Six Little Languages

(and the problems they solve)

David Tilbrook

dt@qef.com

Nov. 11th, 2003

Six Little Languages
Introduction

Overview of this Homily

- Why Create Little Languages
- How We'll Examine Each Language
- Six (or five) Little Languages
  - mkdeps - solving the dependency problem
  - strfix - source configuration
  - envset - environment configuration
  - qvrs & traits - build & system configuration
  - qsg - script generation
- Lessons Learned
- Tricks of the Trade
- Conclusions

Six Little Languages
Why Create Little Languages

- Appropriate Services and Products

  Ironically little languages can offer flexibility within their domain that cannot be offered by general purpose languages (e.g., sh, awk, perl).
- Convenience of Expression

  Input can be succinct, precise, and appropriate. Little languages often easy to extend or adapt to new uses.
- Return on Investment

  Often easier to create and debug a little language than to struggle to adapt another language to meet requirements.
- Reliability, Efficiency, & Robustness

  One can make little languages very fast and virtually bullet–proof. Sh, perl, and awk scripts are subject to version skew, the user's environment, and often very slooooow.

Six Little Languages
How We'll Examine Each Language

- Problem to be Solved
- Requirements
- Alternative Approaches
- History
- Basic Operation
- Syntax & Semantics
- Examples
- Lessons Learned, Problems, Future
- ...

Six Little Languages
mkdeps (1)

Problem to be Solved

- Generation of prerequisite dependencies, such as:
  - C's "#include"
  - Rc's "TYPELIB", "BITMAP", "FONT", "CURSOR"
  - Java's "import"
  - TeX's "\import" and "\input"

  as might be required by developer or other processes (e.g., mimk, make).

Requirements

- Must be fast –– fast enough to be run every time system is built.
- Must be able to accommodate new languages.
- Must provide a variety of outputs.
- Must be able to process single line.

Six Little Languages
mkdeps (2)

Alternative Approaches

- Makedepend, but:
  - But it's much too slow to run every time despite makedepend(1)'s statement:

    "The approach used in this program enables it to run an order of magnitude faster than any other "dependency generator" I have ever seen."

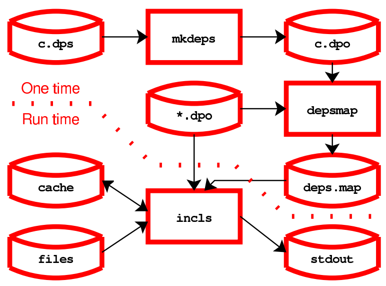
    It's an order of magnitude slower than mkdeps.
  - Its only service is to amend a makefile
  - It's limited to C and C++

History

- qed script to process grep output (circa 1975)
- built–in finite state machine (circa 1979)
  - encoded in C arrays
  - adding new language awkward.
- Attempt to add Boyer–Moore search led to decision to read finite state encoding from a file, hence need for compiler.

Six Little Languages
mkdeps (3)

Basic Operation



- mkdeps compiles *.dps files creating *.dpo files
- depsmap reads *.dpo files to create deps.map (maps suffixes to *.dpo file)
- incls reads input files or lines, runs appro–priate or selected *.dpo file (if necessary) updating cache with scanned results.

Six Little Languages
mkdeps (4)

Example code

```
# /*^ ^{mkdeps}
# x_db source files
# %i file.ext
Description x_db database system
Suffixes *.x{db,i}
Key x
Produces *.xdb/.xo
Search0 @SrcPath ?
Sp=[ \t]
NL=[\n]

Start: '%' =I               # look for leading '%'
       NL
       Else skipline
I: 'i' =Sep                 # got '%'; look for 'i'
       NL =Start
       Else skipline =Start
Sep: Sp                     # skip over white space
       NL =Start
       Else save0 =Gather
Gather: Sp term skipline =Start
       NL term =Start
       Else save
```

Six Little Languages
mkdeps (5)

Portion of c.dps

```
Start: .   =E +7

E:   'e'   =D -1         # got an 'e'; look for preceding 'd'
     'd'   +1
     'u'   +2
     'l'   +3
     'c'   +4
     'n'   +5
     'i'   +6
     '#'   +7
     '\n'  +8
     Sp    +7
     .      skipline +7

D:   'd'   =U -1          # got "de"; look for preceding 'u'
     .     =E hiwater +8
```

- Bops through file looking for an 'e' more than 7 characters into the line.
- If found, checks if preceding char is 'd'. and so on.
- Really motors!

Six Little Languages
mkdeps (6)

Conclusions

- Very cryptic, but no ROI on embellishment. (There are only 12 programs thus far.)
- Doesn't handle #ifdef ... #endif, but that's a bad idea anyway. (mimk handles missing prerequisites properly)
- Speed of finite state machine and caching of previous scans makes it 100 times faster than makedepend, thus can be run for every build.
- Provides large range of outputs, including resolving single line.
- I don't know how people live without it!

Six Little Languages
strfix (1)

Problem to be Solved

In any system, there can be a large number of parameters that need to be incorporated into source at build or installation time.

> In the Q–Tree, 263 files are configured using 367 different parameters such as header file mappings, system capabilities or limitations, and paths.

Requirements

- Need to configure files by replacing symbols by values specified in a dictionary.
- Needs to be able to avoid gratuitous time–stamp propagation, i.e., don't recreate a file that wouldn't change.
- Must be able to process any type of source, e.g., C, sh, make, nroff, tcl, envset, traits, and/or arbitrary data files.
- Needs to be able to check and report usages.
- Needs to be highly portable and reliable as is first tool in use on a new system.

Six Little Languages
strfix (2)

Alternative Approaches

Problem arises in that there are two languages: the dictionary and the file to be processed.

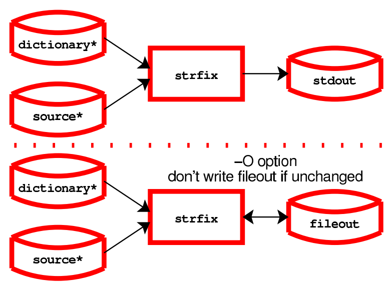
- awk? perl? sh? m4? cpp? sed? qed?
  - Not easy to convert dictionary into code.
  - Must be able to handle arbitrary strings which can raise problems.
  - Need to support multiple languages prohibitive.
- autoconfig? You must be joking!

History

- Used qed (what else?) with hand crafted set of substitutions to process files (circa 1976).
- Created awk script to process dictionary pro–ducing awk script to process files (early 80s).
  - Too slow and vulnerable (not to mention ugly).
  - Gave up when version skews (Unix vs. BSD) proved insurmountable.
- Strfix created in 1981 and used ever since. Many enhancements, most recently to add associative arrays.

Six Little Languages
strfix (3)

Basic Operation



strfix reads dictionary and then source files, replacing @symbol@ expressions by value of symbol in dictionary.

- Can, optionally replace ~XX by special character such as "~RO" –> "®".
- Should have an "include" within dictionaries.
  - finclude used to deal with lack.

Six Little Languages
strfix (4)

Dictionary Syntax & Semantics

`Sym Val`	set symbol Sym to Val
`cset Sym Val`	does set iff Sym unset
`append Sym Val`	append Val to Sym
`unset Sym`	undefine symbol Sym
`setifnil Sym Val`	do set iff Sym empty
`unsetifnil Sym`	unset Sym if empty

- Possible improvements:
  - an include.
  - if ... fi and/or switch statements.

  Resolve lacks using finclude and expressions and switch statements within source file.

Source Syntax & Semantics

`<expression>`	replaced by ...
`@Sym@`	value of Sym
`@Sym:xxx@`	Sym val if set, else "xxx"
`@Sym+set@`	"set" if Sym set
`@Sym?@`	1 if Sym set, otherwise 0
`@Sym=val@`	1 if Sym == "val", else 0

Six Little Languages
strfix (5)

Source Syntax & Semantics continued

`@Sym#d@`	"#define <Sym val> 1", if Sym set
`@Sym#i@`	"\n#include <Sym val>" if set
`@Sym...@`	a variety of other expressions that provide esoteric semantics

Switch statement:

`@{ <string>`	
`@| <pat> ...`	
`...`	interpreted if string matched
`@| ...`	another case; ends previous case
`@}`	end of switch

Additional strfix Facilities

- output resolved dictionary
- output location of symbol uses

Six Little Languages
strfix (6)

Lessons, Problems, Future

- Crucial part of boot strap and system configuration.
  - Used to configure header file wrappers, used to create porting platform and a consistent API.

    Used to create system.h, included in every C source.
  - Isolates all porting issues and host platform attributes and characteristics to one and only one file (i.e., a strfix dictionary). Ports rarely require any source changes other than the creation of strfix dictionary.

  A new port doesn't break any previous ports!!!
- See: http://www.qef.com/html/docs/strfix.pdf
- I don't know how people live without it!

Six Little Languages
envset (1)

Problem to be Solved

The environment variables are very important but rarely well controlled.

Standard systems do not provide adequate tools and mechanisms to control, select, and export an environment specification to a remote host.

Requirements

- Database of selectable environments contain–ing sets and unsets of environment variables, functions, aliases, and local variables.
- Tool to deliver commands to perform sets and unsets for sh or csh as eval arguments, as in: `` eval `envset set1 ...` ``
- Must provide documentation (i.e., list of sets) and debugging aids.
- Must facilitate remote shell environment specification.
- Must be able to specify environment for a single command.

Six Little Languages
envset (2)

Alternative Approaches

- Could create csh or sh scripts to be sourced. No consistent debugging, doesn't facilitate remote execution. Coding is awkward.
- Could use pkg, but non–standard and clumsy. Doesn't provide some necessary facilities. Tends to monotonically grow the environment. Not uncommon to see $PATH settings that are thousands of characters containing all sorts of duplicates and non–existent directories.

History

- Attempted to build rational login scripts and partitioned sets of environments. Created commands to grep and aliases to eval source files.
- Decided on new approach in 1993 and created envset which output evalable [sic] commands.
- envset evolved and other programs enhanced to load envset sets to set internal environ–ments.

Six Little Languages
envset (3)

Basic Operation

- $HOME/.qtree/envset.cf interpreted
  Contains user's personal envsets.
- $QTREE/data/envset.cf interpreted
  Used to specify local settings.
- $QTREE/lib/envset.cf interpreted
  Used to specify standard settings for system.

Interpretation

- Sets to be interpreted selected by argument name or by being added to selection list.
- 1st set encountered for a selected name wins. Subsequent sets with same name not interpreted.
- Interpreter calls program specific routine to do sets.
  - envset outputs sets as sh or csh commands, with ';'s inserted to allow eval interpretation. Actual output delayed until all files processed.
  - Other programs set environment variables directly, and ignores function, alias and variable settings.

Six Little Languages
envset (4)

Set naming and selection

```
<name> <description>      # beginning of a set
&<name>                   # continuation for <name>
~<pat>                    # selected if name matched
addset <name> ...         # names added to list
                          # to be selected
```

Flow Control Commands

```
if <expr>                 # the usual
        ...
elif <expr>
        ...
else
        ...
fi
switch <str>              # Start of switch
case <pat> ...            # if <str> matched
        ...
endswitch
```

Set commands

```
set <Var> <str>           # Set local variable
sset <Var> <str>          # Set shell variable
eset <Var> <str>          # Set export variable
alias <Alias> <str>       # Set alias
function <Funct> <str>      # Declare function
shcmd <command>           # Arbitrary shell command
```

Commands to unset or conditionally set the objects also exist.

Six Little Languages
envset (5)

Example

```
sbin append /sbin:/usr/sbin to PATH
   addset path

path set the PATH
   eset PATH \
      @(trait homeDt)/bin/@System[Name]:\
      @QTREE~sg?:?/bin:?/bin:\
      @(trait STD_PATH):/usr/local/bin

&sbin more sbin settings
   eset PATH @PATH:/sbin:/usr/sbin
```

Notes to Unobfuscate the Above

The '@' is the precursor escape.

- @Var is replaced by current value of Var.
- @(function ...) is call to built–in function
  @(trait ...) retrieves value from traits database described in next section.
- @QTREE~sg?:?/bin:? is replaced by the value of @QTREE with any embedded ':'s replaced by "/bin:". The '~' post–fix operators will be explained later.

Six Little Languages
envset (6)

So ...

Given envset portion on previous page:

```
eval `envset sbin`[1]           # add /sbin & /usr/sbin
                                # to PATH
cush -E sbin <cmmd>             # export PATH before
                                # running <cmmd>
josh -E sbin <job>              # export PATH before
                                # running <job>
rsh host <host-qtree>[2]/bin/cush -Q[3] \
   -sDISPLAY=$DISPLAY \
   -E login <cmmd>              # run remote <cmmd>
                                # with login settings.
```

[1] I have a function Ev() to make this easier:

```
    Ev(){ eval `<qtree>/envset -NQ $*` }
```

[2] <host–qtree> retrieved via socket to qhost.

[3] The –Q flag sets QTREE to <host–qtree>.

Conclusions

- Really useful.
  - My .profile & .bashrc files are tiny but universal
- I don't know how people live without it!

Six Little Languages
qvrs/traits

One or two languages?

qvrs and traits are almost the same language, but

- There are a couple of qvrs facilities not provided in traits.
- traits is host specific (one database per host)
  The traits database is compiled and saved in binary form, and only recompiled when the input files changed.
- qvrs is directory specific.
  qvrs database is compiled whenever needed.
- They process very different files.
- They both deliver variable/value databases.
- They have differing purposes and APIs.

Due to similarities, discussion of traits limited to purpose and basic operations. Discussion of the language deferred to qvrs section.

Six Little Languages
traits (1)

Problem to be Solved

Systems have a number of attributes that vary even within the same operating system such as path names, tool names, system capabilities and facilities. Other software needs to retrieve the value of these attributes.

The attributes (or traits) need to be managed and provided to the user or software easily.

Requirements

- Centralized database of traits and APIs to retrieve these values.
- Debugging aids to help manage the database.

Six Little Languages
traits (2)

Alternative Approaches

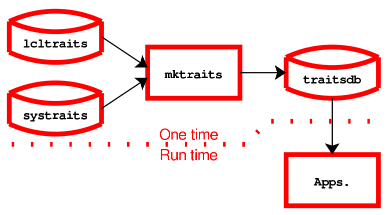
- Require the user to have environment variables that specify the traits.
  - Misplaced responsibility that might be beyond the users' capability or patience.
  - Unreliable and prone to version skew.
- Create overblown, unreliable, crucial, prone–to–failure registry that often crashes and renders the host system unusable, frequently requiring the system to be reinstalled.
  - Not a rational option, but is mentioned here as one organization made this choice.

History

- Initially used environment variables, but this had drawbacks given above.
- Created centralized text file similar to strfix dictionary and associated subroutines to read and parse this file to retrieve the requested traits, but was too slow.
- Adapted qvrs to create mktraits in 1995.

Six Little Languages
traits (3)

Basic Operation



- mktraits –u processes $QTREE/data/traits.ext and $QTREE/lib/traits.vrs to create $QTREE/data/traits/<host>.tab.
  The latter is a network–byte–ordered binary containing a relocatable symbol table defining variable/value tuples.
- Using applications (e.g., qvrs, qsg, envset, traits) use API to load symbol table and extract requested traits.
- sh scripts can use traits (the program) to retrieve a trait.

Six Little Languages
traits (4)

Example traits

```
ARFLAGSDASHED  0                  # ar flags requires '–'
BOGUS_CSH  0                      # sh bug w.r.t. leading #!
BuildPath  /bin /usr/bin              # K.I.S.S.
DefaultQtreeRel  9.1
GOT_PURIFY  0
MAILDIR  /var/mail/%s
Qtree[9.1]  /ph/qtree/linux2_4i/9.1
SNM_CONTROLS  nm -o               # command for snm
STD_PATH  /bin:/usr/bin:/usr/X11R6/bin
STXPREFIXLEN  3
TimeZone  EST5EDT
_T_ar  /usr/bin/ar                # path to ar
_T_cxx  c++
_T_postmail  /usr/sbin/sendmail
_T_ranlib  ranlib                 # could be true, really
_T_troff  groff
```

- Any trait's value may be overridden by setting shell export variable with same name.
- Variables with names of the form _T_*, _F_*, _*_, automatically imported by qvrs and qef.

Six Little Languages
traits (5)

Lessons, Problems, Future

- mktraits has to be setuid owner of Q–Tree. Can lead to problems if root user installs the system as happened at ING in August.
- Reduces porting problems as traits can be set at install time and changed as required.
- Improves control as user does not need to be involved.
- I don't know how people live without it!

Six Little Languages
qvrs (1)

Problem to be Solved

In my experience a build requires hundreds, if not thousands, of parameters such as paths, flags, tool names, and recipes.

Most systems do not have appropriate or sufficient mechanisms to specify, change, manage, test, or document these parameters.

Requirements

- Must be directory specific.
  Must permit trivial local directives to override or modify settings.
- Must provide debugging aids.
  Must provide facilities to determine how and where a variable was set.
- Must provide mechanism to make temporary modifications **without changing the source!!**
- Must be fast, but must be processed at run time.
- Settings must be accessible to all software.

Six Little Languages
qvrs (2)

Alternative Approaches

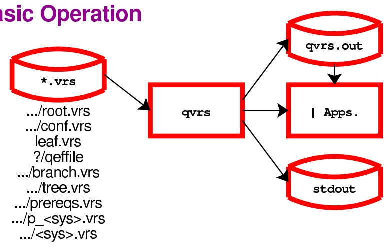
- Make? Poor controls. Inaccessible to other software. Must modify source to change values.
- Imake? Not much improvement over make. No debugging, impossible to tell where something is set.
  - One organization had 30,000 defines. Their builds were very unreliable and slow and they wondered why.
- Ant (i.e., xml)? Well, it is fashionable.

History

- Tried using cpp but rejected that approach quickly. Too many limitations. #define deficient. Not easy to extract the settings.
- Wrote own cpp with #inherit, #append, ... Still wrong approach. Still in use but in a much diminished role.
- Created lclvars, qvrs precursor, and it grew, (>500 deltas) particularly w.r.t. files processed and the ordering of their interpretation.

Six Little Languages
qvrs (3)

Basic Operation

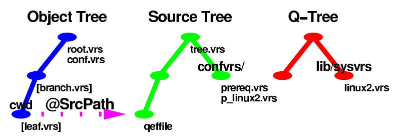


- Applications reads qvrs symbol table from file named by $QVRS_FILE (if defined) or a pipeline from qvrs.
  - Given how many programs use qvrs values, qef creates temporary qvrs symbol table to to eliminate need for children to run qvrs.
- Sh scripts can retrieve qvrs values by running qvrs or qvrsexpr with arguments as in:

```
qvrs @_DestDir_      # retrieve _DestDir_ value
```

Six Little Languages
qvrs (4)

Qvrs Files



File order is crucial and has evolved dramatically

- Starting from cwd search up for root.vrs.
- In root search for @ConfVrs or conf.vrs.
- In cwd look for leaf.vrs – can be suspended.
- Search @SrcPath for qeffile and process preamble ... endpreamble if any.
- Work up to root looking for branch.vrs and search across trees for tree.vrs – may suspend.
- Search @ConfVrsPath for prereq.vrs (optional).
- Search @ConfVrsPath for p_<sysnm>.vrs.
- Search @SysVrsPath for <sysnm>.vrs.
- Pop suspended {branch,tree}.vrs.
- Search @SrcPath for qeffile.
- Pop suspended leaf.vrs.

Six Little Languages
qvrs (5)

Qvrs files continued

root.vrs	Configures the tree's type, src path, config name, build host, revision, etc.
conf.vrs	User's configuration file for flags, compiler options, other options.
leaf.vrs[1]	Temporary overrides for cwd only such as flags, DEBUGGING option.
qeffile[2]	preamble used to specify options and flags before processing following.
branch.vrs[1]	Temporary overrides for sub–tree.
tree.vrs[1]	Common project settings for sub–tree.
prereq.vrs	Controls and sets prerequisite project versions, configs, paths.
p_<sys>.vrs[1]	Project/System specific settings.
<sys>.vrs	System specific settings.
qeffile[2]	Current directory's build controls and build script.

[1] File may be suspended.
[2] Could be qeffile2.

Six Little Languages
qvrs (6)

Keywords

- Usual set of variable settings keywords.
- set, cset, append, prepend, unset, ...
- Special path variable keyword addpath. Maintains ordered list of directories.
- Flow control if ... fi, switch, include, suspend.
- No loops, no input, no procedures.

Expressions

Arguments to keywords expanded to replace @<expressions> by evaluations.

`@<Var>`	replaced by value of <Var>
`@<Var>[str]`	replaced by value of <Var>[str] str is expanded
`@(<F> ...)`	call to internal function <F> such as expr, exists, trait, findfile

@<expression> may be followed by tilde–ops.

`@V~t`	tails of @V elements
`@V~t~r`	basenames of tails of @V elements
`@V~sg/x/y/`	substitute y for x in @V
`@V~m/pat/`	elements of @V that match pat

Six Little Languages
qvrs (7)

Examples

root.vrs portion

```
addpath RootPath /ph/qtree/s9.1/qtree9.1
cset Project qtree
cset Revision 9.1
cset TreeType[/ph/qtree/s9.1/qtree9.1] @
                                    baseline
cset PrereqList tcl(7.4)
cset BuildHost philo
cset QremoteEnv qremote9.1x        # envset selection
```

A qeffile

```
set LibStatic[-ldtree] *
set Library[-lldenv] libldenv.@LibSuffix
set Suffixes -std dat
addpath InclPath @(paths ...
if @(sys unix5.4-mx300i)
        set _F_instal[qmsg] -g tty
fi
cset _D_cc[system.c] -DSYSNAMES_TAB=...
set _F_instal[mktraits] -M4555
Begin qsg -M                # start of the build script
                            # qsg is the script generator
```

Six Little Languages
qvrs (8)

Lessons, Problems, Future

- qvrs is the lynchpin. It has surprised me how important it has become.
- Has been applied to applications beyond software construction such as web page management at the SEI.
- Facilitates rational specification and use of options, controls, and flags.
- I don't know how people live without it!

Six Little Languages
qsg (1)

Problem to be Solved

Translate a simple specification such as:

```
    library -n X c.c yacc.y lex.l
```

(i.e., create libX.a containing named modules) into a script that provides the mechanisms to perform requested constructions.

But there's a lot more than just building libraries and there are target languages other than make (e.g., sh, xfig, ant, & html).

Requirements

- A very, very fast, highly configurable, easily extended, easy to use, comprehensive output generator for arbitrary target languages.

Six Little Languages
qsg (2)

Alternative Approaches

- Imake? As I said before, you must be joking.
  - Clumsy and insufficient levels of abstraction.
  - Little improvement over make itself.
- perl, xml, awk just don't meet requirements.

History

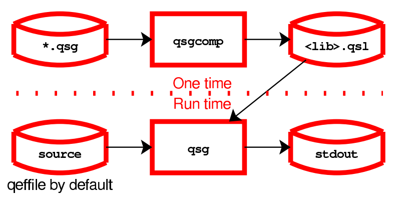
- As always, qed scripts to transform cryptic directives into make files. (1978)
- Specialized programs to create scripts, using advantage of preprocessor facilities (1982).
- Generalized script generation using shell like commands and interpreter (1985).
  - Did support limited '~' ops.
- Interpreter replacing above (1987).

Still not fast enough to run every time.

- qsg created as compiler/interpreter (1989).
  - No input. No procs other than separate files.
  - But fast enough to be run every time.
- Has evolved into a fairly comprehensive language with a wide range of applications.

Six Little Languages
qsg (3)

Basic Operation



- qsgcomp and ar are used to create qsl libraries.
- Normal execution is for qsg to read and compile the source file (defaults to qeffile) and interpret the result.
  - The qsl libraries are searched for scripts called by script being interpreted.
  - Can call source files, rather than qsg object.
- qsg has functions to retrieve traits and qvrs expressions.

Six Little Languages
qsg (4)

Syntax & Semantics

qsg scripts and procs take flags and arguments similar to shell, except syntax strongly enforced.

Scripts and procs specify accepted flags, as in:

```
    summary <Flag-spec> [-] argument ...
    proc <name> <Flag-spec> [-] args ...
```

<Flag-spec>s look like:

–ab	–a and –b accepted. If –a specified, aFlag set to 1, otherwise set to 0.
–k word	flag –k with parameter accepted. If –k word specified, variable kParm set to word. Multiple –k flag not allowed.
–k word+	Same as –k, but can be specified multiple times.
–o[word]	–o flag takes optional argument. oOpt set to specify given options.

This scheme is a real winner.

- No need to encode argument cracking.
- Adding flags and options trivial.

Six Little Languages
qsg (5)

qsg Variables

Local	Similar to C auto variables. Set to empty list on proc or script entry.
Global	Values are persistent.
1st–time–switch	Global test–and–set switches.

qsg Keywords

Variable manipulation: set, cset, add, append, prepend, drop1st, remove, set1st, gset, ...

Flow control: if ... fi, for ... endfor, repeat ... until, while ... endwhile, return, returnval, break, continue, call, <script>.

Miscellaneous: mapscript, qsglib, shell, fatal, message, abort, debug, dump, trace.

I/O: open, reopen, close, flush, write, >, <[\]#>, <<, >>, and ><.

- The functions readline, readstr, and readword used to read input files.

Six Little Languages
qsg (6)

qsg Expressions

All arguments, other than variables, processed to replace @ followed by special symbols:

@<newline>	replaced by space – to wrap lines.
@<space>	replaced by \036. Used to embed literal spaces in list elements.
@<tab>	replaced by \037. Used to embed literal tab in a list element.
@@	A literal @.
@Var[1]	Replaced by value of variable Var.
@(Funct ...)[1]	Function or proc call.
@<name ...>[1]	Equivalent to @(call name ...).
@[V args ...][1]	Args assigned to variable V and returned.
@[/G List ...][1]	Replaced by List ..., which is also assigned to global variable G.
@{Variable}[1]	Replaced by qvrs Variable.

[1] Can be followed by tilde–ops.

Six Little Languages
qsg (7)

The Tilde–Ops

The @ expressions may be followed by one or more tilde–ops. The ops process the argument list, left to right. Some of the 44 ops are:

`@V~0`	The 0th element of V. ~1 for the 2nd element, ~2 for the 3rd, ...
`@V~-3`	V minus first three elements.
`@V~n`	1 if @V not empty, 0 otherwise.
`@V~v`	1 if @V is empty, 0 otherwise.
`@V~t`	tails of the elements
`@V~r`	roots of the elements – suffixes removed.
`@V~x/c.l/`	Elements of V with suffixes .c or .l.
`@V~d`	directories of the elements. "." if none
`@V~h`	heads of the elements. "" if none
`@V~s/p/r/`	Elements of V with 1st parts matched by rxp p replaced by translation of r (e.g., \N: Nth matched part; \uN: upper case of \N.
`@V~=/str/`	Elements of V that equal str.
`@V~g/pat/`	Elements of V matched by rxp pat.
`@V~gt/pat/`	Elements of V whose tails are matched.
`@V~!g/pat/`	Elements of V not matched by rxp pat.
`@V~m/pat/`	Elements of V matched by glob pat.
`@V~(funct)`	Call function with V as arguments.

Six Little Languages
qsg (8)

qsg Functions

There are 64 built–in functions. Some of the more important ones are:

`@(g Var)`	Retrieve global Var
`@(1stset switch)`	Test and set switch. Returns 1 if switch not previously set.
`@(proc ...)`	Call to proc.
`@(call scrpt ...)`	Call to script.
`@(expr <expr>)`	Evaluate <expr> convert to decimal string.

Examples

Echo

```
>@argv
```

Echo basenames of qppt files with counts

```
for A in @argv~x/qppt/~t~r
     >@[N @(expr @N + 1)]~(numf 2d): @A
endfor
```

The Towers of Hanoi

```
summary [-n num] [-] tower names ...
endsummary
hanoi @nFlag~:/4/ @argv     @# default -n to 4
```

Six Little Languages
qsg (9)

Examples

- What about this page?
  This page produced by processing:

```
# /*^ ^{xvxppt}
SlideInit -P
Header -s qsg
H Examples
Bu What about this page?
Bl This page produced by processing:
Cl -O+200 # /*^ ^{xvxppt}
Cl SlideInit -P
...
```

  Who needs powerpoint?
- A generalized version system interface
  - Set of qsg libraries called sccs, rcs, p4, cvs, that provided consistent interface to the version system's commands.
- A html generating library.
  - Provides high level abstractions, taking care of the <>s, popping the stack, etc.
  - Trivial to add new functions (as procs or scripts) to create even higher level abstractions.

Six Little Languages
qsg (10)

Examples cont'd

What about our original requirement?

```
    library -n X c.c yacc.y lex.l
```

The output is too big to show here (105 lines), of preprocessor code. It contains, in part, mimk targets to create {c,yacc,lex}.[ois] (*.i are cpp outputs), to create, update, and install libX.a, directives to remove installed or locally created files.

The following is the output recipe for c.o. Note that incls will be invoked to output #include implied prerequisites for c.o.

```
c.o:P: _S_(c.c) _Touch_(cc)¹
  _T_cc -c _F_cc _F_cc[c.c] _F_cc_c \
    _F_cc_c[c.c] _Optimize_(c.c) \
    _Threads_(c.c) _CcFlags_(c.c) _D_cc \
    _D_cc[c.c] _InclFlags_() _A_(c.c)
```

The qef preprocessor will convert macros such as _T_cc, _F_cc*, _*_ macros appropriately, extracting their values from the qvrs database.

[1] Artificial dependency to force recompilation.

Six Little Languages
qsg (11)

Lessons, Problems, Future

- qsg is a truly remarkable language. It's very fast and fun to use. It has been applied to some remarkably varied problems such as Sci. Am. puzzles and drawing and animating hunt–the–wumpus caves. As for construction:
  - A 554 line Imake file for xfig was replaced by 25 line qeffile which offers far better support.
- One does need readily available document–ation as scripts can have many flags and options.
  - To deal with this need, qsgdump has an option that transforms the summaries as x_db scripts available via x–qsg.
- The <flag–specs> and their interpretation, and tilde–ops are powerful mechanisms that greatly facilitate coding.
- It continues to evolve as it is applied to new applications. Changes sometimes require recompilation of the libraries, but "touchfiles –c qsgcomp" deals with that.
- I don't know how people live without it!

Six Little Languages
Tricks of the Trade

- You'll need a good dynamic string package.
- You'll need a good arithmetic expression evaluator, e.g., Knuth's shunting algorithm.
- You'll need a good string hashing routine for for table look–ups. Here's Peter Pearson's:

```
  int strhash(const char *s,
          Hshtab_t *htab)     /* htab is shuffle of
                               * '\0' to '\377' */
  {
          int h, c;
          unsigned char *p;
          h = 0;
          p = (unsigned char *)s;
          while ((c = *p++) != '\0')
                  h = htab[h^c];
          return h;
  }
```

See Pearson, CACM, June 1990.

Six Little Languages
Tricks of the Trade (2)

- Table driven stuff is easy to extend, and you'll need to.
  qsg started out as a little language but is now approaching medium sized. However, extension is often just a matter of a few table entries and a few lines of code.
- K.I.S.S.
- Make debugging aids part of the language
- Avoid making assumptions w.r.t. limitations of application

Six Little Languages
Conclusions

- Little languages are fun and well worth the effort.
- Is there another way?
- Any other questions?