

NAME

perlintern - autogenerated documentation of purely **internal** Perl functions

DESCRIPTION

This file is the autogenerated documentation of functions in the Perl interpreter that are documented using Perl's internal documentation format but are not marked as part of the Perl API. In other words, **they are not for use in extensions!**

CV reference counts and CvOUTSIDE

CvWEAKOUTSIDE

Each CV has a pointer, `CvOUTSIDE()`, to its lexically enclosing CV (if any). Because pointers to anonymous sub prototypes are stored in `&` pad slots, it is possible to get a circular reference, with the parent pointing to the child and vice-versa. To avoid the ensuing memory leak, we do not increment the reference count of the CV pointed to by `CvOUTSIDE` in the *one specific instance* that the parent has a `&` pad slot pointing back to us. In this case, we set the `CvWEAKOUTSIDE` flag in the child. This allows us to determine under what circumstances we should decrement the refcount of the parent when freeing the child.

There is a further complication with non-closure anonymous subs (i.e. those that do not refer to any lexicals outside that sub). In this case, the anonymous prototype is shared rather than being cloned. This has the consequence that the parent may be freed while there are still active children, eg

```
BEGIN { $a = sub { eval '$x' } }
```

In this case, the `BEGIN` is freed immediately after execution since there are no active references to it: the anon sub prototype has `CvWEAKOUTSIDE` set since it's not a closure, and `$a` points to the same CV, so it doesn't contribute to `BEGIN`'s refcount either. When `$a` is executed, the `eval '$x'` causes the chain of `CvOUTSIDE`s to be followed, and the freed `BEGIN` is accessed.

To avoid this, whenever a CV and its associated pad is freed, any `&` entries in the pad are explicitly removed from the pad, and if the refcount of the pointed-to anon sub is still positive, then that child's `CvOUTSIDE` is set to point to its grandparent. This will only occur in the single specific case of a non-closure anon prototype having one or more active references (such as `$a` above).

One other thing to consider is that a CV may be merely undefined rather than freed, eg `undef &foo`. In this case, its refcount may not have reached zero, but we still delete its pad and its `CvROOT` etc. Since various children may still have their `CvOUTSIDE` pointing at this undefined CV, we keep its own `CvOUTSIDE` for the time being, so that the chain of lexical scopes is unbroken. For example, the following should print 123:

```
my $x = 123;
sub tmp { sub { eval '$x' } }
my $a = tmp();
undef &tmp;
print $a->();
```

```
bool CvWEAKOUTSIDE(CV *cv)
```

Functions in file pad.h

CX_CURPAD_SAVE

Save the current pad in the given context block structure.

```
void CX_CURPAD_SAVE(struct context)
```

CX_CURPAD_SV

Access the SV at offset `po` in the saved current pad in the given context block structure (can be used as an lvalue).

```
SV * CX_CURPAD_SV(struct context, PADOFFSET po)
```

PAD_BASE_SV

Get the value from slot `po` in the base (DEPTH=1) pad of a padlist

```
SV * PAD_BASE_SV(PADLIST padlist, PADOFFSET po)
```

PAD_CLONE_VARS

Clone the state variables associated with running and compiling pads.

```
void PAD_CLONE_VARS(PerlInterpreter *proto_perl, CLONE_PARAMS*  
param)
```

PAD_COMPNAME_FLAGS

Return the flags for the current compiling pad name at offset `po`. Assumes a valid slot entry.

```
U32 PAD_COMPNAME_FLAGS(PADOFFSET po)
```

PAD_COMPNAME_GEN

The generation number of the name at offset `po` in the current compiling pad (lvalue). Note that `SvUVX` is hijacked for this purpose.

```
STRLEN PAD_COMPNAME_GEN(PADOFFSET po)
```

PAD_COMPNAME_GEN_set

Sets the generation number of the name at offset `po` in the current ling pad (lvalue) to `gen`. Note that `SvUV_set` is hijacked for this purpose.

```
STRLEN PAD_COMPNAME_GEN_set(PADOFFSET po, int gen)
```

PAD_COMPNAME_OURSTASH

Return the stash associated with an `our` variable. Assumes the slot entry is a valid `our` lexical.

```
HV * PAD_COMPNAME_OURSTASH(PADOFFSET po)
```

PAD_COMPNAME_PV

Return the name of the current compiling pad name at offset `po`. Assumes a valid slot entry.

```
char * PAD_COMPNAME_PV(PADOFFSET po)
```

PAD_COMPNAME_TYPE

Return the type (stash) of the current compiling pad name at offset `po`. Must be a valid name. Returns null if not typed.

```
HV * PAD_COMPNAME_TYPE(PADOFFSET po)
```

PAD_DUP

Clone a padlist.

```
void PAD_DUP(PADLIST dstpad, PADLIST srcpad, CLONE_PARAMS*
param)
```

PAD_RESTORE_LOCAL

Restore the old pad saved into the local variable opad by PAD_SAVE_LOCAL()

```
void PAD_RESTORE_LOCAL(PAD *opad)
```

PAD_SAVE_LOCAL

Save the current pad to the local variable opad, then make the current pad equal to npad

```
void PAD_SAVE_LOCAL(PAD *opad, PAD *npad)
```

PAD_SAVE_SETNULLPAD

Save the current pad then set it to null.

```
void PAD_SAVE_SETNULLPAD()
```

PAD_SETSV

Set the slot at offset po in the current pad to sv

```
SV * PAD_SETSV(PADOFFSET po, SV* sv)
```

PAD_SET_CUR

Set the current pad to be pad n in the padlist, saving the previous current pad. NB currently this macro expands to a string too long for some compilers, so it's best to replace it with

```
SAVECOMPPAD();
PAD_SET_CUR_NOSAVE(padlist,n);
```

```
void PAD_SET_CUR(PADLIST padlist, I32 n)
```

PAD_SET_CUR_NOSAVE

like PAD_SET_CUR, but without the save

```
void PAD_SET_CUR_NOSAVE(PADLIST padlist, I32 n)
```

PAD_SV

Get the value at offset po in the current pad

```
void PAD_SV(PADOFFSET po)
```

PAD_SVl

Lightweight and lvalue version of PAD_SV. Get or set the value at offset po in the current pad. Unlike PAD_SV, does not print diagnostics with -DX. For internal use only.

```
SV * PAD_SVl(PADOFFSET po)
```

SAVECLEARSV

Clear the pointed to pad value on scope exit. (i.e. the runtime action of 'my')

```
void SAVECLEARSV(SV **svp)
```

SAVECOMPPAD

save PL_comppad and PL_curpad

```
void SAVECOMPPAD()
```

SAVEPADSV

Save a pad slot (used to restore after an iteration)

XXX DAPM it would make more sense to make the arg a PADOFFSET void

```
SAVEPADSV(PADOFFSET po)
```

Functions in file pp_ctl.c

docatch

Check for the cases 0 or 3 of cur_env.je_ret, only used inside an eval context.

0 is used as continue inside eval,

3 is used for a die caught by an inner eval - continue inner loop

See cop.h: je_mustcatch, when set at any runlevel to TRUE, means eval ops must establish a local jmpenv to handle exception traps.

```
OP* docatch(OP *o)
```

GV Functions

gv_try_downgrade

If the typeglob `gv` can be expressed more succinctly, by having something other than a real GV in its place in the stash, replace it with the optimised form. Basic requirements for this are that `gv` is a real typeglob, is sufficiently ordinary, and is only referenced from its package. This function is meant to be used when a GV has been looked up in part to see what was there, causing upgrading, but based on what was found it turns out that the real GV isn't required after all.

If `gv` is a completely empty typeglob, it is deleted from the stash.

If `gv` is a typeglob containing only a sufficiently-ordinary constant sub, the typeglob is replaced with a scalar-reference placeholder that more compactly represents the same thing.

NOTE: this function is experimental and may change or be removed without notice.

```
void gv_try_downgrade(GV* gv)
```

is_gv_magical_sv

Returns TRUE if given the name of a magical GV.

Currently only useful internally when determining if a GV should be created even in rvalue contexts.

`flags` is not used at present but available for future extension to allow selecting particular classes of magical variable.

Currently assumes that `name` is NUL terminated (as well as `len` being valid). This assumption is met by all callers within the perl core, which all pass pointers returned by SvPV.

```
bool is_gv_magical_sv(SV *const name_sv, U32 flags)
```

Hash Manipulation Functions

refcounted_he_chain_2hv

Generates and returns a HV * by walking up the tree starting at the passed in struct refcounted_he *.

NOTE: this function is experimental and may change or be removed without notice.

```
HV * refcounted_he_chain_2hv(const struct refcounted_he *c)
```

refcounted_he_free

Decrements the reference count of the passed in `struct refcounted_he *` by one. If the reference count reaches zero the structure's memory is freed, and `refcounted_he_free` iterates onto the parent node.

NOTE: this function is experimental and may change or be removed without notice.

```
void refcounted_he_free(struct refcounted_he *he)
```

refcounted_he_new

Creates a new `struct refcounted_he`. As key is copied, and value is stored in a compact form, all references remain the property of the caller. The `struct refcounted_he` is returned with a reference count of 1.

NOTE: this function is experimental and may change or be removed without notice.

```
struct refcounted_he * refcounted_he_new(struct refcounted_he
*const parent, SV *const key, SV *const value)
```

IO Functions

start_glob

Function called by `do_readline` to spawn a glob (or do the glob inside perl on VMS). This code used to be inline, but now perl uses `File::Glob` this glob starter is only used by `miniperl` during the build process. Moving it away shrinks `pp_hot.c`; shrinking `pp_hot.c` helps speed perl up.

NOTE: this function is experimental and may change or be removed without notice.

```
PerlIO* start_glob(SV *tmpglob, IO *io)
```

Magical Functions

magic_clearhint

Triggered by a delete from `%^H`, records the key to `PL_compiling.cop_hints_hash`.

```
int magic_clearhint(SV* sv, MAGIC* mg)
```

magic_clearhints

Triggered by clearing `%^H`, resets `PL_compiling.cop_hints_hash`.

```
int magic_clearhints(SV* sv, MAGIC* mg)
```

magic_sethint

Triggered by a store to `%^H`, records the key/value pair to `PL_compiling.cop_hints_hash`. It is assumed that hints aren't storing anything that would need a deep copy. Maybe we should warn if we find a reference.

```
int magic_sethint(SV* sv, MAGIC* mg)
```

mg_localize

Copy some of the magic from an existing SV to new localized version of that SV. Container magic (eg `%ENV`, `$1`, `tie`) gets copied, value magic doesn't (eg `taint`, `pos`).

If `setmagic` is false then no set magic will be called on the new (empty) SV. This

typically means that assignment will soon follow (e.g. 'local \$x = \$y'), and that will handle the magic.

```
void mg_localize(SV* sv, SV* nsv, bool setmagic)
```

MRO Functions

`mro_get_linear_isa_dfs`

Returns the Depth-First Search linearization of @ISA the given stash. The return value is a read-only AV*. `level` should be 0 (it is used internally in this function's recursion).

You are responsible for `SvREFCNT_inc()` on the return value if you plan to store it anywhere semi-permanently (otherwise it might be deleted out from under you the next time the cache is invalidated).

```
AV* mro_get_linear_isa_dfs(HV* stash, U32 level)
```

`mro_isa_changed_in`

Takes the necessary steps (cache invalidations, mostly) when the @ISA of the given package has changed. Invoked by the `setisa` magic, should not need to invoke directly.

```
void mro_isa_changed_in(HV* stash)
```

Pad Data Structures

`CvPADLIST`

CV's can have `CvPADLIST(cv)` set to point to an AV.

For these purposes "forms" are a kind-of CV, `eval""`s are too (except they're not callable at will and are always thrown away after the `eval""` is done executing). Require'd files are simply evals without any outer lexical scope.

XSUBs don't have `CvPADLIST` set - `dXSTARG` fetches values from `PL_curpad`, but that is really the callers pad (a slot of which is allocated by every `entersub`).

The `CvPADLIST` AV has does not have `AvREAL` set, so `REFCNT` of component items is managed "manual" (mostly in `pad.c`) rather than normal `av.c` rules. The items in the AV are not SVs as for a normal AV, but other AVs:

0'th Entry of the `CvPADLIST` is an AV which represents the "names" or rather the "static type information" for lexicals.

The `CvDEPTH`'th entry of `CvPADLIST` AV is an AV which is the stack frame at that depth of recursion into the CV. The 0'th slot of a frame AV is an AV which is `@_`. other entries are storage for variables and op targets.

During compilation: `PL_comppad_name` is set to the names AV. `PL_comppad` is set to the frame AV for the frame `CvDEPTH == 1`. `PL_curpad` is set to the body of the frame AV (i.e. `AvARRAY(PL_comppad)`).

During execution, `PL_comppad` and `PL_curpad` refer to the live frame of the currently executing sub.

Iterating over the names AV iterates over all possible pad items. Pad slots that are `SVs_PADTMP` (targets/GVs/constants) end up having `&PL_sv_undef` "names" (see `pad_alloc()`).

Only my/our variable (`SVs_PADMY`/`SVs_PADOUR`) slots get valid names. The rest are op targets/GVs/constants which are statically allocated or resolved at compile time. These don't have names by which they can be looked up from Perl code at run time through `eval""` like my/our variables can be. Since they can't be looked up by "name" but only by their index allocated at compile time (which is usually in `PL_op->op_targ`), wasting a name SV for them doesn't make sense.

The SVs in the names AV have their PV being the name of the variable. xlow+1..xhigh inclusive in the NV union is a range of cop_seq numbers for which the name is valid. For typed lexicals name SV is SVt_PVMG and SvSTASH points at the type. For our lexicals, the type is also SVt_PVMG, with the SvOURSTASH slot pointing at the stash of the associated global (so that duplicate our declarations in the same package can be detected). SvUVX is sometimes hijacked to store the generation number during compilation.

If SvFAKE is set on the name SV, then that slot in the frame AV is a REFCNT'ed reference to a lexical from "outside". In this case, the name SV does not use xlow and xhigh to store a cop_seq range, since it is in scope throughout. Instead xhigh stores some flags containing info about the real lexical (is it declared in an anon, and is it capable of being instantiated multiple times?), and for fake ANONs, xlow contains the index within the parent's pad where the lexical's value is stored, to make cloning quicker.

If the 'name' is '&' the corresponding entry in frame AV is a CV representing a possible closure. (SvFAKE and name of '&' is not a meaningful combination currently but could become so if `my sub foo {}` is implemented.)

Note that formats are treated as anon subs, and are cloned each time write is called (if necessary).

The flag SVs_PADSTALE is cleared on lexicals each time the `my()` is executed, and set on scope exit. This allows the 'Variable \$x is not available' warning to be generated in evals, such as

```
{ my $x = 1; sub f { eval '$x' } } f();
```

For state vars, SVs_PADSTALE is overloaded to mean 'not yet initialised'

```
AV * CvPADLIST(CV *cv)
```

cv_clone

Clone a CV: make a new CV which points to the same code etc, but which has a newly-created pad built by copying the prototype pad and capturing any outer lexicals.

```
CV* cv_clone(CV* proto)
```

cv_dump

dump the contents of a CV

```
void cv_dump(const CV *cv, const char *title)
```

do_dump_pad

Dump the contents of a padlist

```
void do_dump_pad(I32 level, PerlIO *file, PADLIST *padlist, int full)
```

intro_my

"Introduce" my variables to visible status.

```
U32 intro_my()
```

pad_add_anon

Add an anon code entry to the current compiling pad

```
PADOFFSET pad_add_anon(SV* sv, OPCODE op_type)
```

pad_add_name

Create a new name and associated PADMY SV in the current pad; return the offset. If `typestash` is valid, the name is for a typed lexical; set the name's stash to that value. If `ourstash` is valid, it's an our lexical, set the name's SvOURSTASH to that value

If fake, it means we're cloning an existing entry

NOTE: this function is experimental and may change or be removed without notice.

```
PADOFFSET pad_add_name(const char *name, const STRLEN len,
const U32 flags, HV *typestash, HV *ourstash)
```

pad_alloc

Allocate a new my or tmp pad entry. For a my, simply push a null SV onto the end of `PL_comppad`, but for a tmp, scan the pad from `PL_padix` upwards for a slot which has no name and no active value.

```
PADOFFSET pad_alloc(I32 otype, U32 tmptype)
```

pad_block_start

Update the pad compilation state variables on entry to a new block

```
void pad_block_start(int full)
```

pad_check_dup

Check for duplicate declarations: report any of: * a my in the current scope with the same name; * an our (anywhere in the pad) with the same name and the same stash as `ourstash` is_our indicates that the name to check is an 'our' declaration

```
void pad_check_dup(SV *name, const U32 flags, const HV
*ourstash)
```

pad_findlex

Find a named lexical anywhere in a chain of nested pads. Add fake entries in the inner pads if it's found in an outer one.

Returns the offset in the bottom pad of the lex or the fake lex. `cv` is the CV in which to start the search, and `seq` is the current `cop_seq` to match against. If `warn` is true, print appropriate warnings. The `out_*` vars return values, and so are pointers to where the returned values should be stored. `out_capture`, if non-null, requests that the innermost instance of the lexical is captured; `out_name_sv` is set to the innermost matched `namesv` or fake `namesv`; `out_flags` returns the flags normally associated with the IVX field of a fake `namesv`.

Note that `pad_findlex()` is recursive; it recurses up the chain of CVs, then comes back down, adding fake entries as it goes. It has to be this way because fake `namesvs` in anon prototypes have to store in `xlow` the index into the parent pad.

```
PADOFFSET pad_findlex(const char *name, const CV* cv, U32 seq,
int warn, SV** out_capture, SV** out_name_sv, int *out_flags)
```

pad_fixup_inner_anons

For any anon CVs in the pad, change `CvOUTSIDE` of that CV from `old_cv` to `new_cv` if necessary. Needed when a newly-compiled CV has to be moved to a pre-existing CV struct.

```
void pad_fixup_inner_anons(PADLIST *padlist, CV *old_cv, CV
*new_cv)
```


pad_free

Free the SV at offset po in the current pad.

```
void pad_free(PADOFFSET po)
```

pad_leavemy

Cleanup at end of scope during compilation: set the max seq number for lexicals in this scope and warn of any lexicals that never got introduced.

```
void pad_leavemy()
```

pad_new

Create a new compiling padlist, saving and updating the various global vars at the same time as creating the pad itself. The following flags can be OR'ed together:

```
padnew_CLONE this pad is for a cloned CV
padnew_SAVE  save old globals
padnew_SAVESUB also save extra stuff for start of sub
```

```
PADLIST* pad_new(int flags)
```

pad_push

Push a new pad frame onto the padlist, unless there's already a pad at this depth, in which case don't bother creating a new one. Then give the new pad an @_ in slot zero.

```
void pad_push(PADLIST *padlist, int depth)
```

pad_reset

Mark all the current temporaries for reuse

```
void pad_reset()
```

pad_setsv

Set the entry at offset po in the current pad to sv. Use the macro PAD_SETSV() rather than calling this function directly.

```
void pad_setsv(PADOFFSET po, SV* sv)
```

pad_swipe

Abandon the tmp in the current pad at offset po and replace with a new one.

```
void pad_swipe(PADOFFSET po, bool refadjust)
```

pad_tidy

Tidy up a pad after we've finished compiling it: * remove most stuff from the pads of anonsub prototypes; * give it a @_; * mark tmps as such.

```
void pad_tidy(padtidy_type type)
```

pad_undef

Free the padlist associated with a CV. If parts of it happen to be current, we null the relevant PL_*pad* global vars so that we don't have any dangling references left. We also repoint the CvOUTSIDE of any about-to-be-orphaned inner subs to the outer of this cv.

(This function should really be called `pad_free`, but the name was already taken)

```
void pad_undef(CV* cv)
```

Per-Interpreter Variables

PL_DBsingle

When Perl is run in debugging mode, with the **-d** switch, this SV is a boolean which indicates whether subs are being single-stepped. Single-stepping is automatically turned on after every step. This is the C variable which corresponds to Perl's `$DB::single` variable. See `PL_DBsub`.

```
SV * PL_DBsingle
```

PL_DBsub

When Perl is run in debugging mode, with the **-d** switch, this GV contains the SV which holds the name of the sub being debugged. This is the C variable which corresponds to Perl's `$DB::sub` variable. See `PL_DBsingle`.

```
GV * PL_DBsub
```

PL_DBtrace

Trace variable used when Perl is run in debugging mode, with the **-d** switch. This is the C variable which corresponds to Perl's `$DB::trace` variable. See `PL_DBsingle`.

```
SV * PL_DBtrace
```

PL_dowarn

The C variable which corresponds to Perl's `$^W` warning variable.

```
bool PL_dowarn
```

PL_last_in_gv

The GV which was last used for a filehandle input operation. (`<FH>`)

```
GV* PL_last_in_gv
```

PL_ofsgv

The glob containing the output field separator - `*`, in Perl space.

```
GV* PL_ofsgv
```

PL_rs

The input record separator - `$ /` in Perl space.

```
SV* PL_rs
```

Stack Manipulation Macros

djSP

Declare Just `SP`. This is actually identical to `dSP`, and declares a local copy of perl's stack pointer, available via the `SP` macro. See `SP`. (Available for backward source code compatibility with the old (Perl 5.005) thread model.)

```
djSP;
```

LVRET

True if this op will be the return value of an lvalue subroutine

SV Manipulation Functions

sv_add_arena

Given a chunk of memory, link it to the head of the list of arenas, and split it into a list of free SVs.

```
void sv_add_arena(char *const ptr, const U32 size, const U32 flags)
```

sv_clean_all

Decrement the refcnt of each remaining SV, possibly triggering a cleanup. This function may have to be called multiple times to free SVs which are in complex self-referential hierarchies.

```
I32 sv_clean_all()
```

sv_clean_objs

Attempt to destroy all objects not yet freed

```
void sv_clean_objs()
```

sv_free_arenas

Deallocate the memory used by all arenas. Note that all the individual SV heads and bodies within the arenas must already have been freed.

```
void sv_free_arenas()
```

SV-Body Allocation

sv_2num

Return an SV with the numeric value of the source SV, doing any necessary reference or overload conversion. You must use the `SVNUM(sv)` macro to access this function.

NOTE: this function is experimental and may change or be removed without notice.

```
SV* sv_2num(SV *const sv)
```

Unicode Support

find_uninit_var

Find the name of the undefined variable (if any) that caused the operator `o` to issue a "Use of uninitialized value" warning. If `match` is true, only return a name if its value matches `uninit_sv`. So roughly speaking, if a unary operator (such as `OP_COS`) generates a warning, then following the direct child of the op may yield an `OP_PADSV` or `OP_GV` that gives the name of the undefined variable. On the other hand, with `OP_ADD` there are two branches to follow, so we only print the variable name if we get an exact match.

The name is returned as a mortal SV.

Assumes that `PL_op` is the op that originally triggered the error, and that `PL_comppad/PL_curpad` points to the currently executing pad.

NOTE: this function is experimental and may change or be removed without notice.

```
SV* find_uninit_var(const OP *const obase, const SV *const uninit_sv, bool top)
```

report_uninit

Print appropriate "Use of uninitialized variable" warning

```
void report_uninit(const SV *uninit_sv)
```

Undocumented functions

These functions are currently undocumented:

- F0convert
- Slab_to_rw
- add_data
- add_utf16_textfilter
- addmad
- allocmy
- amagic_cmp
- amagic_cmp_locale
- amagic_i_ncmp
- amagic_ncmp
- anonymise_cv
- ao
- append_elem
- append_list
- append_madprops
- apply
- apply_attrs
- apply_attrs_my
- av_reify
- bad_type
- bind_match
- block_end
- block_start
- boot_core_PerlIO
- boot_core_UNIVERSAL
- boot_core_mro
- bytes_to_uni
- cando
- check_type_and_open
- check_uni
- checkcomma
- checkposixcc
- ck_anoncode
- ck_bitop
- ck_concat
- ck_defined
- ck_delete

ck_die
ck_each
ck_eof
ck_eval
ck_exec
ck_exists
ck_exit
ck_ftst
ck_fun
ck_glob
ck_grep
ck_index
ck_join
ck_lfun
ck_listiob
ck_match
ck_method
ck_null
ck_open
ck_readline
ck_repeat
ck_require
ck_return
ck_rfun
ck_rvconst
ck_sassign
ck_select
ck_shift
ck_sort
ck_spair
ck_split
ck_subr
ck_substr
ck_svconst
ck_trunc
ck_unpack
ckwarn_common
cl_and
cl_anything
cl_init
cl_init_zero
cl_is_anything

cl_or
clear_placeholders
closest_cop
convert
cop_free
cr_textfilter
create_eval_scope
curmad
cv_ckproto_len
deb_curcv
deb_stack_all
deb_stack_n
debprof
debug_start_match
del_sv
delete_eval_scope
deprecate_commaless_var_list
destroy_matcher
die_where
div128
do_aexec
do_aexec5
do_chomp
do_chop
do_delete_local
do_eof
do_exec
do_exec3
do_execfree
do_ipcctl
do_ipcget
do_kv
do_msgrcv
do_msgsnd
do_oddball
do_op_xmldump
do_pmop_xmldump
do_print
do_readline
do_seek
do_semop
do_shmio

do_smartmatch
do_sysseek
do_tell
do_trans
do_trans_complex
do_trans_complex_utf8
do_trans_count
do_trans_count_utf8
do_trans_simple
do_trans_simple_utf8
do_vecget
do_vecset
do_vop
doeval
dofile
dofindlabel
doform
dooneliner
doopen_pm
doparseform
dopoptoeval
dopoptogiven
dopoptolabel
dopoptoloop
dopoptosub_at
dopoptowhen
dump_all_perl
dump_exec_pos
dump_packsubs_perl
dump_sub_perl
dump_sv_child
dump_trie
dump_trie_interim_list
dump_trie_interim_table
dumpuntil
dup_attrlist
emulate_cop_io
exec_failed
expect_number
feature_is_enabled
filter_gets
find_and_forget_pmops

find_array_subscript
find_beginning
find_byclass
find_hash_subscript
find_in_my_stash
find_script
first_symbol
fold_constants
forbid_setid
force_ident
force_list
force_next
force_strict_version
force_version
force_word
forget_pmop
free_tied_hv_pool
gen_constant_list
get_arena
get_aux_mg
get_db_sub
get_debug_opts
get_hash_seed
get_isa_hash
get_no_modify
get_num
get_opargs
get_re_arg
getenv_len
glob_2number
glob_assign_glob
glob_assign_ref
group_end
gv_ename
gv_get_super_pkg
gv_init_sv
hfreeentries
hsplit
hv_auxinit
hv_backreferences_p
hv_copy_hints_hv
hv_delete_common

hv_kill_backrefs
hv_magic_check
hv_notallowed
inline
incpush
incpush_if_exists
incpush_use_sep
ingroup
init_argv_symbols
init_dbargs
init_debugger
init_ids
init_interp
init_main_stash
init_perllib
init_postdump_symbols
init_predump_symbols
intuit_method
intuit_more
invert
io_close
is_an_int
is_handle_constructor
is_inplace_av
is_list_assignment
is_utf8_X_L
is_utf8_X_LV
is_utf8_X_LVT
is_utf8_X_LV_LVT_V
is_utf8_X_T
is_utf8_X_V
is_utf8_X_begin
is_utf8_X_extend
is_utf8_X_non_hangul
is_utf8_X_prepend
is_utf8_char_slow
is_utf8_common
isa_lookup
jmaybe
join_exact
keyword
keyword_plugin_standard

lex_end
lex_start
linklist
list
listkids
localize
looks_like_bool
lop
mad_free
madlex
madparse
magic_clear_all_env
magic_clearenv
magic_clearisa
magic_clearpack
magic_clearsig
magic_existspack
magic_freearylen_p
magic_freeovrld
magic_get
magic_getarylen
magic_getdefelem
magic_getnkeys
magic_getpack
magic_getpos
magic_getsig
magic_getsubstr
magic_gettaint
magic_getuvar
magic_getvec
magic_killbackrefs
magic_len
magic_methcall
magic_methpack
magic_nextpack
magic_regdata_cnt
magic_regdatum_get
magic_regdatum_set
magic_scalarpack
magic_set
magic_set_all_env
magic_setamagic

magic_setarylen
magic_setcollxfrm
magic_setdbline
magic_setdefelem
magic_setenv
magic_setisa
magic_setmglob
magic_setnkeys
magic_setpack
magic_setpos
magic_setregexp
magic_setsig
magic_setsubstr
magic_settaint
magic_setutf8
magic_setuvar
magic_setvec
magic_sizepack
magic_wipepack
make_matcher
make_trie
make_trie_failtable
malloc_good_size
malloced_size
matcher_matches_sv
measure_struct
mem_collxfrm
mem_log_common
mess_alloc
method_common
missingterm
mod
mode_from_discipline
modkids
more_bodies
more_sv
mro_meta_dup
mro_meta_init
mul128
mulexp10
my_attrs
my_betoh16

my_betoh32
my_betoh64
my_betohi
my_betohl
my_betohs
my_clearenv
my_exit_jump
my_htobe16
my_htobe32
my_htobe64
my_htobei
my_htobel
my_htobes
my_htole16
my_htole32
my_htole64
my_htolei
my_htolel
my_htoles
my_kid
my_letoh16
my_letoh32
my_letoh64
my_letohi
my_letohl
my_letohs
my_swabn
my_unexec
need_utf8
newDEFSVOP
newGIVWHENOP
newGP
newMADPROP
newMADsv
newTOKEN
new_constant
new_he
new_logop
new_warnings_bitfield
next_symbol
nextargv
nextchar

no_bareword_allowed
no_fh_allowed
no_op
not_a_number
nuke_stacks
num_overflow
offer_nice_chunk
oopsAV
oopsHV
op_clear
op_const_sv
op_getmad
op_getmad_weak
op_refcnt_dec
op_refcnt_inc
op_xmldump
open_script
opt_scalarhv
pack_rec
package
package_version
pad_add_name_sv
pad_compname_type
pad_peg
parse_body
parse_unicode_opts
parser_free
path_is_absolute
peep
pending_Slabs_to_ro
pidgone
pm_description
pmflag
pmop_xmldump
pmruntime
pmtrans
prepend_elem
prepend_madprops
printbuf
process_special_blocks
ptr_table_find
put_byte

qerror
qsortsvu
re_croak2
readpipe_override
ref_array_or_hash
refcounted_he_fetch
refcounted_he_new_common
refcounted_he_value
refkids
refto
reg
reg_check_named_buff_matched
reg_named_buff
reg_named_buff_iter
reg_namedseq
reg_node
reg_numbered_buff_fetch
reg_numbered_buff_length
reg_numbered_buff_store
reg_qr_package
reg_recode
reg_scan_name
reg_skipcomment
reg_temp_copy
reganode
regatom
regbranch
regclass
regcppop
regcppush
regcurly
regdump_extflags
reghop3
reghop4
reghopmaybe3
reginclass
reginsert
regmatch
regpiece
regposixcc
regprop
regrepeat

regtail
regtail_study
regtry
reguni
regwhite
report_evil_fh
require_tie_mod
restore_magic
rsignal_restore
rsignal_save
run_body
run_user_filter
rxres_free
rxres_restore
rxres_save
same_dirent
save_freeop
save_hek_flags
save_hints
save_lines
save_magic
save_op
save_pushi32ptr
save_pushptri32ptr
save_pushptrptr
save_scalar_at
sawparens
scalar
scalar_mod_type
scalarboolean
scalarkids
scalarseq
scalarvoid
scan_commit
scan_const
scan_formline
scan_heredoc
scan_ident
scan_inputsymbol
scan_pat
scan_str
scan_subst

scan_trans
scan_word
scope
search_const
sequence
sequence_num
sequence_tail
share_hek_flags
sighandler
simplify_sort
skip_space
skip_space0
skip_space1
skip_space2
softref2xv
sortcv
sortcv_stacked
sortcv_xsub
space_join_names_mortal
start_force
stdize_locale
store_cop_label
strip_return
study_chunk
sub_crush_depth
sublex_done
sublex_push
sublex_start
sv_2iuv_common
sv_2iuv_non_preserve
sv_add_backref
sv_catxmlpv
sv_catxmlsv
sv_del_backref
sv_dup_inc_multiple
sv_exp_grow
sv_free2
sv_i_ncmp
sv_kill_backrefs
sv_ncmp
sv_pos_b2u_midway
sv_pos_u2b_cached

sv_pos_u2b_forwards
sv_pos_u2b_midway
sv_release_COW
sv_setsv_cow
sv_unglob
sv_xmlpeek
swallow_bom
swash_get
to_byte_substr
to_utf8_substr
token_free
token_getmad
tokenize_use
tokeq
tokereport
too_few_arguments
too_many_arguments
uiv_2buf
unpack_rec
unshare_hek
unshare_hek_or_pvn
unwind_handler_stack
update_debugger_info
usage
utf16_textfilter
utf8_mg_pos_cache_update
utilize
validate_suid
varname
vdie
vdie_common
vdie_croak_common
visit
vivify_defelem
vivify_ref
wait4pid
watch
write_no_mem
write_to_stderr
xmldump_all
xmldump_all_perl
xmldump_attr

xmldump_eval
xmldump_form
xmldump_indent
xmldump_packsubs
xmldump_packsubs_perl
xmldump_sub
xmldump_sub_perl
xmldump_vindent
yyerror
yylex
yyvsparse
yywarn

AUTHORS

The autodocumentation system was originally added to the Perl core by Benjamin Stuhl. Documentation is by whoever was kind enough to document their functions.

SEE ALSO

perlguts, *perlapi*