a single line. A word may include spaces and/or tabs.

RESTRICTIONS

In writing RPG programs it has been found to be convenient to define an administrator program whose function is to decide generally which application RPG programs shall run. The existence of this administrator is maintained through protocol agreement among RPG programmers. There are therefore a few conventions the RPG application programmer should follow. They are:

1. The assignment of the variable name '%' should be restricted to the administrator program. In the expansion environment, '%' is defined to be equal to "/office/<office name>".

2. The ability to read into buffer 4 (possible only with the *t command) should be restricted to the administrator program.

3. In general the assignment of variable names '0' through '9' should be restricted. These variables are defined by the RPG interpreter to equal the input arguments when RPG is invoked. For instance, argument '3' can be referenced as variable '3' in the program.

4. Registers 0 through 32 are currently available for use by the application programmer. However, in the expansion environment, the assignment of registers 20 through 29 is reserved for the administrator program. Also the assignment of registers 30, 31, and 32 is reserved for the RPG interpreter. Registers 30, 31, and 32 may be used to determine the number of characters read, whether the message overflowed the buffer, and the number of lines read, respectively.

5. File descriptors range from 0 through 9. However, file descriptors 0, 1, and 2 are defined as standard input, standard output, and standard error output, respectively.

OPERATORS

The following operators are in order of precedence:

*, /	multiply, integer divide
+, -	add, subtract
>, <, =	arithmetic comparison
&	logical AND
|	logical OR
!, -	unary NOT, unary MINUS

Caution should be exercised when doing a logical AND of commands that return values other than zero and one. Problems can be avoided by following each command with ">0", so that zeroes and ones will be ANDed i.e. (*d0,a,0,0;>0 & *d0,b,1,1;>0) will AND the two *d commands based upon their success in defining the fields rather than ANDing the number of characters in each field which could be different.

A NOT operator performed on a command returning a value greater than one will be zero. If a command returns a zero, a NOT operator will result in a value of one.

MISCELLANEOUS

Labels are single ASCII characters preceded by a colon i.e. :z, :a, :1. Labels are used by the *> and *< commands and may appear anywhere outside a command.

The meaning of special ASCII characters can be escaped in the *p command by preceding them with *. Hence, *; *(*) and ** will allow the ; ( ) and * to be printed, respectively.

COMMANDS

RPG commands are of the form:

```
          *a arg1,arg2,...,argn;
```

where '*' indicates the start of a command and 'a' is an alpha character denoting the command. In the case where there is more than one argument to a command, commas are used to separate them. A semicolon is used to terminate or close a command. If a command requires no arguments the semicolon may be omitted, (*e and *n). Those arguments enclosed in brackets are optional and may be omitted. However, lack of an optional argument results in a default value.

In the following command descriptions, n's denote numeric characters or expressions, and a's denote any alpha character or non-alphanumeric character, except where used as commands (*a, *n). Also, the space character immediately following the command is for clarity only and should not be included during programming.

*a [0],a1,a2,a3;

Concatenate strings - 'a1' is a variable to be defined. It may be identical to either 'a2' or 'a3'. The new variable defined is the concatenated string of 'a2a3'. 'a2' and 'a3' may be either variables previously defined or ASCII strings delimited by double quotes. Any number of strings may be defined as long as the combined total length of all strings defined is less than or equal to 100 characters in length. If the administrator is to define the variable '%' with this command, it should be the first variable so defined. A "1" is returned if the string is successfully concatenated and a "0" is returned if concatenation failed.

*a 1;

Clear user portion of concatenate buffer - in the event that the buffer area allocated to concatenate strings is filled, this command will clear the buffer of all user defined variables. If the variable '%' is used it is left untouched.

*a 2;

Clear entire concatenate buffer - this usage, similar to above will clear the buffer area allocated to concatenate strings. The difference is that the entire buffer is cleared, including the variable '%'. Its use should be restricted to the program administrator.

*a 3,a1;

Get process ID - the variable 'a1' will be defined equal to the character string representation of the current proccess ID. A "1" is returned for success and a "0" is returned for failure.

*a 4,a1;

Unlink file - the file whose full path name is specified by the variable 'a1' is unlinked. A "1" is returned on success and a "0" is returned on failure.

*a 5,a1;

Close file - the file whose full path name is specified by the variable 'a1' is closed. A "1" is returned on success and a "0" is returned on failure.

*a 6,a1,a2,a3,n1,a4;

Error message generation - this command prints an error message on the user's standard error output device (file descriptor 2). 'a1' is a two-character string associated with an error message. One of the following should be used: "?F" error in format, "?D" error in data, "?A" error in action verb, "?I" error in keyword field, "NG" no good, or "NA" not available/applicable. 'a2' is a priority of action string. It should contain two blanks " " except when the error represents some soft-ware detected system problem that may require immediate attention. In this case the string should contain one blank and an asterisk " *" to indicate a minor problem or two asterisks "**" to indicate a major problem. 'a3'is a three-character code (typed in capitol letters) which identifies the feature in which the error is detected. For RPG programs this string usually contains "RPG". 'n1' is a three-character decimal sequential number which identifies the error message. Its range of values is 100 to 999. 'a4' is a message string of fifty characters or less which describes the error. This message should be typed in capitol letters. 'a4' may also be a variable. Note that all arguments must be contained in double quotes. The SCCS Output Manual should be referenced for current error messages which may be appropriate before a new message is created.

*b a1,n1,n2;

Octal bit extraction - 'a1' is a previously defined word or field. This word or field must consist of octal numbers (0, 1, 2, 3, 4, 5, 6, 7). Each octal number represents three binary bits. 'n1' represents the left-most bit to be extracted and 'n2' represents the right-most bit. Bits are counted from the rightmost bit of the predefined word or field to the leftmost, beginning with zero. The bits between 'n1' and 'n2' are returned as a right adjusted binary number. 'n1 minus n2' must be less than 15.

*c aa...a;

Comment - the characters between *c and ; are ignored, allowing the programmer to insert descriptions of the code.

*d n1,a1,n2,n3;

Define field - 'a1' is the variable to be defined. 'n1' is the number of the message buffer into which the message has been previously read. 'n2' is the line on which the field is found and 'n3' is the number of the field. Lines are counted from top to bottom beginning with zero and fields are counted from left to right beginning with zero. The number of characters defined is returned if the define is successful and a "0" if the define failed.

*D n1,a1,n2,n3;

Define field and delete leading zeroes - the properties of this command are identical to those of *d except that the variable will represent the field as though no leading zeroes were present. If the field is all zeroes, then the value of the variable is a single zero, '0'.

*e

Exit - the program is terminated. The *e command does not require a semicolon, but may have one.

*f n1,a1,a2;

Open file to read or write - 'n1' is a file descriptor which is associated with the file 'a2'. 'a2' is a variable containing the full path name of the file, otherwise the full path name of the file is given in double quotes. The file 'a2' is opened as described by 'a1'. If a previous file with this number was opened, it is first closed automatically. If 'a1' is "r" the file is opened to read, if "w" to write. The "we" option also opens the file to write, but an End-of-Text character (003) is placed after every message so the resulting file can be read as a message file. Note: when specifying the argument 'a1', double quotes should not be used. They are used here for clarity only.

*g n1[,0];

Mark current position - the current block and displacement for the file descriptor 'n1' is saved. Only one block and displacement may be saved at any one time. The marked position may be restored as described in '*g n1,1;' below.

*g n1,1;

Restore marked position - the saved block and displacement for the file descriptor 'n1' is restored. 'n1' must have been marked as described in '*g n1[,0];' above.

*g n1,2,a1,a2;

Open file to read or write - 'n1' is a file descriptor which is associated with the file 'a2'. 'a2' is a variable containing the full path name of the file, otherwise the full path name of the file is given in double quotes. The file 'a2' is opened as described by 'a1'. If a previous file with this number was opened, it is first closed automatically. If 'a1' is "r" the file is opened to read, if "w" to write. The "we" option also opens the file to write, but an End-of-Text character (003) is placed after every message so the resulting file can be read as a message file. A "1" is returned if the open is successful and a "0" is returned on failure. Note: when specifying the argument 'a1', double quotes should not be used. They are used here for clarity only.

*h a1,n1,n2;

Hexidecimal bit extraction - 'a1' is a previously defined word or field. This word or field must consist of No. 101 ESS hexidecimal numbers (-, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, M, S, C, T, R). Each No. 101 ESS hexidecimal number represents four binary bits. 'n1' represents the leftmost bit to be extracted and 'n2' represents the rightmost bit. Bits are counted from the rightmost bit of the predefined word or field to the leftmost, beginning with zero. The bits between 'n1' and 'n2' are returned as a right adjusted binary number. 'n1 minus n2' must be less than 15.

*i n1;

Increment register - 'n1' is the name of a register whose value is incremented by one.

*i n1,n2;

Set register - 'n1' is the name of a register whose value is set to the value if 'n2'. If 'n2' is a negative number, it should be inclosed in parenthesis.

*l n1,n2;

Log (output) message buffer - 'n1' is the number of the message buffer whose contents are written to file descriptor 'n2'. Everything is written up to an End-of-Text character (003) or 512 bytes, whichever occurs first. A "1" is returned on success and a "0" on failure. An error will result if 'n2' is not open for writing (see *f and *g) or if 'n1' is empty.

*m base,a1,n1;

Integer to ASCII conversion - 'n1' is an expression to be converted into an ASCII string representing digits of the 'base' chosen. 'base' must equal "b" for binary, "d" for decimal, or "o" for octal. The result is stored in variable 'a1'. Only one converted string may be stored at any one time. Thus each time the command is invoked, any previous result will be overwritten. Note: when specifying the argument 'base', double quotes should not be used. They are used here for clarity only.

*n

Newline - outputs a new line character (012). The *n command does not require a semicolon, but may have one.

*o a1;

Output word or field - 'a1' is a previously defined word or field that is output as an ASCII string.

*p aa...a;

Print string - 'aa...a' is a string of characters. All output is directed to the standard output, file descriptor 1.

The string 'aa...a' may include:

1. * commands - this allows *p to be used as a grouping command.

2. Expressions - all expressions are evaluated and output as decimal.

3. ASCII characters - these are printed directly.

4. Special ASCII characters preceded by '*' which escapes the regular meaning. Hence, *; *(*) and ** will allow the ; ( ) and * to be printed, respectively.

*r n1[,n2];

Read message into buffer - 'n1' is the number of the message buffer into which the next message is read, where 'n1' ranges in value from 0 through 3. Buffer 0 through 3 can each hold a maximum of 512 bytes. The read will continue until an End-of-Text character (003) is read or the last complete line of the message prior to the 512 byte boundary. Registers 30, 31, and 32 may be used to determine the number of characters read, whether the message overflowed the buffer, and the number of lines read, respectively. Subsequent reads may be performed to read the remainder of a message into a buffer. The optional value 'n2', if given, is taken to be the file descriptor, otherwise zero is assumed. If other than file descriptor 0 is used as input, the corresponding file must be open for reading (see *f and *g). A "1" is returned if the message is successfully read into a buffer and a "0" is returned on failure.

*s a1;

Switch to another file of commands - 'a1' is a variable containing the full path name of the file from which the next program statement is taken, otherwise the full path name is given in double quotes.

*t n1,n2,0,a1[,a2];

Lexical table search - 'n1' specifies the buffer into which the found record is to be read minus the keywords on which the search is performed. It is permitted for this command to specify a value of "4" for 'n1'. This is a special buffer only 80 characters in length. 'n2' specifies the RPG file descriptor of the file to be searched as opened for reading (see *f and *g). The table is assumed to be lexically ordered with fields delimited by spaces (040) and records delimited by the newline character (012). A binary search of the table is performed in this case. 'a1' and 'a2' represent the keywords on which the binary search is to be performed and must match the first two fields of a record. Keyword 'a2' is optional. If a record is found a "1" is returned. If not a "0" is returned.

*t n1,n2,1,n3;

Index table search - 'n1' specifies the buffer into which the found record is to be read. It is permitted for this command to specify a value of "4" for 'n1'. This is a special buffer only 80 characters in length. 'n2' specifies the RPG file descriptor of the file to be searched as opened for reading (see *f and *g). The table is assumed to represent an indexed table with records of equal length. 'n3' specifies the index into the table to be searched. If a record is found a "1" is returned. If not a "0" is returned.

*t n1,n2,2,a1;

Range table search - 'n1' specifies the buffer into which the found record is to be read. It is permitted for this command to specify a value of "4" for 'n1'. This is a special buffer only 80 characters in length. 'n2' specifies the RPG file descriptor of the file to be searched as opened for reading (see *f and *g). 'a1' is the keyword on which the range check is performed. A record whose first entry is less than or equal to the keyword 'a1' and whose second entry is greater than or equal to the keyword 'a1' is retrieved and stored in buffer 'n1'. The table must be sorted in ascending order on the first field. Note that the entire record includig keywords are placed in buffer 'n1'.

*w n1,a1,n2,n3,n4;

Define word - 'n1' is the number of the message buffer into which the message has been previously read. 'a1' is the variable to be defined. 'n2' is the line on which the word is found. Lines are counted from top to bottom beginning with zero. 'n3' is the leftmost character of the word and 'n4' is the rightmost character. The characters are counted from the leftmost character of the line to the rightmost, beginning with zero. This command returns the value "1" if the word is successfully defined or "0" if it cannot be defined. Caution: the End-of-Text character (003) which delimits messages is stripped by RPG and should not be counted as a character when defining a word on the same line.

*x a1,a2;

Character string comparison - 'a1' is a previously defined word or field which is compared to the previously defined word or field 'a2'. 'a2' may also be a string contained in double quotes. 'a1' is permitted to have more characters than 'a2' but not vice-versa. Only the number of characters in 'a2' will be compared with 'a1' beginning with the left-most character of 'a1'. A "1" is returned if the strings match, "0" if they do not.

*X a1,a2;

Exact character string comparison - 'a1' is a previously defined word or field which is compared to the previously defined word or field 'a2'. 'a2' may also be a string contained in double quotes. A "1" will be returned if 'a1' and 'a2' match identically, i.e. same length, same characters, same order. A "0" will be returned if 'a1' and 'a2' do not match identically.

*y a1,a2,n1,n2;

Define subset of word or field - 'a1' is the variable to be defined which may be equal to 'a2'. 'a2' is a previously defined variable. 'n1' specifies the leftmost character to be extracted and 'n2' specifies the rightmost character. The characters are counted from the rightmost character of the predefined word or field to the leftmost, beginning with zero. 'a1' is defined to be the string specified by 'n1' through 'n2' of 'a2'. If this string is larger than 'a2', then 'a1' will be defined as 'a2'. A "1" is returned if the word is successfully defined and a "0" is returned on failure.

*z n1;

Zero register - 'n1' is the name of a register which is set to zero.

*? (expression) command

If statement - if '(expression)' is non-zero, execute 'command', otherwise, skip this command. The terminator of the command is also the terminator of the if statement. Multiple commands may follow the if statement if the *!] command is utilized.

*! command 1,command 2,...,command n]

Grouping - allows multiple commands to be handled as a single command following an if statement. If these commands are executed, they are executed only once. Note: A *>, *<, or *s command cannot be used within a *!]

command. Also, its terminator is a right square bracket.

*! (expression),command 1,...,command n]

While statement - the list of commands is executed as long as '(expression)' is non-zero. Note: A *>, *<, or *s command cannot be used within a *!(exp)] statement. Also, its terminator is a right square bracket.

*> a;

Forward goto - jump forward to label 'a'. Label names may be longer than one character, but only the first character is significant.

*< a;

Backward goto - return to the beginning of the program, then jump forward to label 'a'. Label names may be longer than one character, but only the first character is significant.

*= a1,a2;

Assignment of word or field - 'a1' is a previously defined word or field whose contents are set equal to word or field 'a2'. 'a2' may be string contained in double quotes. Caution should be exercised when 'a1' and 'a2' are of unequal length. The entire contents of 'a2' are written into the buffer on which 'a1' is defined, but the length of 'a1' remains unchanged. This means two things: if 'a2' is shorter than 'a1', 'a1' will contain all of 'a2' (left adjusted) and the remaining portion of its previous value. If 'a2' is longer than 'a1', 'a1' will contain only the left-most portion of 'a2' that it can hold.

*$ n0,a1,n1/.../am,nm;

Initialize and allocate space in a buffer - buffer 'n0' is initialized with space characters (040). Then a template is created in the buffer by defining the name and length of each variable. For instance, variable 'a1' would have a length of 'n1' characters. Subsequent words are defined accordingly. Note that the contents of the buffer is still space characters (040). Data may be placed into each variable in buffer 'n0' via the *= command. All data entered is left adjusted.

DIAGNOSTICS

RPG provides two methods of invoking a trace mechanism so that the flow of the program may be followed. One method is by means of an argument, i.e. "rpg <file> -1" or "rpg <file> -2".

-1   Prints a message each time a goto is executed.

-2   Prints a message each time any * command is encountered.

When either argument is used, trace is invoked for the entire duration of the program, including any programs given control by means of the *s command.

The second method allows trace to be turned on and off in line with the code via an * command. This allows a particular portion of code to be examined without doing a trace on the entire program. Also, two commands (identical) are available that will cause a core image to be created which can be examined with the debugger.

*'0; Causes a core image to be created.

*'1; Turn on trace. This causes messages to be printed each time an * command is encountered, the same as the -2 option above.

*'2; Turn off trace.

*'3; Identical to the *'0 command.

ERROR MESSAGES

In the following error messages <file> is the program in which the error was detected.

RPG 100   CANNOT SWITCH MAIN REPORT FILE. PROGRAM: <file>.
The system call to open the file specified as the main report file failed.

RPG 101   SYS CALL TELL FAILED. PROGRAM: <file>.
The system call TELL on the standard input has failed.

RPG 102   SYS CALL STAT FAILED. PROGRAM: <file>.
The system call STAT on either the standard input or the file /dev/logdev has failed.

RPG 103   READ ERROR. PROGRAM: <file>.
In executing the *r command, one of the system calls READ or SEEK has failed.

RPG 104   EOF. PROGRAM: <file>.
RPG detected an end of file without encountering an *e command.

RPG 105   'I' INDEX OUT OF RANGE. PROGRAM: <file>.

The index specified for an I register is outside the allowable range.

RPG 106 ILLEGAL JMP. PROGRAM: <file>.
An attempt was made to execute an illegal jump statement.

RPG 107 CANNOT SWITCH FILE <file a>. PROGRAM: <file>.
An error has occurred in attempting to switch control to another file of commands, <file a>.

RPG 108 UNDEFINED VARIABLE. PROGRAM: <file>.
An attempt was made to evaluate an undefined variable.

RPG 109 ERROR IN BIT CONVERSION. PROGRAM: <file>.
In attempting to convert an octal or hexidecimal character string to a binary number, an error occurred.

RPG 110 BUFFER NUMBER OUT OF RANGE. PROGRAM: <file>.
A value supplied as an argument to a function was outside an allowable range of values for message buffers.

RPG 111 WRITE ERROR. PROGRAM: <file>.
In attempting to write a message buffer to a file, an error occurred.

RPG 112 FILE OPEN ERROR. PROGRAM: <file>.
In attempting to open a file, an error occurred.

RPG 113 TOO MANY CHARACTERS. PROGRAM: <file>.
In executing the *$ command, one of the arguments encountered was too large or else there were too many arguments supplied.

RPG 114 INVALID OPERATOR. PROGRAM: <file>.
In evaluating an expression, the interpreter has detected an invalid operator.

RPG 115 INVALID OPERAND. PROGRAM: <file>.
While evaluating an expression, the program has detected an invalid operand.

RPG 116 CHL # OUT OF RANGE. PROGRAM: <file>.
The argument to the *t command specifying an index into the channel file was out of range.

RPG 117 BAD SCCERR CALL. PROGRAM: <file>.
In executing an *a6 command, an error occurred.

BUGS
RPG is a bug.