PACKAGE	SLIDES	EuroTcl
1.5	1.7	2019
		June 2019

Queue

"list" extension package

Purpose

Add path-list functionality

While Addressing / Reflecting:

- namespace pollution by ungrouped extensions
- legacy of ungrouped list commands
- trend toward ensembles

And Preserving:

- compatibility, in all directions

Tcl 8.6 Toplevel Commands

toplevel	~110 cmd/procs (~17 ensembles with ~230 subcommands)	
QUEUE	concat lappend lassign lindex linsert list/create llength lmap lrange lrepeat lreplace lreverse lsearch lset lsort	15

tm::path	add remove list	3
auto	execok import load mkindex reset qualify	6
auto()	execs index noexec noload path	5
tcl	findLibrary endOfWord startOfNextWord startOfPreviousWord wordBreakAfter wordBreakBefore	6
tcl()	library patchLevel pkgPath platform* precision rcFileName traceCompile traceExec nonwordchars wordchars version interactive	12+
redundancies	append close eof fblocked fconfigure fcopy flush gets open puts read seek split tell	14~
array	anymore donesearch exists get names nextelement set size startsearch statistics unset	11
binary	decode encode format scan	4
chan	blocked close configure copy create eof event flush gets names pending pipe pop postevent push puts read seek tell truncate	19
clock	add clicks format microseconds milliseconds scan seconds	7
dde	servername execute poke request services eval	6
dict	append create exists filter for get incr info keys lappend map merge remove replace set size unset update values with	20
encoding	convertfrom convertto dirs names system	5
file	atime attributes channels copy delete dirname executable exists extension isdirectory isfile join link lstat mkdir mtime nativename normalize owned pathtype readable readlink rename rootname separator size split stat system tail tempfile type volumes writeable	34
history	add change clear event info keep nextid redo	8
info	args body class* cmdcount commands complete coroutine default errorstack exists frim functions globals hostname level library loaded locals nameofexecutable object* patchlevel procs script sharedlibextension tclversion vars	26+
interp	alias aliases cancel create debug delete eval exists expose hide hidden invokehidden issafe limit marktrusted recursionlimit share	20

	slaves target transfer	
memory	active break_on_malloc info init objs onexit tag trace tracon_on_at_malloc validate	10
namespace	children code current delect ensemble eval exists export forget import inscope origin parent path qualifiers tail upvar unknown which ensemble*	20+
package	forget ifneeded names present provide require unknown vcompare versions vsatisfies prefer	11
string	cat compare equal first index is* last length map match range repeat replace reverse tolower totitle toupper trim trimleft trimright	20+
trace	add* remove* info* variable vdelete vinfo	6+
zlib	compress decompress deflate gunzip* gzip* inflate push* stream* adler32 crc32	10+

Naming

- the ensemble itself
- the ensemble's subcommands
  - actions with verbs
  - predicates with markers

There are only two hard things in computer science: cache invalidation and naming things
— P. Karlton

- modifiers with markers

Ensemble Name Candidates

- ~~`list`~~ – rejected, because `::list` exists
- `queue`
- `q` – (or `Q`) shortest homonym of queue
- `L` – telephonic; lowercase `l` is deprecated
- `ell` – telephonic and lowercase

Of course, there is always `interp`

```
interp alias {} ł {} queue
```

Rubyish ? Names

Predicate methods end with '?'

```
"hello".empty?                    #=> false
"world".ascii_only?               #=> true
file.upcase.end_with?('.C', '.H') #=> ?
```

Rubyish ! Names

```
unsorted = %w{m a t z}
sorted = unsorted.sort  #=> %w{a m t z}
sorted == unsorted      #=> false
```

`self`-modifying methods end with '!'

```
unsorted = %w{m a t z}
sorted = unsorted.sort! #=> %w{a m t z}
sorted == unsorted      #=> true
```

Random Idioms

lunchp

LISPers might ask questions by using `'p'` to make predicates[1]: `numberp`, `integerp`, `floatp`, `rationalp`, ...

LaTeX

E.g.: `figure*`, `table*`, `section*`, `subsection*`, `paragraph*`, ...

Use of non-alphabetic characters … is not uncommon. … `?` [for] booleans, or `*` for extended … commands
— Ashok P. Nadkarni

Reference Card

`queue` Subcommands 1


```
queue append! varName ?item …?
queue assign list ?item …?
queue car list
queue cdr list
queue c*r list
```

```
queue contains? list ?option …? pattern …
queue create ?item …?
queue create! varName ?item …?
queue empty? list
```

queue Subcommands 2


```
queue index list ?index …?
queue insert list index ?item …?
queue insert! varName index ?item …?
queue length list
queue map name list ?name list …? body
queue prepend list ?item …?
queue prepend! varName ?item …?
```

```
queue postpend list ?item …?
queue postpend! varName ?item …?
```

queue Subcommands 3


```
queue range list first last
queue remove! varName ?option …? pattern
queue repeat count ?item …?
queue replace list first last ?item …?
queue reverse list
queue search ?option …? list pattern
queue set varName ?index …? value
queue sort ?option …? list
queue without list ?option …? pattern
```

`queue` Subcommands 4


```
queue help ?subcommand …?
```

queue Reserved 1

not implemented

```
queue append list ?item …?
queue assign! varName ?varName …?
queue insert! varName index ?item …?
queue pop list ?varName …?
queue pop! varName ?varName …?
queue push list ?item …?
queue push! varName ?item …?
queue range! varName index first last
queue remove varName ?option …? pattern
```

queue Reserved 2


```
queue replace! varName first last ?item …?
queue shift list ?varName …?
queue shift! varName ?varName …?
queue sort! ?option …? varName
queue unshift list ?item …?
queue unshift! varName ?item …?
queue without list ?option …? pattern
```

not implemented

Take-Aways

and Timings

User Take-Away

the `queue` package

- “ensembles” traditional list commands
- adds singleton set operations
  - `prepend`, `prepend!`
  - `postpend`, `postpend!`

Implementer Take-Aways

- rubyisms
  - `predicate?`
  - `mutilator!`
- self-documention
  - `help ?subcommand…?`
- another cute use of `proc unknown`
  - `caddar`, etc.

Wrapper : Builtin

	ms	vs.	ms	:	s.up	slowdown	
queue append!	3.296	lappend	0.354	0.11		9.31	
queue assign	3.780	lassign	0.312	0.08		12.11	
queue create	0.478	list	0.011	0.02		42.86	
queue index	3.639	lindex	0.137	0.04		26.61	
queue insert	4.565	linsert	0.516	0.11		8.85	
queue length	2.358	llength	0.032	0.01		73.61	
queue map	4.719	lmap	1.801	0.38		2.62	min
queue range	3.824	lrange	0.021	0.01		178.59	max
queue repeat	2.418	lrepeat	0.164	0.07		14.71	
queue replace	6.640	lreplace	0.375	0.06		17.69	
queue reverse	2.573	lreverse	0.188	0.07		13.66	
queue search	2.216	lsearch	0.118	0.05		18.85	
queue set!	4.120	lset	0.830	0.20		4.97	
queue sort	2.797	lsort	0.403	0.14		6.94	
(avg drop max)						19.5	

OMG!!

Alias : Builtin

	ms	vs.	ms	:	s.up	slowdown	
queue.append!	0.582	lappend	0.389	0.67		1.50	
queue.assign	0.858	lassign	0.427	0.50		2.01	
queue.create	0.441	list	0.013	0.03		33.09	max
queue.index	0.556	lindex	0.146	0.26		3.80	
queue.insert	0.548	linsert	0.536	0.98			
queue.length	0.406	llength	0.044	0.11		9.23	
queue.map	1.974	lmap	1.764	0.89		1.12	
queue.range	0.422	lrange	0.022	0.05		19.24	
queue.repeat	0.345	lrepeat	0.165	0.48		2.09	
queue.replace	0.496	lreplace	0.519	1.05	1.05		min
queue.reverse	0.372	lreverse	0.169	0.46		2.20	
queue.search	0.342	lsearch	0.131	0.38		2.61	
queue.set!	1.141	lset	0.774	0.68		1.47	
queue.sort	0.536	lsort	0.334	0.62		1.61	
(avg drop max)						4.8	

omg!

Wrapper : Alias

i.e., slower vs. slow					s.up	slowdown	
		lappend			0.16	6.09	
		lassign			0.16	6.25	
		list			0.67	1.50	min
		lindex			0.15	6.50	
		linsert			0.11	8.91	
		llength			0.09	11.00	
		lmap			0.43	2.34	
		lrange			0.20	5.00	
		lrepeat			0.15	6.86	
		lreplace			0.06	17.50	max
		lreverse			0.15	6.57	
		lsearch			0.13	7.60	
		lset			0.29	3.40	
		lsort			0.23	4.43	
(avg drop max)						5.9	

Optimization

“quick and messy” vs. “slow and neat”
specifically, flatland vs. ensembles

First make it work, then make it fast — maybe.

- little-used code needs little speed-up
- development time vs. run time

Question Authoridea(s)

- “ensemble : procs” penalties are less horrendous?
- after compilation, there will be no ensemble penalties?

- coherent ensembles are better than flatland

  ... `linsert list llist llength lmap load lrange` ...