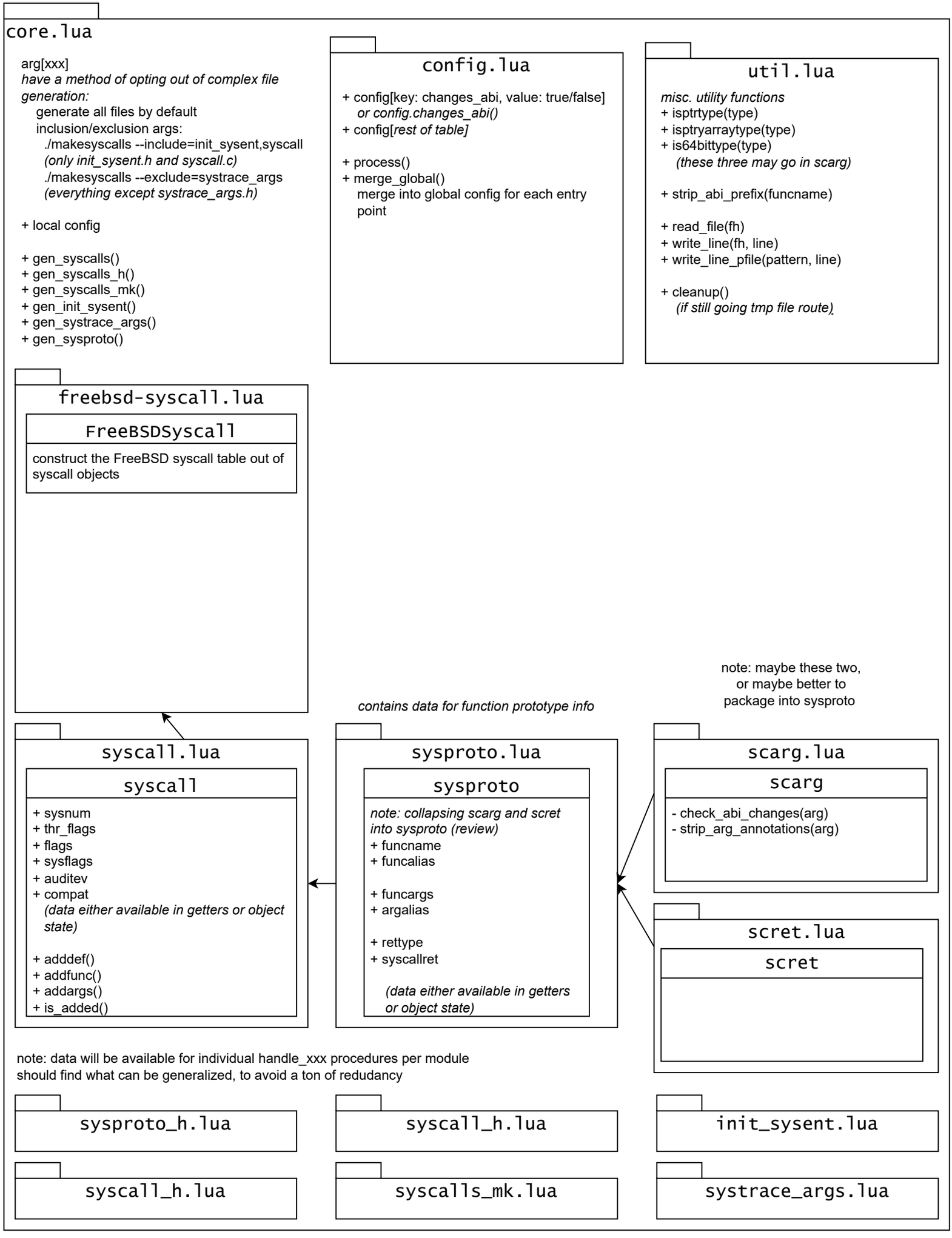
core.lua
arg[xxx]
have a method of opting out of complex file generation:
generate all files by default
inclusion/exclusion args:
./makesyscalls --include=init_sysent,syscall
(only init_sysent.h and syscall.c)
./makesyscalls --exclude=systrace_args
(everything except systrace_args.h)
+ local config
+ gen_syscalls()
+ gen_syscalls_h()
+ gen_syscalls_mk()
+ gen_init_sysent()
+ gen_systrace_args()
+ gen_sysproto()
config.lua
+ config[key: changes_abi, value: true/false]
or config.changes_abi()
+ config[rest of table]
+ process()
+ merge_global()
merge into global config for each entry point
util.lua
misc. utility functions
+ isptrtype(type)
+ isptryarraytype(type)
+ is64bittype(type)
(these three may go in scarg)
+ strip_abi_prefix(funcname)
+ read_file(fh)
+ write_line(fh, line)
+ write_line_pfile(pattern, line)
+ cleanup()
(if still going tmp file route)
freebsd-syscall.lua
FreeBSDSyscall
construct the FreeBSD syscall table out of syscall objects
note: maybe these two, or maybe better to package into sysproto
contains data for function prototype info
syscall.lua
syscall
+ sysnum
+ thr_flags
+ flags
+ sysflags
+ auditev
+ compat
(data either available in getters or object state)
+ adddef()
+ addfunc()
+ addargs()
+ is_added()
sysproto.lua
sysproto
note: collapsing scarg and scret into sysproto (review)
+ funcname
+ funcalias
+ funcargs
+ argalias
+ rettype
+ syscallret
(data either available in getters or object state)
scarg.lua
scarg
- check_abi_changes(arg)
- strip_arg_annotations(arg)
scret.lua
scret
note: data will be available for individual handle_xxx procedures per module
should find what can be generalized, to avoid a ton of redudancy
sysproto_h.lua
syscall_h.lua
init_sysent.lua
syscall_h.lua
syscalls_mk.lua
systrace_args.lua