NAME

B − The Perl Compiler Backend

SYNOPSIS

```
        use B;
```

DESCRIPTION

The `B` module supplies classes which allow a Perl program to delve into its own innards. It is the module used to implement the "backends" of the Perl compiler. Usage of the compiler does not require knowledge of this module: see the *O* module for the user-visible part. The `B` module is of use to those who want to write new compiler backends. This documentation assumes that the reader knows a fair amount about perl's internals including such things as SVs, OPs and the internal symbol table and syntax tree of a program.

OVERVIEW

The `B` module contains a set of utility functions for querying the current state of the Perl interpreter; typically these functions return objects from the B::SV and B::OP classes, or their derived classes. These classes in turn define methods for querying the resulting objects about their own internal state.

Utility Functions

The `B` module exports a variety of functions: some are simple utility functions, others provide a Perl program with a way to get an initial "handle" on an internal object.

Functions Returning `B::SV` `B::AV` `B::HV` and `B::CV` objects

For descriptions of the class hierarchy of these objects and the methods that can be called on them, see below, "OVERVIEW OF CLASSES" and "SV-RELATED CLASSES".

sv_undef
: Returns the SV object corresponding to the C variable `sv_undef`.

sv_yes
: Returns the SV object corresponding to the C variable `sv_yes`.

sv_no
: Returns the SV object corresponding to the C variable `sv_no`.

svref_2object(SVREF)
: Takes a reference to any Perl value, and turns the referred-to value into an object in the appropriate B::OP or B::SV class. Apart from functions such as `main_root`, this is the primary way to get an initial "handle" on an internal perl data structure which can then be followed with the other access methods.

  The returned object will only be valid as long as the underlying OPs and SVs continue to exist. Do not attempt to use the object after the underlying structures are freed.

amagic_generation
: Returns the SV object corresponding to the C variable `amagic_generation`. As of Perl 5.18, this is just an alias to `PL_na`, so its value is meaningless.

init_av
: Returns the AV object (i.e. in class B::AV) representing INIT blocks.

check_av
: Returns the AV object (i.e. in class B::AV) representing CHECK blocks.

unitcheck_av
: Returns the AV object (i.e. in class B::AV) representing UNITCHECK blocks.

begin_av
: Returns the AV object (i.e. in class B::AV) representing BEGIN blocks.

end_av
: Returns the AV object (i.e. in class B::AV) representing END blocks.

comppadlist

Returns the PADLIST object (i.e. in class B::PADLIST) of the global comppadlist. In Perl 5.16 and earlier it returns an AV object (class B::AV).

regex_padav

Only when perl was compiled with ithreads.

main_cv

Return the (faked) CV corresponding to the main part of the Perl program.

Functions for Examining the Symbol Table

walksymtable(SYMREF, METHOD, RECURSE, PREFIX)

Walk the symbol table starting at SYMREF and call METHOD on each symbol (a B::GV object) visited. When the walk reaches package symbols (such as "Foo::") it invokes RECURSE, passing in the symbol name, and only recurses into the package if that sub returns true.

PREFIX is the name of the SYMREF you're walking.

For example:

```
# Walk CGI's symbol table calling print_subs on each symbol.
# Recurse only into CGI::Util::
walksymtable(\%CGI::, 'print_subs',
             sub { $_[0] eq 'CGI::Util::' }, 'CGI::');
```

print_subs() is a B::GV method you have declared. Also see "B::GV Methods", below.

Functions Returning `B::OP` objects or for walking op trees

For descriptions of the class hierarchy of these objects and the methods that can be called on them, see below, "OVERVIEW OF CLASSES" and "OP-RELATED CLASSES".

main_root

Returns the root op (i.e. an object in the appropriate B::OP class) of the main part of the Perl program.

main_start

Returns the starting op of the main part of the Perl program.

walkoptree(OP, METHOD)

Does a tree-walk of the syntax tree based at OP and calls METHOD on each op it visits. Each node is visited before its children. If `walkoptree_debug` (see below) has been called to turn debugging on then the method `walkoptree_debug` is called on each op before METHOD is called.

walkoptree_debug(DEBUG)

Returns the current debugging flag for `walkoptree`. If the optional DEBUG argument is non-zero, it sets the debugging flag to that. See the description of `walkoptree` above for what the debugging flag does.

Miscellaneous Utility Functions

ppname(OPNUM)

Return the PP function name (e.g. "pp_add") of op number OPNUM.

hash(STR)

Returns a string in the form "0x..." representing the value of the internal hash function used by perl on string STR.

cast_I32(I)

Casts I to the internal I32 type used by that perl.

minus_c

Does the equivalent of the `-c` command-line option. Obviously, this is only useful in a BEGIN block or else the flag is set too late.

cstring(STR)
: Returns a double-quote-surrounded escaped version of STR which can be used as a string in C source code.

perlstring(STR)
: Returns a double-quote-surrounded escaped version of STR which can be used as a string in Perl source code.

safename(STR)
: This function returns the string with the first character modified if it is a control character. It converts it to ˆX format first, so that "\cG" becomes "ˆG". This is used internally by B::GV::SAFENAME, but you can call it directly.

class(OBJ)
: Returns the class of an object without the part of the classname preceding the first `"::"`. This is used to turn `"B::UNOP"` into `"UNOP"` for example.

threadsv_names
: This used to provide support for the old 5.005 threading module. It now does nothing.

Exported utility variables

`@optype`

```
    my $op_type = $optype[$op_type_num];
```

A simple mapping of the op type number to its type (like 'COP' or 'BINOP').

`@specialsv_name`

```
    my $sv_name = $specialsv_name[$sv_index];
```

Certain SV types are considered 'special'. They're represented by B::SPECIAL and are referred to by a number from the specialsv_list. This array maps that number back to the name of the SV (like 'Nullsv' or '&PL_sv_undef').

OVERVIEW OF CLASSES

The C structures used by Perl's internals to hold SV and OP information (PVIV, AV, HV, ..., OP, SVOP, UNOP, ...) are modelled on a class hierarchy and the `B` module gives access to them via a true object hierarchy. Structure fields which point to other objects (whether types of SV or types of OP) are represented by the `B` module as Perl objects of the appropriate class.

The bulk of the `B` module is the methods for accessing fields of these structures.

Note that all access is read-only. You cannot modify the internals by using this module. Also, note that the B::OP and B::SV objects created by this module are only valid for as long as the underlying objects exist; their creation doesn't increase the reference counts of the underlying objects. Trying to access the fields of a freed object will give incomprehensible results, or worse.

SV-RELATED CLASSES

B::IV, B::NV, B::PV, B::PVIV, B::PVNV, B::PVMG, B::PVLV, B::AV, B::HV, B::CV, B::GV, B::FM, B::IO. These classes correspond in the obvious way to the underlying C structures of similar names. The inheritance hierarchy mimics the underlying C "inheritance":

```
                           B::SV
                             |
                +------------+------------+
                |            |            |
              B::PV        B::IV        B::NV
                  \         /           /
                   \       /           /
                    B::PVIV           /
                         \           /
                          \         /
                           \       /
                           B::PVNV
                              |
                              |
                           B::PVMG
                              |
       +-------+-------+---+---+-------+-------+
       |       |       |       |       |       |
     B::AV   B::GV   B::HV   B::CV   B::IO B::REGEXP
               |               |
               |               |
            B::PVLV          B::FM
```

Access methods correspond to the underlying C macros for field access, usually with the leading "class indication" prefix removed (Sv, Av, Hv, ...). The leading prefix is only left in cases where its removal would cause a clash in method name. For example, `GvREFCNT` stays as-is since its abbreviation would clash with the "superclass" method `REFCNT` (corresponding to the C function `SvREFCNT`).

B::SV Methods

REFCNT

FLAGS

object_2svref
: Returns a reference to the regular scalar corresponding to this B::SV object. In other words, this method is the inverse operation to the **svref_2object()** subroutine. This scalar and other data it points at should be considered read-only: modifying them is neither safe nor guaranteed to have a sensible effect.

B::IV Methods

IV
: Returns the value of the IV, *interpreted as a signed integer*. This will be misleading if `FLAGS & SVf_IVisUV`. Perhaps you want the `int_value` method instead?

IVX

UVX

int_value
: This method returns the value of the IV as an integer. It differs from `IV` in that it returns the correct value regardless of whether it's stored signed or unsigned.

needs64bits

packiv

B::NV Methods

NV

NVX

COP_SEQ_RANGE_LOW

COP_SEQ_RANGE_HIGH
: These last two are only valid for pad name SVs. They only existed in the B::NV class before Perl 5.22. In 5.22 they were moved to the B::PADNAME class.

B::RV Methods

RV

B::PV Methods

PV
: This method is the one you usually want. It constructs a string using the length and offset information in the struct: for ordinary scalars it will return the string that you'd see from Perl, even if it contains null characters.

RV
: Same as B::RV::RV, except that it will **die()** if the PV isn't a reference.

PVX
: This method is less often useful. It assumes that the string stored in the struct is null-terminated, and disregards the length information.

It is the appropriate method to use if you need to get the name of a lexical variable from a padname array. Lexical variable names are always stored with a null terminator, and the length field (CUR) is overloaded for other purposes and can't be relied on here.

CUR

This method returns the internal length field, which consists of the number of internal bytes, not necessarily the number of logical characters.

LEN

This method returns the number of bytes allocated (via malloc) for storing the string. This is 0 if the scalar does not ‘‘own'' the string.

B::PVMG Methods

MAGIC
SvSTASH

B::MAGIC Methods

MOREMAGIC
precomp

Only valid on r−magic, returns the string that generated the regexp.

PRIVATE
TYPE
FLAGS
OBJ

Will **die()** if called on r−magic.

PTR
REGEX

Only valid on r−magic, returns the integer value of the REGEX stored in the MAGIC.

B::PVLV Methods

TARGOFF
TARGLEN
TYPE
TARG

B::BM Methods

USEFUL
PREVIOUS
RARE
TABLE

B::REGEXP Methods

REGEX
precomp
qr_anoncv
compflags

The last two were added in Perl 5.22.

B::GV Methods

is_empty

This method returns TRUE if the GP field of the GV is NULL.

NAME
SAFENAME

This method returns the name of the glob, but if the first character of the name is a control character, then it converts it to ^X first, so that *^G would return ‘‘^G'' rather than ‘‘\cG''.

It's useful if you want to print out the name of a variable. If you restrict yourself to globs which exist at compile-time then the result ought to be unambiguous, because code like `${"^G"} = 1` is compiled as two ops − a constant string and a dereference (rv2gv) − so that the glob is created at runtime.

If you're working with globs at runtime, and need to disambiguate *^G from *{"^G"}, then you should use the raw NAME method.

STASH
SV
IO
FORM
AV
HV
EGV
CV
CVGEN
LINE
FILE
FILEGV
GvREFCNT
FLAGS
GPFLAGS
This last one is present only in perl 5.22.0 and higher.

B::IO Methods

B::IO objects derive from IO objects and you will get more information from the IO object itself.

For example:

```
  $gvio = B::svref_2object(\*main::stdin)->IO;
  $IO = $gvio->object_2svref();
  $fd = $IO->fileno();
```

LINES
PAGE
PAGE_LEN
LINES_LEFT
TOP_NAME
TOP_GV
FMT_NAME
FMT_GV
BOTTOM_NAME
BOTTOM_GV
SUBPROCESS
IoTYPE
A character symbolizing the type of IO Handle.

```
    -     STDIN/OUT
    I     STDIN/OUT/ERR
    <     read-only
    >     write-only
    a     append
    +     read and write
    s     socket
    |     pipe
    I     IMPLICIT
    #     NUMERIC
    space closed handle
    \0    closed internal handle
```

IoFLAGS

IsSTD

Takes one argument ('stdin' | 'stdout' | 'stderr') and returns true if the IoIFP of the object is equal to the handle whose name was passed as argument; i.e., `$io->IsSTD('stderr')` is true if IoIFP($io) == **PerlIO_stderr()**.

B::AV Methods

FILL

MAX

ARRAY

ARRAYelt

Like `ARRAY`, but takes an index as an argument to get only one element, rather than a list of all of them.

B::CV Methods

STASH

START

ROOT

GV

FILE

DEPTH

PADLIST

Returns a B::PADLIST object.

OUTSIDE

OUTSIDE_SEQ

XSUB

XSUBANY

For constant subroutines, returns the constant SV returned by the subroutine.

CvFLAGS

const_sv

NAME_HEK

Returns the name of a lexical sub, otherwise `undef`.

B::HV Methods

FILL

MAX

KEYS

RITER

NAME

ARRAY

OP-RELATED CLASSES

`B::OP B::UNOP B::UNOP_AUX B::BINOP B::LOGOP B::LISTOP B::PMOP B::SVOP B::PADOP B::PVOP B::LOOP B::COP B::METHOP`

These classes correspond in the obvious way to the underlying C structures of similar names. The inheritance hierarchy mimics the underlying C "inheritance":

```
                            B::OP
                              |
             +----------+---------+--------+-------+---------+
             |          |         |        |       |         |
          B::UNOP    B::SVOP  B::PADOP  B::COP  B::PVOP  B::METHOP
             |
         +---+---+---------+
         |       |         |
      B::BINOP  B::LOGOP  B::UNOP_AUX
         |
         |
      B::LISTOP
         |
     +---+---+
     |       |
  B::LOOP   B::PMOP
```

Access methods correspond to the underlying C structure field names, with the leading "class indication" prefix (`"op_"`) removed.

B::OP Methods

These methods get the values of similarly named fields within the OP data structure. See top of `op.h` for more info.

next
sibling
parent

Returns the OP's parent. If it has no parent, or if your perl wasn't built with `-DPERL_OP_PARENT`, returns NULL.

Note that the global variable `$B::OP::does_parent` is undefined on older perls that don't support the `parent` method, is defined but false on perls that support the method but were built without `-DPERL_OP_PARENT`, and is true otherwise.

name

This returns the op name as a string (e.g. "add", "rv2av").

ppaddr

This returns the function name as a string (e.g. "PL_ppaddr[OP_ADD]", "PL_ppaddr[OP_RV2AV]").

desc

This returns the op description from the global C PL_op_desc array (e.g. "addition" "array deref").

targ
type
opt
flags
private
spare

B::UNOP Method

first

B::UNOP_AUX Methods (since 5.22)

aux_list(cv)

This returns a list of the elements of the op's aux data structure, or a null list if there is no aux. What will be returned depends on the object's type, but will typically be a collection of `B::IV` `B::GV` etc. objects. `cv` is the `B::CV` object representing the sub that the op is contained within.

string(cv)
: This returns a textual representation of the object (likely to b useful for deparsing and debugging), or an empty string if the op type doesn't support this. `cv` is the `B::CV` object representing the sub that the op is contained within.

B::BINOP Method

last

B::LOGOP Method

other

B::LISTOP Method

children

B::PMOP Methods

pmreplroot
pmreplstart
pmflags
precomp
pmoffset
: Only when perl was compiled with ithreads.

code_list
: Since perl 5.17.1

pmregexp
: Added in perl 5.22, this method returns the B::REGEXP associated with the op. While PMOPs do not actually have `pmregexp` fields under threaded builds, this method returns the regexp under threads nonetheless, for convenience.

B::SVOP Methods

sv
gv

B::PADOP Method

padix

B::PVOP Method

pv

B::LOOP Methods

redoop
nextop
lastop

B::COP Methods

The `B::COP` class is used for "nextstate" and "dbstate" ops. As of Perl 5.22, it is also used for "null" ops that started out as COPs.

label
stash
stashpv
stashoff (threaded only)
file
cop_seq
arybase
line
warnings
io
hints

hints_hash

B::METHOP Methods (Since Perl 5.22)

first

meth_sv

PAD-RELATED CLASSES

Perl 5.18 introduced a new class, B::PADLIST, returned by B::CV's `PADLIST` method.

Perl 5.22 introduced the B::PADNAMELIST and B::PADNAME classes.

B::PADLIST Methods

MAX

ARRAY
: A list of pads. The first one is a B::PADNAMELIST containing the names. The rest are currently B::AV objects, but that could change in future versions.

ARRAYelt
: Like `ARRAY`, but takes an index as an argument to get only one element, rather than a list of all of them.

NAMES
: This method, introduced in 5.22, returns the B::PADNAMELIST. It is equivalent to `ARRAYelt` with a 0 argument.

REFCNT

id
: This method, introduced in 5.22, returns an ID shared by clones of the same padlist.

outid
: This method, also added in 5.22, returns the ID of the outer padlist.

B::PADNAMELIST Methods

MAX

ARRAY

ARRAYelt
: These two methods return the pad names, using B::SPECIAL objects for null pointers and B::PADNAME objects otherwise.

REFCNT

B::PADNAME Methods

PV

PVX

LEN

REFCNT

FLAGS
: For backward-compatibility, if the PADNAMEt_OUTER flag is set, the FLAGS method adds the SVf_FAKE flag, too.

TYPE
: A B::HV object representing the stash for a typed lexical.

SvSTASH
: A backward-compatibility alias for TYPE.

OURSTASH
: A B::HV object representing the stash for 'our' variables.

PROTOCV
: The prototype CV for a 'my' sub.

COP_SEQ_RANGE_LOW

COP_SEQ_RANGE_HIGH
: Sequence numbers representing the scope within which a lexical is visible. Meaningless if PADNAMEt_OUTER is set.

PARENT_PAD_INDEX
: Only meaningful if PADNAMEt_OUTER is set.

PARENT_FAKELEX_FLAGS
: Only meaningful if PADNAMEt_OUTER is set.

`$B::overlay`

Although the optree is read-only, there is an overlay facility that allows you to override what values the various B::*OP methods return for a particular op. `$B::overlay` should be set to reference a two-deep hash: indexed by OP address, then method name. Whenever a an op method is called, the value in the hash is returned if it exists. This facility is used by B::Deparse to "undo" some optimisations. For example:

```
local $B::overlay = {};
...
if ($op->name eq "foo") {
    $B::overlay->{$$op} = {
            name => 'bar',
            next => $op->next->next,
    };
}
...
$op->name # returns "bar"
$op->next # returns the next op but one
```

AUTHOR

Malcolm Beattie, `mbeattie@sable.ox.ac.uk`