NAME

B - The Perl Compiler

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-derived or B::SV-derived 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.

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 AV object (i.e. in class B::AV) of the global comppadlist.
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::*');

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

Functions Returning B::OP objects or for walking op trees
main_root
Returns the root op (i.e. an object in the appropriate B::OP-derived 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.

`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`  
In a perl compiled for threads, this returns a list of the special per-thread threadsv variables.

Exported utility variables

@optype

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

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

@specialsv_name

```perl
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::RV, B::PV, B::PVIV, B::PVNV, B::PVMG, B::BM (5.9.5 and earlier), 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". For 5.9.5 and later this is:

```
+------------+------------+------------+
|            |            |            |
| B::PV      | B::IV      | B::NV      |
|            |            | B::RV      |
|            |            |            |
| B::PVIV    |            |            |            |
|            |            |            |
| B::PVLV    | B::FM      |
```

For 5.9.0 and earlier, PVLV is a direct subclass of PVMG, and BM is still present as a distinct type, so the base of this diagram is:

```
+------------+------------+------------+------------+------------+------------+
|            |            |            |            |            |            |
| B::PVLV    | B::BM      | B::AV      | B::GV      | B::HV      | B::CV      |
|            |            |            |            |            | B::IO      |
|            |            |            |            |            |            |
| 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**

**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 (SvCUR) is overloaded for other purposes and can't be relied on here.

**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::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
B::IO Methods

LINES
PAGE
PAGE_LEN
LINES_LEFT
TOP_NAME
TOP_GV
FMT_NAME
FMT_GV
BOTTOM_NAME
BOTTOM_GV
SUBPROCESS
IoTYPE
IoFLAGS
IsSTD

Takes one arguments (‘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_stdin()).

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.

OFF

This method is deprecated if running under Perl 5.8, and is no longer present if running under Perl 5.9

AvFLAGS

This method returns the AV specific flags. In Perl 5.9 these are now stored in with the main SV flags, so this method is no longer present.

B::CV Methods

STASH
START
ROOT
GV
FILE
DEPTH
PADLIST
OUTSIDE
OUTSIDE_SEQ
XSUB
XSUBANY

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

CvFLAGS
const_sv

**B::HV Methods**

FILL
MAX
KEYS
RITER
NAME
ARRAY
PMROOT

This method is not present if running under Perl 5.9, as the PMROOT information is no longer stored directly in the hash.

**OP-RELATED CLASSES**

B::OP, B::UNOP, B::BINOP, B::LOGOP, B::LISTOP, B::PMOP, B::SVOP, B::PADOP, B::PVOP,
B::LOOP, B::COP.

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

```
+---------------+--------+--------+-------+
|                   |        |        |       |
+---------------+--------+--------+-------+
| B::UNOP        | B::SVOP| B::PADOP| B::COP |
| /               | /      | /      | /     |
| `-'             | `-'    | `-'    | `-'   |
| B::BINOP       | B::LOGOP|
| /              | /      |
| `-'            | `-'    |
| 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
next_sibling
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]").
desc  "PL_ppaddr[OP_RV2AV]".
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::BINOP METHOD
  last

B::LOGOP METHOD
  other

B::LISTOP METHOD
  children

B::PMOP Methods
  pmreplroot
  pmreplstart
  pmnext
  pmregexp
  pmflags
  extflags
  precomp
  pmoffset

  Only when perl was compiled with ithreads.

B::SVOP METHOD
  sv
  gv

B::PADOP METHOD
  padix

B::PVOP METHOD
  pv

B::LOOP Methods
  redoop
  nextop
  lastop
B::COP Methods

label
stash
stashpv
file
cop_seq
arybase
line
warnings
io
hints
hints_hash

AUTHOR

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