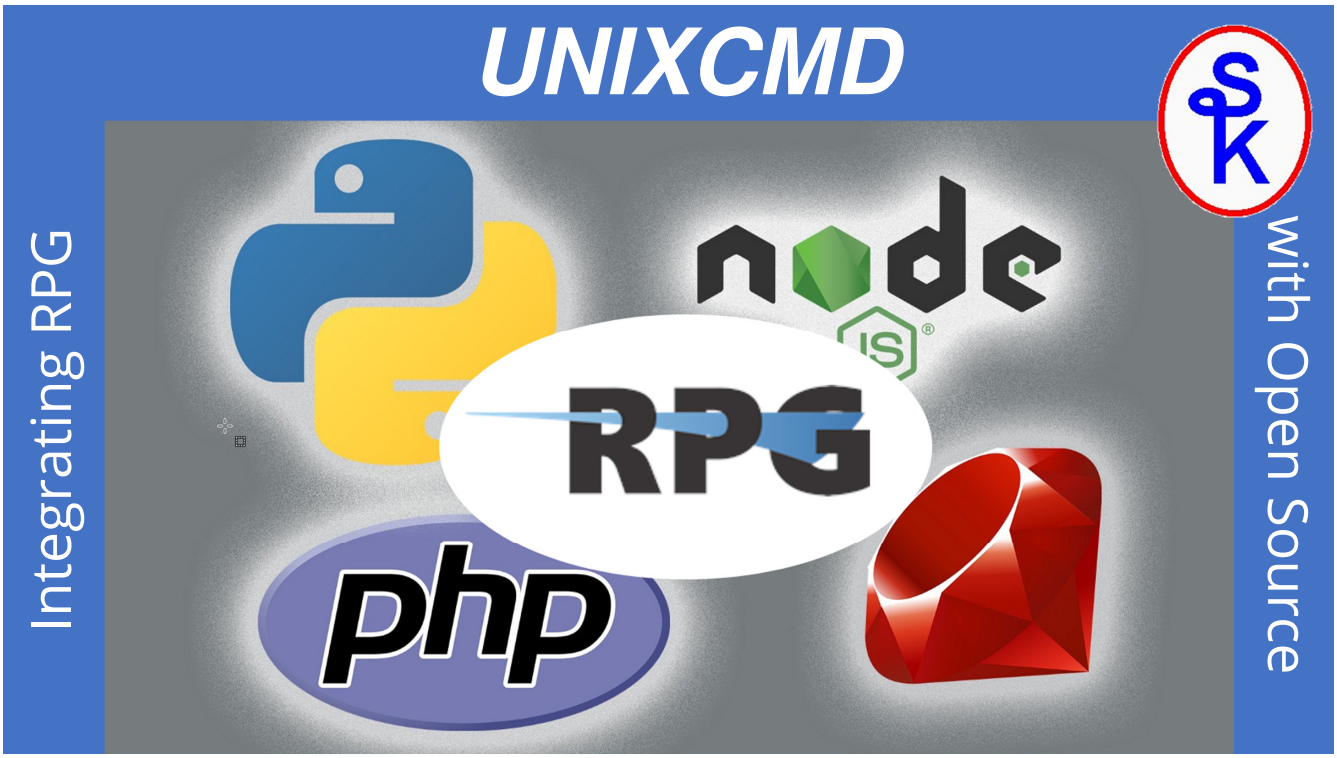


The Problem

We're used to calling programs like this:

```
CALL PGM(the-program) PARM(&PARM1 &PARM2)
```

Or maybe like this:

```
dcl-pr program extpgm('the-program');
  parm1 char(10) const;
  parm2 char(10) const;
end-pr;

program(parm1: parm2);
```

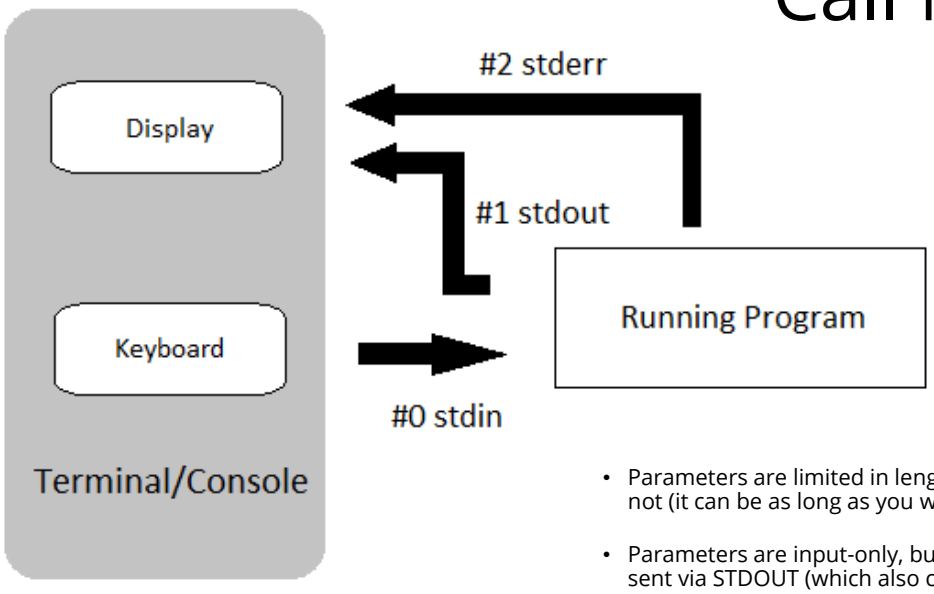
But this doesn't work when calling programs in the PASE/Unix-like environment

- perl
- PHP
- Java
- Ruby
- Node.js
- Python
- Shell scripts
- LUA
- SSH/SFTP

There are two reasons this doesn't work:

1. Unix, Linux, macOS have a different call model
2. Parameters to programs are input-only
3. Parameters must be (relatively) short

Call Model



- Parameters are limited in length, but STDIN is not (it can be as long as you want.)
- Parameters are input-only, but output can be sent via STDOUT (which also can be as long as you want.)
- Error messages are written to STDERR.

Typical CL Solution

Unix programs expect

1. Their own process (vs. call stack entry)
2. Parameters passed by value (vs. reference/pointer)
3. Standard streams (input, output, error)
4. Exit status code (0-255)
5. The services of a Unix shell (Qshell, KSH, Bourne shell, Bash, c-shell, etc)

```
ADDENVVAR ENVVAR(QIBM_QSH_CMD_ESCAPE_MSG) +
          VALUE(Y) +
          REPLACE(*YES)

ADDENVVAR ENVVAR(QIBM_QSH_CMD_OUTPUT) +
          VALUE('FILEAPPEND=/tmp/program.log') +
          REPLACE(*YES)

QSH CMD('program parm1 parm2')
```

This is only a partial solution, because:

1. We can't pass any input to the standard input stream.
2. We can't easily process the output stream.
3. We have to wait for the program to complete before we can get any output at all.

UNIXCMD

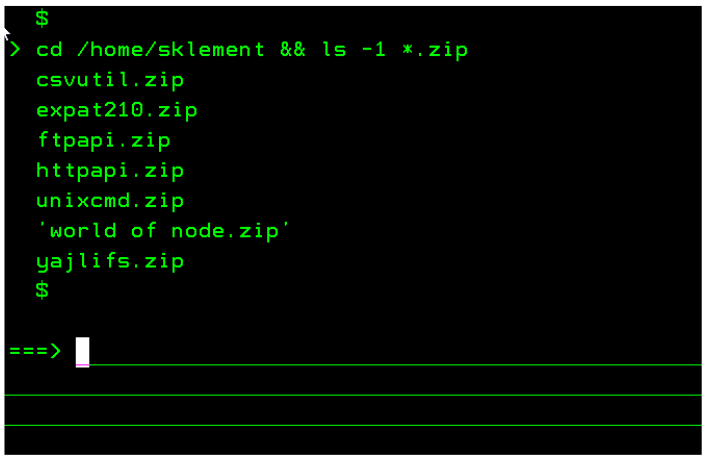
An RPG Open Access handler.

1. The RPG OPEN opcode starts a new process
2. The RPG READ and WRITE opcodes can be used to write data to the process.
3. WRITEs write to the process' STDIN
4. READs read the STDOUT and STDERR (which are combined)
5. The CLOSE opcode checks the exit code and stops the process.

QShell Example

To illustrate the basic concept, the ls command from Unix lists the files from an IFS directory.

For example, I could switch to the /home/sklement directory and list all of the .ZIP files.



RPG QShell Example

```
**FREE
dcl-f UNIX disk(132) handler('UNIXCMDOA': cmd) usropn;
dcl-f QSYSPRT printer(132) usage(*output);

dcl-s  cmd    char(5000);
dcl-ds record len(132) end-ds;

cmd = 'cd /home/sklement && ls -1 *.zip';
open UNIX;

read UNIX record;
dow not %eof(UNIX);
   write QSYSPRT record;    // Print IFS file to spool
   read UNIX record;
enddo;

close UNIX;
```

Notes:

- File must be USROPN so we can set cmd before opening it.
- The && separates two Unix commands. It runs the 2nd only if the first succeeded.
- If Unix program returns an error, the RPG will get an exception during the CLOSE opcode.

The PATH Concept

```
addenvvar envvar(PATH) replace(*yes) level(*job) +
          value('/usr/bin:/usr/sbin:/QOpenSys/pkgs/bin:/QOpenSys/usr/bin')

addenvvar envvar(PASE_PATH)  replace(*yes) level(*job) +
          value('/QOpenSys/pkgs/bin:/QOpenSys/usr/bin:/usr/bin:/usr/sbin')
```

IBM i people often don't understand the concept of a PATH.

- When you run a UNIX program/command, it finds it by searching PATH
- PATH is an environment variable
- The name PATH must be in all capital letters ("PATH" not "path")
- It is a list of IFS directories, separated by colons
- Spaces do count as part of the directory name, so don't put spaces before/after the colons.
- Similar to a library list (albeit, only for program calls.)

How to Set PATH

Use LEVEL(*SYS) to set the default. Users will need to sign off/on to pick up changes.

```
addenvvar envvar(PATH) replace(*yes) level(*sys) +
          value('/usr/bin:/usr/sbin:/QOpenSys/pkgs/bin:/QOpenSys/usr/bin')
```

Use LEVEL(*JOB) to change it within a job.

```
addenvvar envvar(PATH) replace(*yes) level(*job) +
          value('/home/sklement/bin:/usr/bin:/QOpenSys/pkgs/bin:/QOpenSys/usr/bin')
```

In this example *SYS is used to set a reasonable PATH for all users, including any packages installed via YUM.

But I have my own custom programs in /home/sklement/bin, so I change it for my job.

NOTE: Don't set PATH within a program -- just as you wouldn't set your library list in a program! If you do, users can't change it for different purposes!

Sophisticated Example

To illustrate controlling STDIN and STDOUT I will

1) Call a Node.js program called "geocode.js"
2) It expects a street address passed via STDIN
3) It will call a REST API (provided by the US Census Bureau) to "geocode" (determine latitude/longitude) the address
4) The lat/lon coordinates are returned via standard output

From STRQSH I could do this:

```
> cd geocode
  $
> echo "8825 S Howell Avenue, Oak Creek, WI 53154" | node geocode.js
  lat:42.884373
  lon:-87.91229
  $

===>
```

Geocode Example (1 of 2)

```
**free
ctl-opt dftactgrp(*no);

dcl-f UNIX disk(1000) usage(*input:*output) handler('UNIXCMDOA': cmd) usropn;

dcl-pr putenv int(10) extproc(*dclcase);
  varval pointer value options(*string);
end-Pr;

dcl-s cmd char(5000);
dcl-s lat packed(11: 7);
dcl-s lon packed(11: 7);
dcl-s msg char(52);

dcl-ds record qualified;
  type char(3);
  *n   char(1);
  data char(996);
end-ds;

putenv('QIBM_MULTI_THREADED=Y');

cmd = 'cd geocode && node geocode.js';
open UNIX;
```

Geocode Example (2 of 2)

```
record = '8825 S. Howell Avenue Ste 301, Oak Creek, WI 53154';
write UNIX record;

read UNIX record;
if record.type = 'err';
  close UNIX;
  msg = record.data;
  dsply msg;                  // FIXME: Use a real display file
  return;
endif;

lat = %dec(record.data: 11: 7);
read UNIX record;
lon = %dec(record.data: 11: 7);

msg = 'coords=' + %char(lat) + ',' + %char(lon);
dsply msg;                    // FIXME: Use a real display file

close UNIX;

*inlr = *on;
```

Works From CL, Too!

CL doesn't support Open Access, so custom CL commands are supplied in the UNIXCMD library, instead.

```
PGM

   DCL VAR(&REC) TYPE(*CHAR) LEN(1000)
   DCL VAR(&EOF) TYPE(*LGL)

   OPNPIPE CMD('cd /home/sklement; ls *.zip') TYPE(*QSHELL)

   RCVPIPE RCD(&REC) EOF(&EOF)
   DOWHILE (&EOF *EQ '0')

      SNDUSRMSG  MSG(&REC)    /* FIXME: Use a real display file */

      RCVPIPE RCD(&REC) EOF(&EOF)
   ENDDO

   CLOPIPE

ENDPGM
```

Questions?

For this presentation as well as the sample code, visit my web site:

http://www.scottklement.com/presentations/