

hp e3000

perl
programming



perl programming

presented by Mark Bixby
mark_bixby@hp.com

Solution Symposium 2001



Solution Symposium

February 9, 2001

Page 1

hp e3000

perl
programming

introduction and history

- Practical Extraction and Report Language
 - Pathologically Eclectic Rubbish Lister?
- the Swiss Army chainsaw of scripting languages
- optimized for text processing
- combines the best of C, sh, awk, and sed
- released in 1987 by Larry Wall
- initially ported to MPE by Mark Klein
- re-reported by Mark Bixby in 1997 with periodic updates since then
- "There's more than one way to do it!"

hp e3000

perl
programming

current status

- latest & greatest Perl release v5.6.0 available for MPE from bixby.org
- Perl is not supported by HP, but if your use of Perl uncovers any underlying MPE or POSIX bugs, then we certainly want to hear from you!
- the best way to get assistance with Perl on MPE is to post your questions to HP3000-L

hp e3000

perl
programming

installation

- download from <http://www.bixby.org/mark/perlix.html>
- edit and run the INSTALL script
- creates a PERL account
- does not use Priv-Mode capability
- /PERL/PUB/perl is the interpreter
 - don't start scripts with #!/PERL/PUB/PERL

variable names

- scalar values
 - `$days` # the simple scalar value "days"
 - `$days[28]` # the 29th element of array `@days`
 - `$days{'Feb'}` # the 'Feb' value from hash `%days`
 - `$#days` # the last index of array `@days`
- entire arrays or array slices (aka lists)
 - `@days` # (`$days[0]`, `$days[1]`,... `$days[n]`)
 - `@days[3,4,5]` # same as `@days[3..5]`
 - `@days{'a','c'}` # same as (`$days{'a'}`,`$days{'c'}`)
- entire hashes
 - `%days` # (key1, val1, key2, val2 ...)

value constructors

- scalar values
 - `$abc = 12345;`
 - `$abc = 12345.67;`
 - `$abc = 0xffff; # hex`
 - `$abc = 0377 # octal`
 - `$abc = 'a simple string';`
 - `$abc = "a string with a newline\n";`
- list values
 - `@abc = ("cat", "dog", $def);`
 - `($dev, $ino, undef, undef, $uid, $gid) = stat($file);`
- hash values
 - `$abc{'December'} = 12;`
 - `$month = $abc{'December'};`

scalar vs. list context

- the context of some operations will determine the type of the data returned
 - scalar
 - list
- assignment to a scalar variable will evaluate the righthand side in a scalar context
 - `$onerecord = <STDIN>`
- assignment to a list variable will evaluate the righthand side in a list context
 - `@entirefile = <STDIN>`
- context-based behavior is always documented

simple statements

- terminated with a semicolon
- may be followed by one optional modifier
 - `if` `EXPR`
 - `unless` `EXPR`
 - `while` `EXPR`
 - `until` `EXPR`
 - `foreach` `EXPR`
- `$os = 'mpe';`
- `$os = 'mpe' if $model == 3000;`

compound statements

- a block is a sequence of statements delimited by curly brackets (braces) that defines a scope
- compound statements that control flow:
 - `if (EXPR) BLOCK`
 - `if (EXPR) BLOCK else BLOCK`
 - `if (EXPR) BLOCK elsif (EXPR) BLOCK ... else BLOCK`
 - `LABEL while (EXPR) BLOCK`
 - `LABEL while (EXPR) BLOCK continue BLOCK`
 - `LABEL for (EXPR; EXPR; EXPR) BLOCK`
 - `LABEL foreach VAR (LIST) BLOCK`
 - loop control via `next`, `last`, and `redo`
- `if ($model == 3000) { $os = 'mpe' };`

subroutines

```
sub max {  
    my $max = shift(@_);  
    foreach $foo (@_) {  
        $max = $foo if $max < $foo; }  
    return $max;  
}
```

```
$bestday = max($mon,$tue,$wed,$thu,$fri);
```

- parameters passed via @_ array
 - \$_[0] = parm1, \$_[1] = parm2, etc
 - @_ is an alias (i.e. call by reference)
- private variables declared with my
- return or the value of the last expression is the functional return value

hp e3000

perl
programming

arithmetic operators

- addition: +
- subtraction: -
- multiplication: *
- division: /
- modulus: %
- exponentiation: **
- auto-increment and -decrement: ++ --
 - ++\$a - increments \$a, returns new value
 - \$a++ - returns current value, then increments \$a

assignment operators

- works like C
 - `$a += 2;` is equivalent to `$a = $a + 2;`
- `**=` `+=` `*=` `&=` `<<=` `&&=` `-=` `/=`
- `|=` `>>=` `||=` `.=` `%=` `^=` `x=`

relational operators

- numeric comparisons:
 - `<` `>` `<=` `>=` `==` `!=` `<=>`
 - `<=>` returns -1, 0, or 1 depending on whether the left argument is numerically less than, equal to, or greater than the right argument
- string comparisons:
 - `lt` `gt` `le` `ge` `eq` `ne` `cmp`
 - `cmp` returns -1, 0, or 1 depending on whether the left argument is stringwise less than, equal to, or greater than the right argument

hp e3000

perl
programming

bitwise operators

- shift left: <<
- shift right: >>
- AND: &
- OR: |
- XOR: ^
- negation: ~

i/o and file handles

- open files are identified via file handles
- uppercase handle names by convention
- predefined file handles: STDIN, STDOUT, STDERR
- <FILEHANDLE> in a scalar context reads the next record from the file
- <FILEHANDLE> in a list context reads ALL of the remaining records from the file

opening files with open()

- `open(HANDLE, "/file/path")` - open for reading
- `open(HANDLE, "< /file/path")` - open for reading
- `open(HANDLE, "> /file/path")` - open for writing
- `open(HANDLE, ">> /file/path")` - open for appending
- `open(HANDLE, "| shell command")` - open pipe for writing
- `open(HANDLE, "shell command |")` - open pipe for reading
- be very careful when passing user data to `open()` as a file name!

a file i/o example

```
#!/PERL/PUB/perl
```

```
open(HPSW, "/SYS/PUB/HPSWINFO"); # open for input
$one = <HPSW>; # read first line
$two = <HPSW>; # read second line
$three = <HPSW>; # read third line
@therest = <HPSW>; # read all remaining lines
close(HPSW); # close the file
```

```
open(PATCHES, "> /tmp/MPE.patches"); # open for output
foreach $line (@therest) { # access each array line
    print PATCHES $line if $line =~ /^MPE/; # print if match
}
close(PATCHES); # close the file
```

regular expressions

- a vast superset beyond standard Unix regexps
- a ? modifier to make patterns non-greedy
- zero-width lookahead and lookbehind assertions
- conditional expressions
- extra character class matches:
 - \w - match a "word" character (alphanumeric, "_")
 - \W - match a non-word character
 - \s - match a whitespace character
 - \S - match a non-whitespace character
 - \d - match a digit
 - \D - match a non-digit
- <http://www.perl.com/pub/doc/manual/html/pod/perlre.html>

using regular expressions

```
$showme=`callci showme`;
```

```
if ($showme =~ /RELEASE: ([A-Z]\.(\d)(\d)\. \d\d)/) {  
    $release = $1; # the matching V.UU.FF  
    $mpe = "$2.$3"; # the matching U and U (i.e. 7.0)  
}
```

```
$showme =~ s/LDev/Logical Device/gi; # global substitution
```

- $\$n$ contains the value of the n-th matching parenthesized regexp
- the g suffix causes a global substitution
- the i suffix causes case-insensitive matching

how to execute the interpreter

- `/PERL/PUB/perl` [optional parameters]
- `#!/PERL/PUB/perl` [optional parameters]
- `-c` - check syntax without doing execution
- `-d` - run the Perl debugger
- `-e` - specify one line of script (like sed)
- `-v` - print minimal version information
- `-V` - print verbose version information
- `-w` - prints VERY useful syntax and runtime warnings!!!

predefined variables

- `$|` or `$OUTPUT_AUTOFLUSH`
 - By default, all Perl output is buffered (0). To enable automatic flushing, set this variable to 1. Needed when doing MPE I/O which is usually unbuffered.
- `$$` or `$PID`
 - POSIX PID of the current process
- `$^O` or `$OSNAME`
 - operating system name (mpeix)
- `@ARGV`
 - script parameters if any
- `%ENV` or `$ENV{varname}`
 - accesses the POSIX environment variables

built-in functions - a partial list

- Functions for SCALARs or strings
 - [chomp](#), [chop](#), [chr](#), [crypt](#), [hex](#), [index](#), [lc](#), [lcfirst](#), [length](#), [oct](#), [ord](#), [pack](#), [q/STRING/](#), [qq/STRING/](#), [reverse](#), [rindex](#), [sprintf](#), [substr](#), [tr///](#), [uc](#), [ucfirst](#), [y///](#)
- Regular expressions and pattern matching
 - [m//](#), [pos](#), [quotemeta](#), [s///](#), [split](#), [study](#), [qr//](#)
- Numeric functions
 - [abs](#), [atan2](#), [cos](#), [exp](#), [hex](#), [int](#), [log](#), [oct](#), [rand](#), [sin](#), [sqrt](#), [srand](#)
- Functions for real @ARRAYs
 - [pop](#), [push](#), [shift](#), [splice](#), [unshift](#)
- Functions for list data
 - [grep](#), [join](#), [map](#), [qw/STRING/](#), [reverse](#), [sort](#), [unpack](#)
- Functions for real %HASHes
 - [delete](#), [each](#), [exists](#), [keys](#), [values](#)
- Functions for fixed length data or records
 - [pack](#), [read](#), [syscall](#), [sysread](#), [syswrite](#), [unpack](#), [vec](#)



built-in functions (cont.)

- Input and output functions
 - [binmode](#), [close](#), [closedir](#), [dbmclose](#), [dbmopen](#), [die](#), [eof](#), [fileno](#), [flock](#), [format](#), [getc](#), [print](#), [printf](#), [read](#), [readdir](#), [rewinddir](#), [seek](#), [seekdir](#), [select](#), [syscall](#), [sysread](#), [sysseek](#), [syswrite](#), [tell](#), [telldir](#), [truncate](#), [warn](#), [write](#)
- Functions for filehandles, files, or directories
 - [-X](#), [chdir](#), [chmod](#), [chown](#), [chroot](#), [fcntl](#), [glob](#), [ioctl](#), [link](#), [lstat](#), [mkdir](#), [open](#), [opendir](#), [readlink](#), [rename](#), [rmdir](#), [stat](#), [symlink](#), [umask](#), [unlink](#), [utime](#)
- Keywords related to the control flow of your perl program
 - [caller](#), [continue](#), [die](#), [do](#), [dump](#), [eval](#), [exit](#), [goto](#), [last](#), [next](#), [redo](#), [return](#), [sub](#)
- Keywords related to perl modules
 - [do](#), [import](#), [no](#), [package](#), [require](#), [use](#)
- Functions for processes and process groups
 - [alarm](#), [exec](#), [fork](#), [getpgrp](#), [getppid](#), [getpriority](#), [kill](#), [pipe](#), [qx/STRING/](#), [setpgrp](#), [setpriority](#), [sleep](#), [system](#), [times](#), [wait](#), [waitpid](#)
- Time-related functions
 - [gmtime](#), [localtime](#), [time](#), [times](#)

built-in functions (cont.)

- Keywords related to classes and object-orientedness
 - [bless](#), [dbmclose](#), [dbmopen](#), [package](#), [ref](#), [tie](#), [tied](#), [untie](#), [use](#)
- Low-level socket functions
 - [accept](#), [bind](#), [connect](#), [getpeername](#), [getsockname](#), [getsockopt](#), [listen](#), [recv](#), [send](#), [setsockopt](#), [shutdown](#), [socket](#), [socketpair](#)
- System V interprocess communication functions
 - [msgctl](#), [msgget](#), [msgrcv](#), [msgsnd](#), [semctl](#), [semget](#), [semop](#), [shmctl](#), [shmget](#), [shmread](#), [shmwrite](#)
- Fetching user and group info
 - [endgrent](#), [endhostent](#), [endnetent](#), [endpwent](#), [getgrent](#), [getgrgid](#), [getgrnam](#), [getlogin](#), [getpwent](#), [getpwnam](#), [getpwuid](#), [setgrent](#), [setpwent](#)
- Fetching network info
 - [endprotoent](#), [endservent](#), [gethostbyaddr](#), [gethostbyname](#), [gethostent](#), [getnetbyaddr](#), [getnetbyname](#), [getnetent](#), [getprotobyname](#), [getprotobynumber](#), [getprotoent](#), [getservbyname](#), [getservbyport](#), [getservent](#), [sethostent](#), [setnetent](#), [setprotoent](#), [setservent](#)

hp e3000

perl
programming

debugging

- invoke the debugger by starting Perl with the -d parameter
 - `#!/PERL/PUB/perl -d`
- examine or modify variables
- single-step execution
- set breakpoints
- list source code
- set actions to be done before a line is executed
 - `a 53 print "DB FOUND $foo\n"`
- debugger terminal I/O may act a bit strangely on MPE

interprocess communications

- POSIX signals between related processes
- named pipes between unrelated processes
 - create named pipes with POSIX mkfifo command
- unnamed pipes to child processes
 - create using Perl open() function with " | "
- Internet-domain TCP and UDP sockets
- Unix-domain stream sockets
- SysV IPC - shared memory, semaphores, messages

sockets - a procedural client example

```
#!/PERL/PUB/perl -w
```

```
use Socket;
```

```
$proto = getprotobyname('tcp');  
$ipaddr = inet_aton('localhost');  
$port = getservbyname('daytime', 'tcp');  
$address = sockaddr_in($port, $ipaddr);  
socket(SOCK, PF_INET, SOCK_STREAM, $proto);  
connect(SOCK, $address);
```

```
$timestamp = <SOCK>;  
print "$timestamp\n";  
close(SOCK);
```

hp e3000

sockets - an object-oriented client example

perl
programming

```
#!/PERL/PUB/perl -w
```

```
use IO::Socket;
```

```
$remote = IO::Socket::INET->new(  
    Proto = 'tcp',  
    PeerAddr = 'localhost',  
    PeerPort = 'daytime');
```

```
$timestamp = <$remote>;  
print "$timestamp\n";  
close($remote);
```

hp e3000

perl
programming

comprehensive perl archive network (cpan)

- <http://www.cpan.org/>
- a vast collection of free Perl modules
 - ~800 megabytes total as of July 2000
 - mirrored at more than 100 sites around the world
- typical installation process for a CPAN module:
 - `perl Makefile.PL`
 - `make`
 - `make test`
 - `make install`

object oriented programming

- an object consists of:
 - attributes (data)
 - methods (functions to manipulate the attributes)
- many CPAN modules are object-oriented
- for more info:
 - <http://www.perl.com/pub/2000/12/begperl5.html>
 - <http://www.perl.com/pub/doc/manual/html/pod/perltoot.html>

object definitions example - Foo.pm

```
package Foo;

sub new { # method subroutine
    my ($class_name) = @_;

    my ($self) = {}; # create an empty hash to store attributes
    bless ($self, $class_name); # make it an object
    $self->{'_created'} = 1;
    return $self;
}

sub put { # method subroutine
    my ($self, $data) = @_;
    $self->{_bar} = $data; # store data in the _bar attribute
}

sub get { # method subroutine
    my ($self) = @_;
    return $self->{_bar}; # return data from the _bar attribute
}

1; # return code for use statement
```

hp e3000

perl
programming

object usage example

```
#!/PERL/PUB/perl
```

```
use Foo; # refers to Foo.pm file
```

```
$it = new Foo(); # create a new object
```

```
$it->put('hello world'); # use the put method
```

```
printf "The value is %s\n",$it->get(); # get method
```


hp e3000

perl
programming

perl extensions

- binary code residing in an external NMXL loaded at run time
- a thin layer of C that allows the Perl interpreter to call compiled code written in other languages
- several extension libraries come bundled with Perl (sockets, POSIX, etc)
- a decent tutorial is available - the examples even work on MPE!
 - <http://www.perl.com/pub/doc/manual/html/pod/perlxtut.html>
- this is how you would do it to add support for intrinsics or TurboIMAGE

web server cgi - a simple example

```
use CGI qw(:standard);
```

```
print header;  
print start_html('A Simple Example'),  
      h1('A Simple Example'),  
      start_form,  
      "What's your name? ", textfield('name'),  
      p,  
      "What's the combination?",  
      p,  
      checkbox_group(-name=>'words',  
                     -values=>['eenie', 'meenie', 'minie', 'moe'],  
                     -defaults=>['eenie', 'minie']),  
      p,  
      "What's your favorite color? ",  
      popup_menu(-name=>'color',  
                 -values=>['red', 'green', 'blue', 'chartreuse']),  
      p,  
      submit,  
      end_form,  
      hr;
```

web server cgi - a simple example (cont.)

```
if (param()) {  
    print  
        "Your name is ",em(param('name')),  
        p,  
        "The keywords are: ",em(join(" ", param('words'))),  
        p,  
        "Your favorite color is ",em(param('color')),  
        hr;  
}  
print end_html;
```

- <http://stein.cshl.org/WWW/software/CGI/> for more information

hp e3000

perl
programming

mpe as a web client

- it's now possible to write MPE applications that look like web browsers
- perform simple HTTP GET requests, or even complicated HTTP POST requests to fill out remote web forms

```
#!/PERL/PUB/perl  
use LWP::Simple;  
$webpage = get('http://www.bixby.org/mark/perlrx.html');
```

- See <http://www.linpro.no/lwp/> for more information

hp e3000

perl
programming

integration with mpe

- for access to MPE commands:
 - `system("callci mpe_command")`
 - ``callci mpe_command``
- integration with Apache via mod_perl available from <http://www.bixby.org/mark/apacheix.html>
- no direct access to TurboIMAGE yet; workarounds:
 - run a traditional MPE program as a child process
 - use IPC to talk to a custom application server
- no direct access to intrinsics yet
 - a great opportunity for somebody to write the first MPE-specific Perl extension library

perl resources

- <http://www.perl.com/> - the starting point for all things Perl
- http://perl.oreilly.com/news/success_stories.html - how Perl is being used in real-life situations
- <http://www.perl.com/pub/2000/10/begperl1.html> - Beginner's Introduction to Perl
- <http://perl.apache.org/> - The Apache/Perl Integration Project
- <http://www.bixby.org/mark/perlix.html> - for the latest info about Perl on MPE
- Usenet newsgroups comp.lang.perl.*

hp e3000

perl
programming

join the hp3000-I community!

- Available as a mailing list and as the Usenet newsgroup comp.sys.hp.mpe
- In-depth discussions of all things HP e3000
- Talk with other people using Perl on MPE
 - seek advice, exchange tips & techniques
- Keep up with the latest HP e3000 news
- Interact with CSY
- <http://jazz.external.hp.com/papers/hp3000-info.html>