OpenCOBOL

1.1pre-rel

Generated by Doxygen 1.7.4

Sun May 1 2011 16:55:26

Contents

Chapter 1

OpenCOBOL 1.1pre-rel source tree

Authors

Keisuke Nishida, Roger While

1.1 Introduction

This is the OpenCOBOL compiler suite source tree, beautified by Doxygen. All rights remain the authors, Keisuke Nishida and Roger While.

See the Classes (try cb_field for it's dependency graph) and Files, where:

1. src/cobc Contains the compiler sources
2. src/libcob Contains the runtime support sources
3. src/lib and
4. src/bin contain build support files

As you prepare to develop code with OpenCOBOL, please be sure to visit OpenCOBOL.org

Many thanks to Dimitri for Doxygen. This is sweet.

Many thanks to Roger and Keisuke. Copyrights for OpenCOBOL are Roger While and Keisuke Nishida.

Chapter 2

Directory Hierarchy

2.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

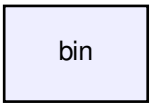
Here is a list of all files with brief descriptions:

Chapter 5

Directory Documentation

5.1 bin/ Directory Reference

Directory dependency graph for bin/:

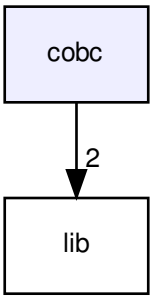


Files

- file cobcrun.c

5.2 cobc/ Directory Reference

Directory dependency graph for cobc/:

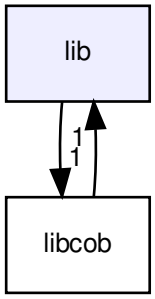


Files

- file cobc.c
- file cobc.h
- file codegen.c
- file config.c
- file error.c
- file field.c
- file parser.c
- file parser.h
- file pplex.c
- file ppparse.c
- file ppparse.h
- file reserved.c
- file scanner.c
- file tree.c
- file tree.h
- file typeck.c

5.3 lib/ Directory Reference

Directory dependency graph for lib/:

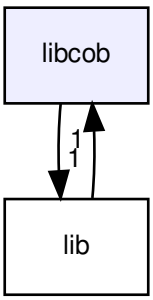


Files

- file dummymac.c
- file getopt.c
- file getopt.h
- file getopt1.c
- file gettext.h

5.4 libcob/ Directory Reference

Directory dependency graph for libcob/:



Files

- file byteswap.h
- file call.c
- file call.h
- file coblocal.h
- file codegen.h
- file common.c
- file common.h
- file fileio.c
- file fileio.h
- file intrinsic.c
- file intrinsic.h
- file move.c
- file move.h
- file numeric.c
- file numeric.h
- file screenio.c
- file screenio.h
- file strings.c
- file strings.h
- file termio.c
- file termio.h

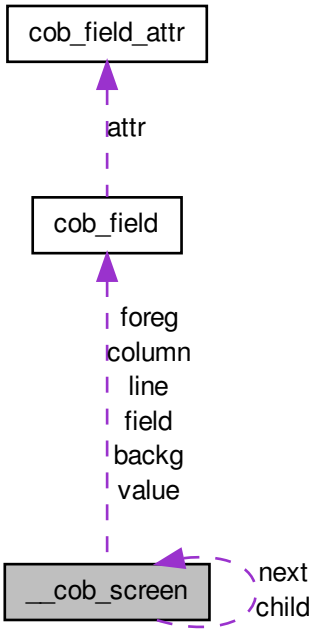
Chapter 6

Class Documentation

6.1 __cob_screen Struct Reference

`#include <screenio.h>`

Collaboration diagram for __cob_screen:



Public Attributes

- cob_screen * next
- cob_screen * child
- cob_field * field
- cob_field * value
- cob_field * line
- cob_field * column
- cob_field * foreg
- cob_field * backg
- int type
- int occurs
- int attr

6.1.1 Detailed Description

Definition at line 67 of file screenio.h.

6.1.2 Member Data Documentation

6.1.2.1 int __cob_screen::attr

Definition at line 78 of file screenio.h.

6.1.2.2 cob_field* __cob_screen::backg

Definition at line 75 of file screenio.h.

6.1.2.3 cob_screen* __cob_screen::child

Definition at line 69 of file screenio.h.

6.1.2.4 cob_field* __cob_screen::column

Definition at line 73 of file screenio.h.

6.1.2.5 cob_field* __cob_screen::field

Definition at line 70 of file screenio.h.

6.1.2.6 cob_field* __cob_screen::foreg

Definition at line 74 of file screenio.h.

6.1.2.7 cob_field* __cob_screen::line

Definition at line 72 of file screenio.h.

6.1.2.8 cob_screen* __cob_screen::next

Definition at line 68 of file screenio.h.

6.1.2.9 int __cob_screen::occurs

Definition at line 77 of file screenio.h.

6.1.2.10 int __cob_screen::type

Definition at line 76 of file screenio.h.

6.1.2.11 cob_field* __cob_screen::value

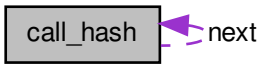
Definition at line 71 of file screenio.h.

The documentation for this struct was generated from the following file:

- libcob/screenio.h

6.2 call_hash Struct Reference

Collaboration diagram for call_hash:



Public Attributes

- struct call_hash * next
- const char * name
- void * func
- void * cancel

- size_t flag_is_active

6.2.1 Detailed Description

Definition at line 132 of file call.c.

6.2.2 Member Data Documentation

6.2.2.1 void* call_hash::cancel

Definition at line 136 of file call.c.

6.2.2.2 size_t call_hash::flag_is_active

Definition at line 137 of file call.c.

6.2.2.3 void* call_hash::func

Definition at line 135 of file call.c.

6.2.2.4 const char* call_hash::name

Definition at line 134 of file call.c.

6.2.2.5 struct call_hash* call_hash::next

Definition at line 133 of file call.c.

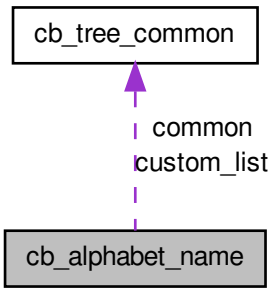
The documentation for this struct was generated from the following file:

- libcob/call.c

6.3 cb_alphabet_name Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_alphabet_name:



Public Attributes

- struct cb_tree_common common
- const char * name
- char * cname
- cb_tree custom_list
- enum cb_alphabet_name_type type
- int low_val_char
- int high_val_char

6.3.1 Detailed Description

Definition at line 328 of file tree.h.

6.3.2 Member Data Documentation

6.3.2.1 char* cb_alphabet_name::cname

Definition at line 331 of file tree.h.

6.3.2.2 struct cb_tree_common cb_alphabet_name::common

Definition at line 329 of file tree.h.

6.3.2.3 cb_tree cb_alphabet_name::custom_list

Definition at line 332 of file tree.h.

6.3.2.4 int cb_alphabet_name::high_val_char

Definition at line 335 of file tree.h.

6.3.2.5 int cb_alphabet_name::low_val_char

Definition at line 334 of file tree.h.

6.3.2.6 const char* cb_alphabet_name::name

Definition at line 330 of file tree.h.

6.3.2.7 enum cb_alphabet_name_type cb_alphabet_name::type

Definition at line 333 of file tree.h.

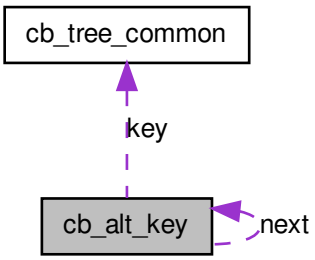
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.4 cb_alt_key Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_alt_key:



Public Attributes

- struct cb_alt_key * next
- cb_tree key
- int duplicates
- int offset

6.4.1 Detailed Description

Definition at line 592 of file tree.h.

6.4.2 Member Data Documentation

6.4.2.1 int cb_alt_key::duplicates

Definition at line 595 of file tree.h.

6.4.2.2 cb_tree cb_alt_key::key

Definition at line 594 of file tree.h.

6.4.2.3 struct cb_alt_key* cb_alt_key::next

Definition at line 593 of file tree.h.

6.4.2.4 int cb_alt_key::offset

Definition at line 596 of file tree.h.

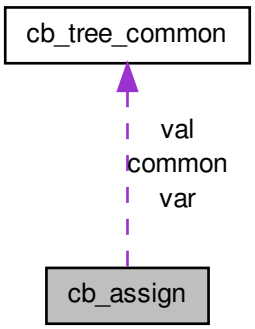
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.5 cb_assign Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_assign:



Public Attributes

- struct cb_tree_common common
- cb_tree var
- cb_tree val

6.5.1 Detailed Description

Definition at line 793 of file tree.h.

6.5.2 Member Data Documentation

6.5.2.1 struct cb_tree_common cb_assign::common

Definition at line 794 of file tree.h.

6.5.2.2 cb_tree cb_assign::val

Definition at line 796 of file tree.h.

6.5.2.3 cb_tree cb_assign::var

Definition at line 795 of file tree.h.

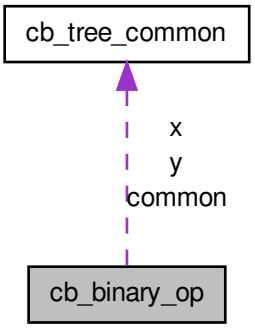
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.6 cb_binary_op Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_binary_op:



Public Attributes

- struct cb_tree_common common
- int op
- cb_tree x
- cb_tree y

6.6.1 Detailed Description

Definition at line 704 of file tree.h.

6.6.2 Member Data Documentation

6.6.2.1 struct cb_tree_common cb_binary_op::common

Definition at line 705 of file tree.h.

6.6.2.2 int cb_binary_op::op

Definition at line 706 of file tree.h.

6.6.2.3 cb_tree cb_binary_op::x

Definition at line 707 of file tree.h.

6.6.2.4 cb_tree cb_binary_op::y

Definition at line 708 of file tree.h.

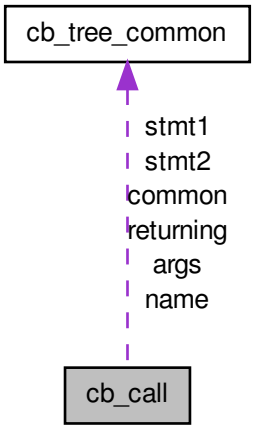
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.7 cb_call Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_call:



Public Attributes

- struct cb_tree_common common

- cb_tree name
- cb_tree args
- cb_tree stmt1
- cb_tree stmt2
- cb_tree returning
- int is_system

6.7.1 Detailed Description

Definition at line 974 of file tree.h.

6.7.2 Member Data Documentation

6.7.2.1 cb_tree cb_call::args

Definition at line 977 of file tree.h.

6.7.2.2 struct cb_tree_common cb_call::common

Definition at line 975 of file tree.h.

6.7.2.3 int cb_call::is_system

Definition at line 981 of file tree.h.

6.7.2.4 cb_tree cb_call::name

Definition at line 976 of file tree.h.

6.7.2.5 cb_tree cb_call::returning

Definition at line 980 of file tree.h.

6.7.2.6 cb_tree cb_call::stmt1

Definition at line 978 of file tree.h.

6.7.2.7 cb_tree cb_call::stmt2

Definition at line 979 of file tree.h.

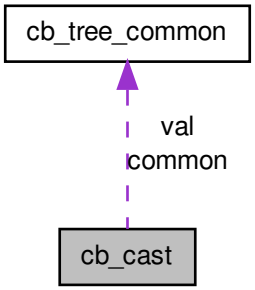
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.8 cb_cast Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_cast:



Public Attributes

- struct cb_tree_common common
- enum cb_cast_type type
- cb_tree val

6.8.1 Detailed Description

Definition at line 771 of file tree.h.

6.8.2 Member Data Documentation

6.8.2.1 struct cb_tree_common cb_cast::common

Definition at line 772 of file tree.h.

6.8.2.2 enum cb_cast_type cb_cast::type

Definition at line 773 of file tree.h.

6.8.2.3 cb_tree cb_cast::val

Definition at line 774 of file tree.h.

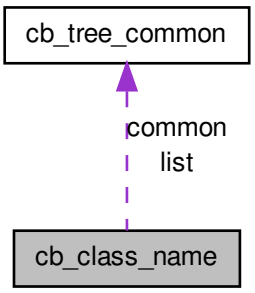
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.9 cb_class_name Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_class_name:



Public Attributes

- struct cb_tree_common common
- const char * name
- char * cname
- cb_tree list

6.9.1 Detailed Description

Definition at line 348 of file tree.h.

6.9.2 Member Data Documentation

6.9.2.1 char* cb_class_name::cname

Definition at line 351 of file tree.h.

6.9.2.2 struct cb_tree_common cb_class_name::common

Definition at line 349 of file tree.h.

6.9.2.3 cb_tree cb_class_name::list

Definition at line 352 of file tree.h.

6.9.2.4 const char* cb_class_name::name

Definition at line 350 of file tree.h.

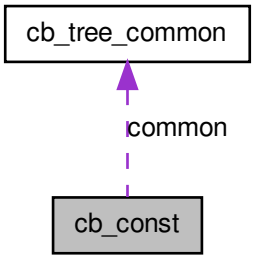
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.10 cb_const Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_const:



Public Attributes

- struct cb_tree_common common
- const char * val

6.10.1 Detailed Description

Definition at line 280 of file tree.h.

6.10.2 Member Data Documentation

6.10.2.1 struct cb_tree_common cb_const::common

Definition at line 281 of file tree.h.

6.10.2.2 const char* cb_const::val

Definition at line 282 of file tree.h.

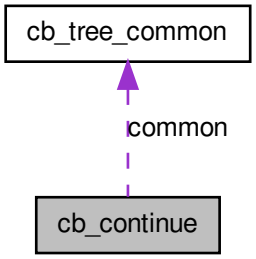
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.11 cb_continue Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_continue:



Public Attributes

- struct cb_tree_common common

6.11.1 Detailed Description

Definition at line 1090 of file tree.h.

6.11.2 Member Data Documentation

6.11.2.1 struct cb_tree_common cb_continue::common

Definition at line 1091 of file tree.h.

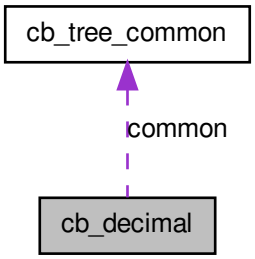
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.12 cb_decimal Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_decimal:



Public Attributes

- struct cb_tree_common common
- int id

6.12.1 Detailed Description

Definition at line 421 of file tree.h.

6.12.2 Member Data Documentation

6.12.2.1 struct cb_tree_common cb_decimal::common

Definition at line 422 of file tree.h.

6.12.2.2 int cb_decimal::id

Definition at line 423 of file tree.h.

The documentation for this struct was generated from the following file:

- cobc/tree.h

6.13 cb_exception Struct Reference

```
#include <cobc.h>
```

Public Attributes

- const char * name
- const int code
- int enable

6.13.1 Detailed Description

Definition at line 46 of file cobc.h.

6.13.2 Member Data Documentation

6.13.2.1 const int cb_exception::code

Definition at line 48 of file cobc.h.

6.13.2.2 int cb_exception::enable

Definition at line 49 of file cobc.h.

6.13.2.3 const char* cb_exception::name

Definition at line 47 of file cobc.h.

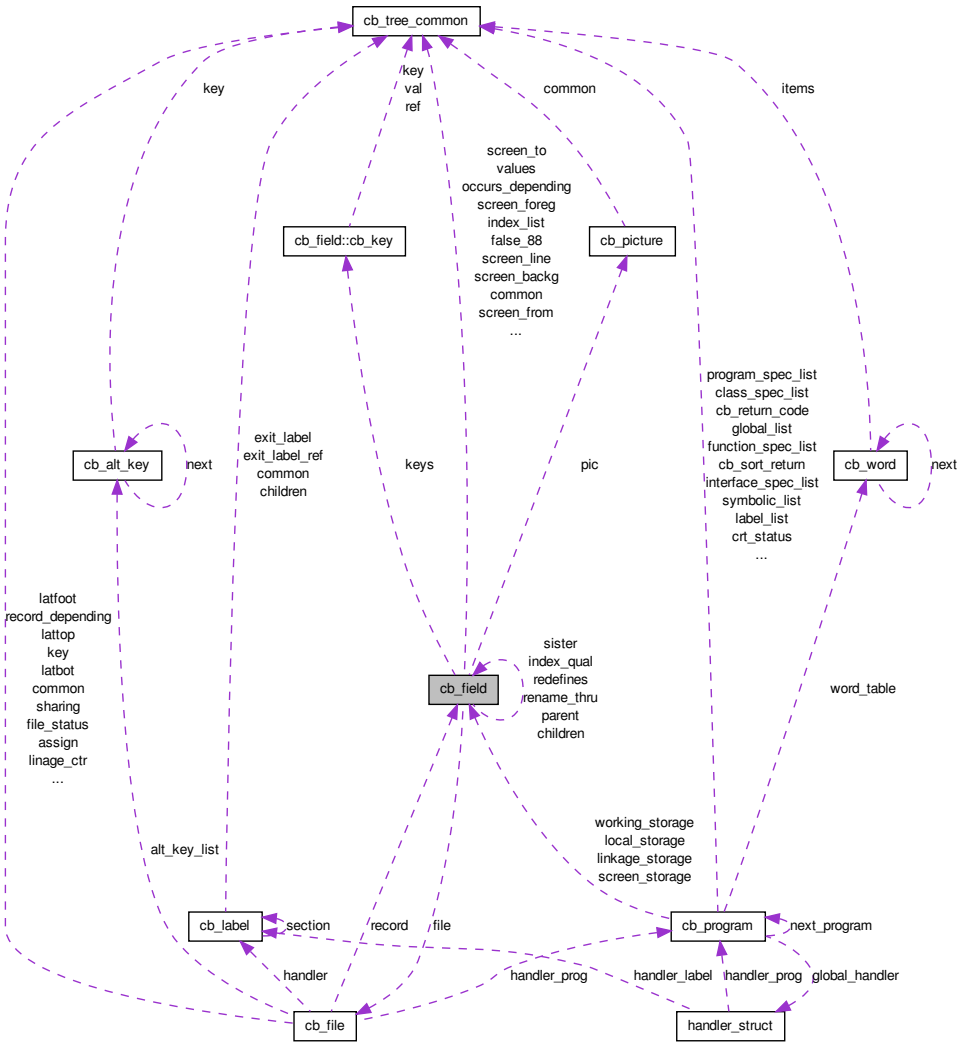
The documentation for this struct was generated from the following file:

- cobc/cobc.h

6.14 cb_field Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_field:



Classes

- struct cb_key

Public Attributes

- struct cb_tree_common common
- int id
- int storage_id
- const char * name
- const char * ename
- int size
- int memory_size
- int offset
- int level
- int occurs_min
- int occurs_max
- int indexes
- int count
- cb_tree occurs_depending
- enum cb_storage storage
- enum cb_usage usage
- cb_tree values
- cb_tree false_88
- cb_tree index_list
- struct cb_field * parent
- struct cb_field * children
- struct cb_field * sister
- struct cb_field * redefines
- struct cb_field * rename_thru
- struct cb_field * index_qual
- struct cb_file * file
- struct cb_field::cb_key * keys
- int nkeys
- int param_num
- struct cb_picture * pic
- cb_tree screen_line
- cb_tree screen_column
- cb_tree screen_from
- cb_tree screen_to
- cb_tree screen_foreg
- cb_tree screen_backg
- int screen_flag
- unsigned int flag_external: 1
- unsigned int flag_blank_zero: 1
- unsigned int flag_justified: 1

- unsigned int flag_sign_leading: 1
- unsigned int flag_sign_separate: 1
- unsigned int flag_synchronized: 1
- unsigned int flag_occurs: 1
- unsigned int flag_invalid: 1
- unsigned int flag_binary_swap: 1
- unsigned int flag_local: 1
- unsigned int flag_base: 1
- unsigned int flag_field: 1
- unsigned int flag_item_external: 1
- unsigned int flag_chained: 1
- unsigned int flag_real_binary: 1
- unsigned int flag_item_based: 1
- unsigned int flag_item_78: 1
- unsigned int flag_any_length: 1
- unsigned int flag_anylen_done: 1
- unsigned int flag_indexed_by: 1
- unsigned int flag_is_pointer: 1
- unsigned int flag_is_verified: 1
- unsigned int flag_is_global: 1
- unsigned int flag_is_c_long: 1
- unsigned int flag_is_pdiv_parm: 1
- unsigned int flag_local_alloced: 1
- unsigned int flag_no_init: 1
- unsigned int flag_spare: 5

6.14.1 Detailed Description

Definition at line 459 of file tree.h.

6.14.2 Member Data Documentation

6.14.2.1 struct cb_field* cb_field::children

Definition at line 480 of file tree.h.

6.14.2.2 struct cb_tree_common cb_field::common

Definition at line 460 of file tree.h.

6.14.2.3 int cb_field::count

Definition at line 472 of file tree.h.

6.14.2.4 const char* cb_field::ename

Definition at line 464 of file tree.h.

6.14.2.5 cb_tree cb_field::false_88

Definition at line 477 of file tree.h.

6.14.2.6 struct cb_file* cb_field::file

Definition at line 485 of file tree.h.

6.14.2.7 unsigned int cb_field::flag_any_length

Definition at line 521 of file tree.h.

6.14.2.8 unsigned int cb_field::flag_anylen_done

Definition at line 522 of file tree.h.

6.14.2.9 unsigned int cb_field::flag_base

Definition at line 514 of file tree.h.

6.14.2.10 unsigned int cb_field::flag_binary_swap

Definition at line 512 of file tree.h.

6.14.2.11 unsigned int cb_field::flag_blank_zero

Definition at line 505 of file tree.h.

6.14.2.12 unsigned int cb_field::flag_chained

Definition at line 517 of file tree.h.

6.14.2.13 unsigned int cb_field::flag_external

Definition at line 504 of file tree.h.

6.14.2.14 unsigned int cb_field::flag_field

Definition at line 515 of file tree.h.

6.14.2.15 unsigned int cb_field::flag_indexed_by

Definition at line 523 of file tree.h.

6.14.2.16 unsigned int cb_field::flag_invalid

Definition at line 511 of file tree.h.

6.14.2.17 unsigned int cb_field::flag_is_c_long

Definition at line 527 of file tree.h.

6.14.2.18 unsigned int cb_field::flag_is_global

Definition at line 526 of file tree.h.

6.14.2.19 unsigned int cb_field::flag_is_pdiv_parm

Definition at line 528 of file tree.h.

6.14.2.20 unsigned int cb_field::flag_is_pointer

Definition at line 524 of file tree.h.

6.14.2.21 unsigned int cb_field::flag_is_verified

Definition at line 525 of file tree.h.

6.14.2.22 unsigned int cb_field::flag_item_78

Definition at line 520 of file tree.h.

6.14.2.23 unsigned int cb_field::flag_item_based

Definition at line 519 of file tree.h.

6.14.2.24 unsigned int cb_field::flag_item_external

Definition at line 516 of file tree.h.

6.14.2.25 unsigned int cb_field::flag_justified

Definition at line 506 of file tree.h.

6.14.2.26 unsigned int cb_field::flag_local

Definition at line 513 of file tree.h.

6.14.2.27 unsigned int cb_field::flag_local_alloced

Definition at line 529 of file tree.h.

6.14.2.28 unsigned int cb_field::flag_no_init

Definition at line 530 of file tree.h.

6.14.2.29 unsigned int cb_field::flag_occurs

Definition at line 510 of file tree.h.

6.14.2.30 unsigned int cb_field::flag_real_binary

Definition at line 518 of file tree.h.

6.14.2.31 unsigned int cb_field::flag_sign_leading

Definition at line 507 of file tree.h.

6.14.2.32 unsigned int cb_field::flag_sign_separate

Definition at line 508 of file tree.h.

6.14.2.33 unsigned int cb_field::flag_spare

Definition at line 531 of file tree.h.

6.14.2.34 unsigned int cb_field::flag_synchronized

Definition at line 509 of file tree.h.

6.14.2.35 int cb_field::id

Definition at line 461 of file tree.h.

6.14.2.36 cb_tree cb_field::index_list

Definition at line 478 of file tree.h.

6.14.2.37 struct cb_field* cb_field::index_qual

Definition at line 484 of file tree.h.

6.14.2.38 int cb_field::indexes

Definition at line 471 of file tree.h.

6.14.2.39 struct cb_field::cb_key * cb_field::keys

6.14.2.40 int cb_field::level

Definition at line 468 of file tree.h.

6.14.2.41 int cb_field::memory_size

Definition at line 466 of file tree.h.

6.14.2.42 const char* cb_field::name

Definition at line 463 of file tree.h.

6.14.2.43 int cb_field::nkeys

Definition at line 492 of file tree.h.

6.14.2.44 cb_tree cb_field::occurs_depending

Definition at line 473 of file tree.h.

6.14.2.45 int cb_field::occurs_max

Definition at line 470 of file tree.h.

6.14.2.46 int cb_field::occurs_min

Definition at line 469 of file tree.h.

6.14.2.47 int cb_field::offset

Definition at line 467 of file tree.h.

6.14.2.48 int cb_field::param_num

Definition at line 493 of file tree.h.

6.14.2.49 struct cb_field* cb_field::parent

Definition at line 479 of file tree.h.

6.14.2.50 struct cb_picture* cb_field::pic

Definition at line 494 of file tree.h.

6.14.2.51 struct cb_field* cb_field::redefines

Definition at line 482 of file tree.h.

6.14.2.52 struct cb_field* cb_field::rename_thru

Definition at line 483 of file tree.h.

6.14.2.53 cb_tree cb_field::screen_backg

Definition at line 501 of file tree.h.

6.14.2.54 cb_tree cb_field::screen_column

Definition at line 497 of file tree.h.

6.14.2.55 int cb_field::screen_flag

Definition at line 502 of file tree.h.

6.14.2.56 cb_tree cb_field::screen_foreg

Definition at line 500 of file tree.h.

6.14.2.57 cb_tree cb_field::screen_from

Definition at line 498 of file tree.h.

6.14.2.58 cb_tree cb_field::screen_line

Definition at line 496 of file tree.h.

6.14.2.59 cb_tree cb_field::screen_to

Definition at line 499 of file tree.h.

6.14.2.60 struct cb_field* cb_field::sister

Definition at line 481 of file tree.h.

6.14.2.61 int cb_field::size

Definition at line 465 of file tree.h.

6.14.2.62 enum cb_storage cb_field::storage

Definition at line 474 of file tree.h.

6.14.2.63 int cb_field::storage_id

Definition at line 462 of file tree.h.

6.14.2.64 enum cb_usage cb_field::usage

Definition at line 475 of file tree.h.

6.14.2.65 cb_tree cb_field::values

Definition at line 476 of file tree.h.

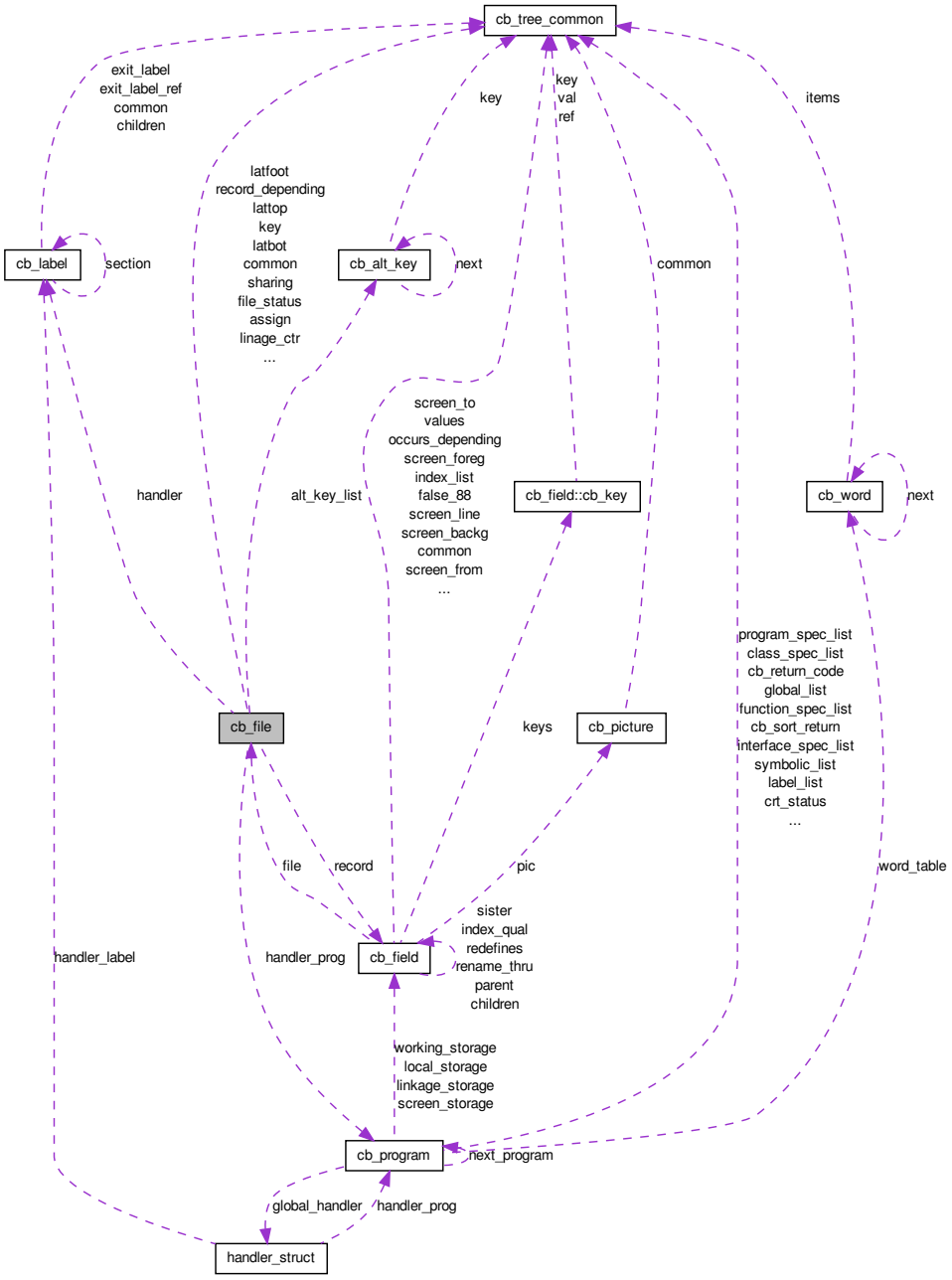
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.15 cb_file Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_file:



Public Attributes

- struct cb_tree_common common
- const char * name
- char * cname
- cb_tree assign
- cb_tree file_status
- cb_tree sharing
- cb_tree key
- struct cb_alt_key * alt_key_list
- struct cb_field * record
- cb_tree record_depending
- cb_tree linage
- cb_tree linage_ctr
- cb_tree latfoot
- cb_tree lattop
- cb_tree latbot
- struct cb_label * handler
- struct cb_program * handler_prog
- int record_min
- int record_max
- int optional
- int organization
- int access_mode
- int lock_mode
- int same_clause
- int finalized
- int external
- int special
- int external_assign
- int fileid_assign
- int global

6.15.1 Detailed Description

Definition at line 599 of file tree.h.

6.15.2 Member Data Documentation

6.15.2.1 int cb_file::access_mode

Definition at line 623 of file tree.h.

6.15.2.2 struct cb_alt_key* cb_file::alt_key_list

Definition at line 608 of file tree.h.

6.15.2.3 cb_tree cb_file::assign

Definition at line 604 of file tree.h.

6.15.2.4 char* cb_file::cname

Definition at line 602 of file tree.h.

6.15.2.5 struct cb_tree_common cb_file::common

Definition at line 600 of file tree.h.

6.15.2.6 int cb_file::external

Definition at line 627 of file tree.h.

6.15.2.7 int cb_file::external_assign

Definition at line 629 of file tree.h.

6.15.2.8 cb_tree cb_file::file_status

Definition at line 605 of file tree.h.

6.15.2.9 int cb_file::fileid_assign

Definition at line 630 of file tree.h.

6.15.2.10 int cb_file::finalized

Definition at line 626 of file tree.h.

6.15.2.11 int cb_file::global

Definition at line 631 of file tree.h.

6.15.2.12 struct cb_label* cb_file::handler

Definition at line 617 of file tree.h.

6.15.2.13 struct cb_program* cb_file::handler_prog

Definition at line 618 of file tree.h.

6.15.2.14 cb_tree cb_file::key

Definition at line 607 of file tree.h.

6.15.2.15 cb_tree cb_file::latbot

Definition at line 616 of file tree.h.

6.15.2.16 cb_tree cb_file::latfoot

Definition at line 614 of file tree.h.

6.15.2.17 cb_tree cb_file::lattop

Definition at line 615 of file tree.h.

6.15.2.18 cb_tree cb_file::linage

Definition at line 612 of file tree.h.

6.15.2.19 cb_tree cb_file::linage_ctr

Definition at line 613 of file tree.h.

6.15.2.20 int cb_file::lock_mode

Definition at line 624 of file tree.h.

6.15.2.21 const char* cb_file::name

Definition at line 601 of file tree.h.

6.15.2.22 int cb_file::optional

Definition at line 621 of file tree.h.

6.15.2.23 int cb_file::organization

Definition at line 622 of file tree.h.

6.15.2.24 struct cb_field* cb_file::record

Definition at line 610 of file tree.h.

6.15.2.25 cb_tree cb_file::record_depending

Definition at line 611 of file tree.h.

6.15.2.26 int cb_file::record_max

Definition at line 620 of file tree.h.

6.15.2.27 int cb_file::record_min

Definition at line 619 of file tree.h.

6.15.2.28 int cb_file::same_clause

Definition at line 625 of file tree.h.

6.15.2.29 cb_tree cb_file::sharing

Definition at line 606 of file tree.h.

6.15.2.30 int cb_file::special

Definition at line 628 of file tree.h.

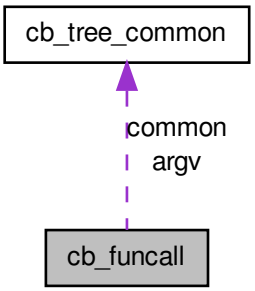
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.16 cb_funcall Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_funcall:



Public Attributes

- struct cb_tree_common common
- const char * name
- cb_tree argv [7]
- int argc
- int varcnt
- size_t screenptr

6.16.1 Detailed Description

Definition at line 725 of file tree.h.

6.16.2 Member Data Documentation

6.16.2.1 int cb_funcall::argc

Definition at line 729 of file tree.h.

6.16.2.2 cb_tree cb_funcall::argv[7]

Definition at line 728 of file tree.h.

6.16.2.3 struct cb_tree_common cb_funcall::common

Definition at line 726 of file tree.h.

6.16.2.4 const char* cb_funcall::name

Definition at line 727 of file tree.h.

6.16.2.5 size_t cb_funcall::screenptr

Definition at line 731 of file tree.h.

6.16.2.6 int cb_funcall::varcnt

Definition at line 730 of file tree.h.

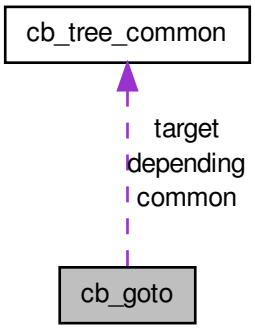
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.17 cb_goto Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_goto:



Public Attributes

- struct cb_tree_common common
- cb_tree target
- cb_tree depending

6.17.1 Detailed Description

Definition at line 995 of file tree.h.

6.17.2 Member Data Documentation

6.17.2.1 struct cb_tree_common cb_goto::common

Definition at line 996 of file tree.h.

6.17.2.2 cb_tree cb_goto::depending

Definition at line 998 of file tree.h.

6.17.2.3 cb_tree cb_goto::target

Definition at line 997 of file tree.h.

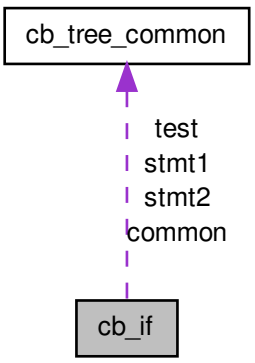
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.18 cb_if Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_if:



Public Attributes

- struct cb_tree_common common
- cb_tree test
- cb_tree stmt1
- cb_tree stmt2

6.18.1 Detailed Description

Definition at line 1011 of file tree.h.

6.18.2 Member Data Documentation

6.18.2.1 struct cb_tree_common cb_if::common

Definition at line 1012 of file tree.h.

6.18.2.2 cb_tree cb_if::stmt1

Definition at line 1014 of file tree.h.

6.18.2.3 cb_tree cb_if::stmt2

Definition at line 1015 of file tree.h.

6.18.2.4 cb_tree cb_if::test

Definition at line 1013 of file tree.h.

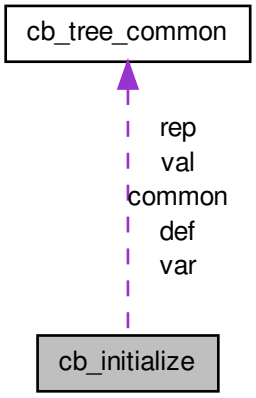
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.19 cb_initialize Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_initialize:



Public Attributes

- struct cb_tree_common common
- cb_tree var
- cb_tree val
- cb_tree rep
- cb_tree def
- int flag_statement

6.19.1 Detailed Description

Definition at line 932 of file tree.h.

6.19.2 Member Data Documentation

6.19.2.1 struct cb_tree_common cb_initialize::common

Definition at line 933 of file tree.h.

6.19.2.2 cb_tree cb_initialize::def

Definition at line 937 of file tree.h.

6.19.2.3 int cb_initialize::flag_statement

Definition at line 938 of file tree.h.

6.19.2.4 cb_tree cb_initialize::rep

Definition at line 936 of file tree.h.

6.19.2.5 cb_tree cb_initialize::val

Definition at line 935 of file tree.h.

6.19.2.6 cb_tree cb_initialize::var

Definition at line 934 of file tree.h.

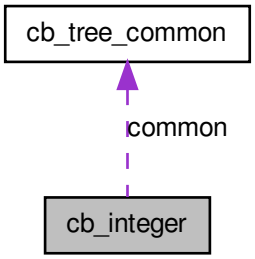
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.20 cb_integer Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_integer:



Public Attributes

- struct cb_tree_common common
- int val

6.20.1 Detailed Description

Definition at line 295 of file tree.h.

6.20.2 Member Data Documentation

6.20.2.1 struct cb_tree_common cb_integer::common

Definition at line 296 of file tree.h.

6.20.2.2 int cb_integer::val

Definition at line 297 of file tree.h.

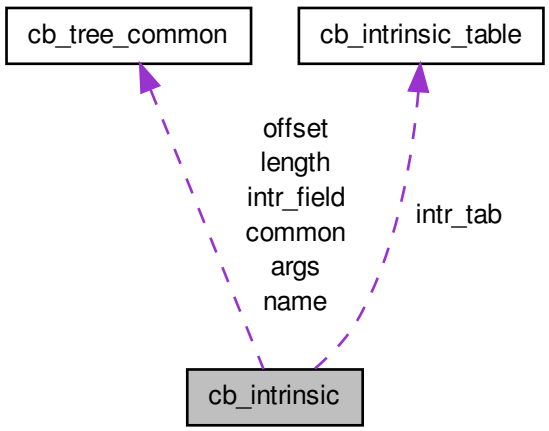
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.21 cb_intrinsic Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_intrinsic:



Public Attributes

- struct cb_tree_common common
- cb_tree name
- cb_tree args
- cb_tree intr_field
- struct cb_intrinsic_table * intr_tab
- cb_tree offset
- cb_tree length

6.21.1 Detailed Description

Definition at line 907 of file tree.h.

6.21.2 Member Data Documentation

6.21.2.1 cb_tree cb_intrinsic::args

Definition at line 910 of file tree.h.

6.21.2.2 struct cb_tree_common cb_intrinsic::common

Definition at line 908 of file tree.h.

6.21.2.3 cb_tree cb_intrinsic::intr_field

Definition at line 911 of file tree.h.

6.21.2.4 struct cb_intrinsic_table* cb_intrinsic::intr_tab

Definition at line 912 of file tree.h.

6.21.2.5 cb_tree cb_intrinsic::length

Definition at line 914 of file tree.h.

6.21.2.6 cb_tree cb_intrinsic::name

Definition at line 909 of file tree.h.

6.21.2.7 cb_tree cb_intrinsic::offset

Definition at line 913 of file tree.h.

The documentation for this struct was generated from the following file:

- cobc/tree.h

6.22 cb_intrinsic_table Struct Reference

`#include <tree.h>`

Public Attributes

- const char * name
- const int args
- const int implemented
- enum cb_intr_enum intr_enum
- const char * intr_routine
- enum cb_category category
- const unsigned int refmod

6.22.1 Detailed Description

Definition at line 897 of file tree.h.

6.22.2 Member Data Documentation

6.22.2.1 const int cb_intrinsic_table::args

Definition at line 899 of file tree.h.

6.22.2.2 enum cb_category cb_intrinsic_table::category

Definition at line 903 of file tree.h.

6.22.2.3 const int cb_intrinsic_table::implemented

Definition at line 900 of file tree.h.

6.22.2.4 enum cb_intr_enum cb_intrinsic_table::intr_enum

Definition at line 901 of file tree.h.

6.22.2.5 const char* cb_intrinsic_table::intr_routine

Definition at line 902 of file tree.h.

6.22.2.6 const char* cb_intrinsic_table::name

Definition at line 898 of file tree.h.

6.22.2.7 const unsigned int cb_intrinsic_table::refmod

Definition at line 904 of file tree.h.

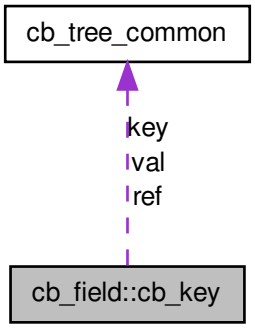
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.23 cb_field::cb_key Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_field::cb_key:



Public Attributes

- int dir
- cb_tree key
- cb_tree ref
- cb_tree val

6.23.1 Detailed Description

Definition at line 486 of file tree.h.

6.23.2 Member Data Documentation

6.23.2.1 int cb_field::cb_key::dir

Definition at line 487 of file tree.h.

6.23.2.2 cb_tree cb_field::cb_key::key

Definition at line 488 of file tree.h.

6.23.2.3 cb_tree cb_field::cb_key::ref

Definition at line 489 of file tree.h.

6.23.2.4 cb_tree cb_field::cb_key::val

Definition at line 490 of file tree.h.

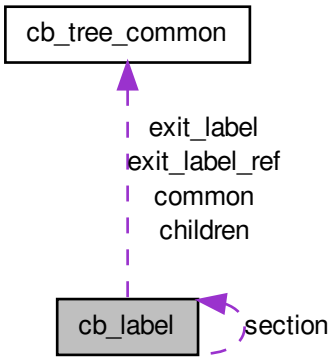
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.24 cb_label Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_label:



Public Attributes

- struct cb_tree_common common
- const unsigned char * name
- struct cb_label * section
- cb_tree exit_label
- cb_tree exit_label_ref
- cb_tree children
- const unsigned char * orig_name
- int id
- int is_section
- int is_entry
- unsigned char need_begin
- unsigned char need_return
- unsigned char is_global
- unsigned char spare

6.24.1 Detailed Description

Definition at line 561 of file tree.h.

6.24.2 Member Data Documentation

6.24.2.1 cb_tree cb_label::children

Definition at line 567 of file tree.h.

6.24.2.2 struct cb_tree_common cb_label::common

Definition at line 562 of file tree.h.

6.24.2.3 cb_tree cb_label::exit_label

Definition at line 565 of file tree.h.

6.24.2.4 cb_tree cb_label::exit_label_ref

Definition at line 566 of file tree.h.

6.24.2.5 int cb_label::id

Definition at line 569 of file tree.h.

6.24.2.6 int cb_label::is_entry

Definition at line 571 of file tree.h.

6.24.2.7 unsigned char cb_label::is_global

Definition at line 574 of file tree.h.

6.24.2.8 int cb_label::is_section

Definition at line 570 of file tree.h.

6.24.2.9 const unsigned char* cb_label::name

Definition at line 563 of file tree.h.

6.24.2.10 unsigned char cb_label::need_begin

Definition at line 572 of file tree.h.

6.24.2.11 unsigned char cb_label::need_return

Definition at line 573 of file tree.h.

6.24.2.12 const unsigned char* cb_label::orig_name

Definition at line 568 of file tree.h.

6.24.2.13 struct cb_label* cb_label::section

Definition at line 564 of file tree.h.

6.24.2.14 unsigned char cb_label::spare

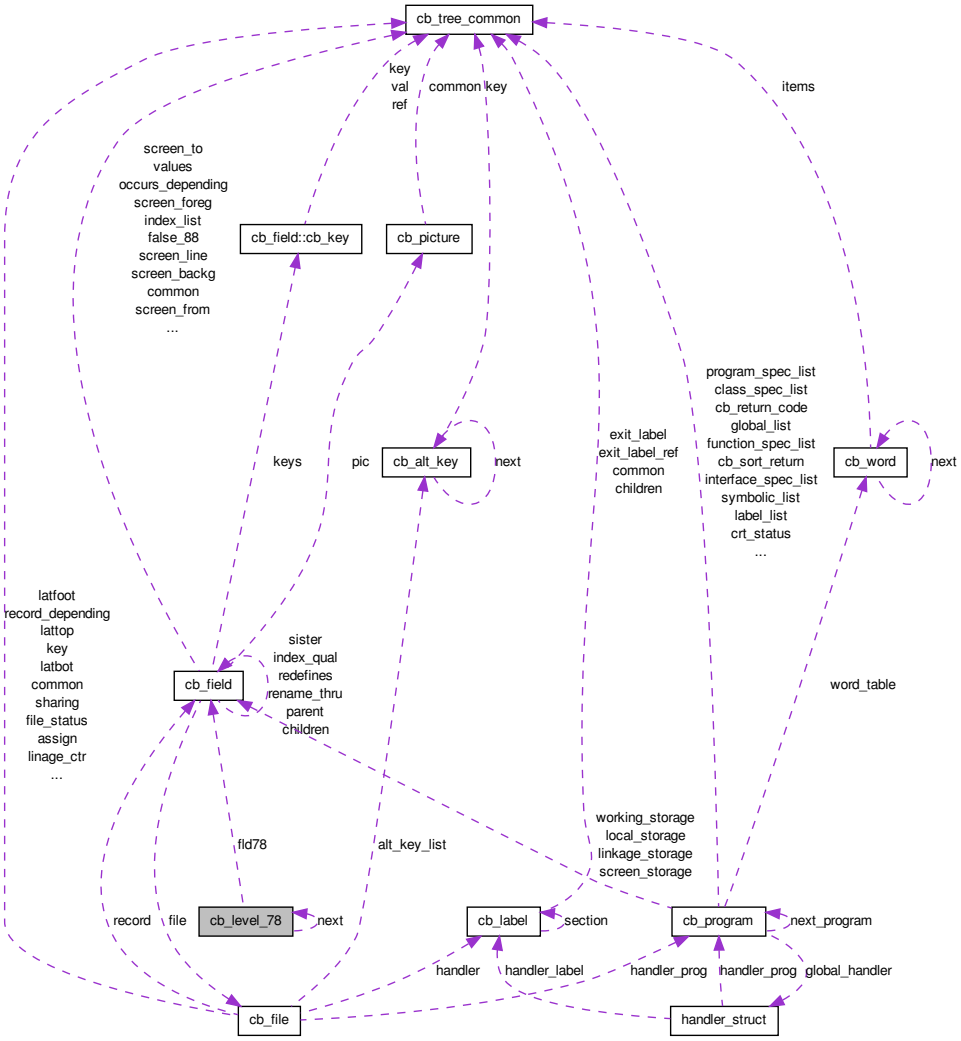
Definition at line 575 of file tree.h.

The documentation for this struct was generated from the following file:

- cobc/tree.h

6.25 cb_level_78 Struct Reference

Collaboration diagram for cb_level_78:



Public Attributes

- struct cb_level_78 * next
- struct cb_field * fld78

6.25.1 Detailed Description

Definition at line 901 of file scanner.c.

6.25.2 Member Data Documentation

6.25.2.1 struct cb_field* cb_level_78::fld78

Definition at line 903 of file scanner.c.

6.25.2.2 struct cb_level_78* cb_level_78::next

Definition at line 902 of file scanner.c.

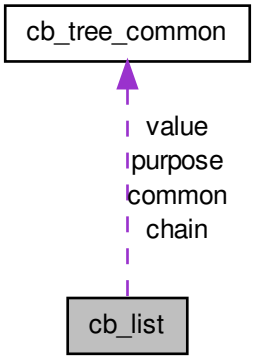
The documentation for this struct was generated from the following file:

- cobc/scanner.c

6.26 cb_list Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_list:



Public Attributes

- struct cb_tree_common common

- cb_tree purpose
- cb_tree value
- cb_tree chain
- int sizes

6.26.1 Detailed Description

Definition at line 1104 of file tree.h.

6.26.2 Member Data Documentation

6.26.2.1 cb_tree cb_list::chain

Definition at line 1108 of file tree.h.

6.26.2.2 struct cb_tree_common cb_list::common

Definition at line 1105 of file tree.h.

6.26.2.3 cb_tree cb_list::purpose

Definition at line 1106 of file tree.h.

6.26.2.4 int cb_list::sizes

Definition at line 1109 of file tree.h.

6.26.2.5 cb_tree cb_list::value

Definition at line 1107 of file tree.h.

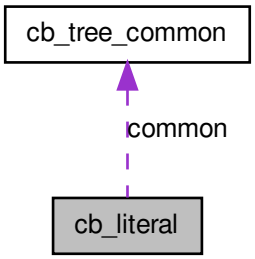
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.27 cb_literal Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_literal:



Public Attributes

- struct cb_tree_common common
- size_t size
- unsigned char * data
- signed char all
- signed char sign
- signed char scale
- signed char spare

6.27.1 Detailed Description

Definition at line 397 of file tree.h.

6.27.2 Member Data Documentation

6.27.2.1 signed char cb_literal::all

Definition at line 401 of file tree.h.

6.27.2.2 struct cb_tree_common cb_literal::common

Definition at line 398 of file tree.h.

6.27.2.3 unsigned char* cb_literal::data

Definition at line 400 of file tree.h.

6.27.2.4 signed char cb_literal::scale

Definition at line 403 of file tree.h.

6.27.2.5 signed char cb_literal::sign

Definition at line 402 of file tree.h.

6.27.2.6 size_t cb_literal::size

Definition at line 399 of file tree.h.

6.27.2.7 signed char cb_literal::spare

Definition at line 404 of file tree.h.

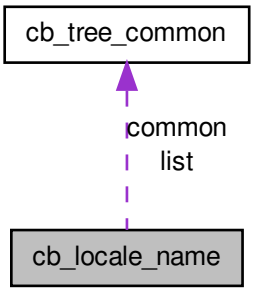
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.28 cb_locale_name Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_locale_name:



Public Attributes

- struct cb_tree_common common

- const char * name
- char * cname
- cb_tree list

6.28.1 Detailed Description

Definition at line 365 of file tree.h.

6.28.2 Member Data Documentation

6.28.2.1 char* cb_locale_name::cname

Definition at line 368 of file tree.h.

6.28.2.2 struct cb_tree_common cb_locale_name::common

Definition at line 366 of file tree.h.

6.28.2.3 cb_tree cb_locale_name::list

Definition at line 369 of file tree.h.

6.28.2.4 const char* cb_locale_name::name

Definition at line 367 of file tree.h.

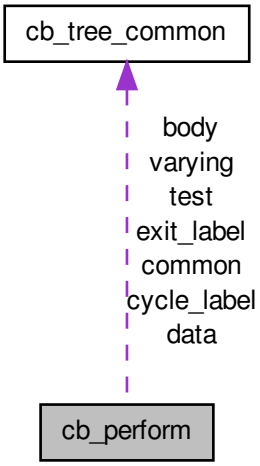
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.29 cb_perform Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_perform:



Public Attributes

- struct cb_tree_common common
- enum cb_perform_type type
- cb_tree test
- cb_tree body
- cb_tree data
- cb_tree varying
- cb_tree exit_label
- cb_tree cycle_label

6.29.1 Detailed Description

Definition at line 1044 of file tree.h.

6.29.2 Member Data Documentation

6.29.2.1 cb_tree cb_perform::body

Definition at line 1048 of file tree.h.

6.29.2.2 struct cb_tree_common cb_perform::common

Definition at line 1045 of file tree.h.

6.29.2.3 cb_tree cb_perform::cycle_label

Definition at line 1052 of file tree.h.

6.29.2.4 cb_tree cb_perform::data

Definition at line 1049 of file tree.h.

6.29.2.5 cb_tree cb_perform::exit_label

Definition at line 1051 of file tree.h.

6.29.2.6 cb_tree cb_perform::test

Definition at line 1047 of file tree.h.

6.29.2.7 enum cb_perform_type cb_perform::type

Definition at line 1046 of file tree.h.

6.29.2.8 cb_tree cb_perform::varying

Definition at line 1050 of file tree.h.

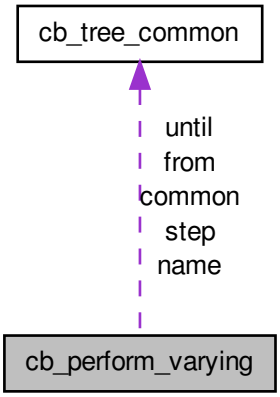
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.30 cb_perform_varying Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_perform_varying:



Public Attributes

- struct cb_tree_common common
- cb_tree name
- cb_tree from
- cb_tree step
- cb_tree until

6.30.1 Detailed Description

Definition at line 1036 of file tree.h.

6.30.2 Member Data Documentation

6.30.2.1 struct cb_tree_common cb_perform_varying::common

Definition at line 1037 of file tree.h.

6.30.2.2 cb_tree cb_perform_varying::from

Definition at line 1039 of file tree.h.

6.30.2.3 cb_tree cb_perform_varying::name

Definition at line 1038 of file tree.h.

6.30.2.4 cb_tree cb_perform_varying::step

Definition at line 1040 of file tree.h.

6.30.2.5 cb_tree cb_perform_varying::until

Definition at line 1041 of file tree.h.

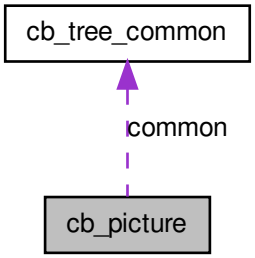
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.31 cb_picture Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_picture:



Public Attributes

- struct cb_tree_common common
- int size
- int lenstr
- char * orig
- char * str

- enum cb_category category
- unsigned char digits
- signed char scale
- unsigned char have_sign
- unsigned char spare

6.31.1 Detailed Description

Definition at line 436 of file tree.h.

6.31.2 Member Data Documentation

6.31.2.1 enum cb_category cb_picture::category

Definition at line 442 of file tree.h.

6.31.2.2 struct cb_tree_common cb_picture::common

Definition at line 437 of file tree.h.

6.31.2.3 unsigned char cb_picture::digits

Definition at line 443 of file tree.h.

6.31.2.4 unsigned char cb_picture::have_sign

Definition at line 445 of file tree.h.

6.31.2.5 int cb_picture::lenstr

Definition at line 439 of file tree.h.

6.31.2.6 char* cb_picture::orig

Definition at line 440 of file tree.h.

6.31.2.7 signed char cb_picture::scale

Definition at line 444 of file tree.h.

6.31.2.8 int cb_picture::size

Definition at line 438 of file tree.h.

6.31.2.9 unsigned char cb_picture::spare

Definition at line 446 of file tree.h.

6.31.2.10 char* cb_picture::str

Definition at line 441 of file tree.h.

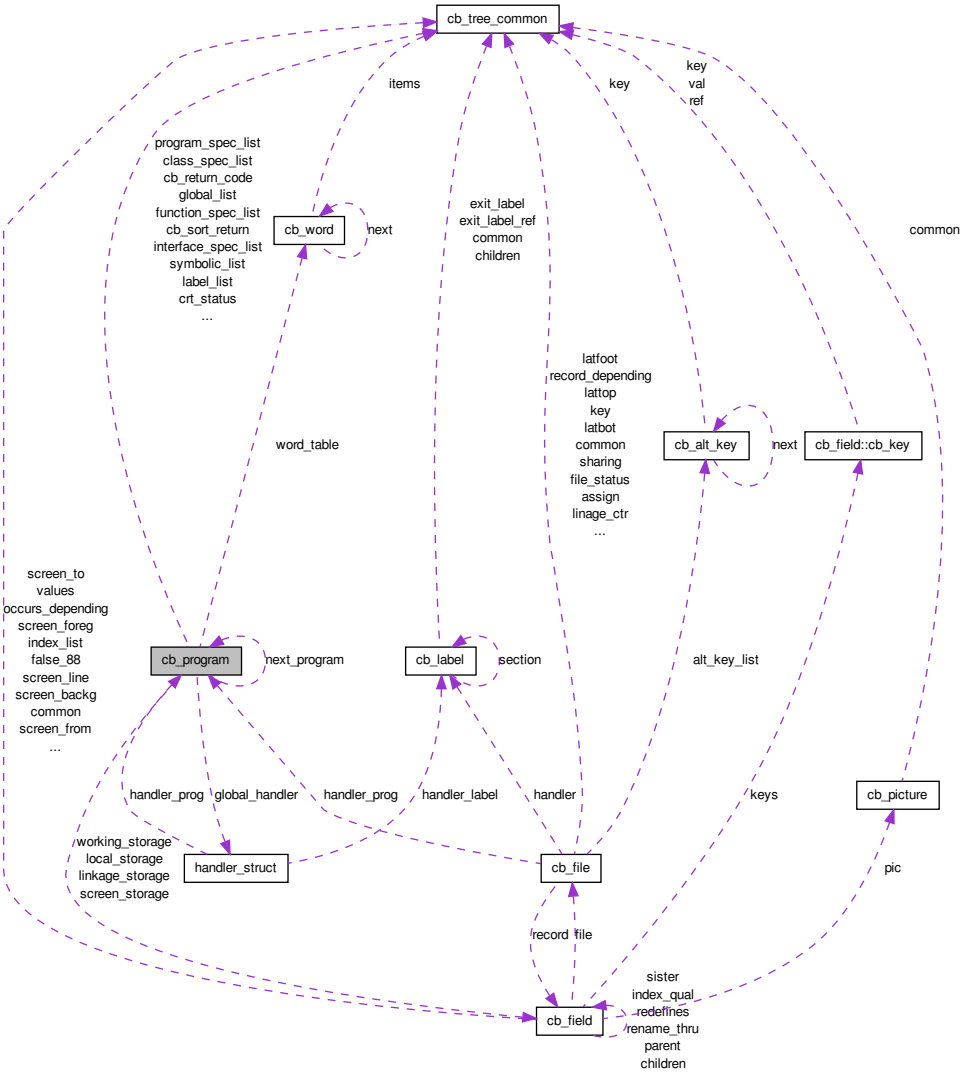
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.32 cb_program Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_program:



Public Attributes

- struct cb_program * next_program
- const char * program_id
- char * source_name
- char * orig_source_name
- FILE * local_storage_file
- char * local_storage_name

- cb_tree entry_list
- cb_tree file_list
- cb_tree exec_list
- cb_tree label_list
- cb_tree reference_list
- cb_tree alphabet_name_list
- cb_tree class_name_list
- cb_tree parameter_list
- cb_tree locale_list
- cb_tree symbolic_list
- cb_tree global_list
- cb_tree cb_return_code
- cb_tree cb_sort_return
- cb_tree cb_call_params
- cb_tree class_spec_list
- cb_tree interface_spec_list
- cb_tree function_spec_list
- cb_tree program_spec_list
- cb_tree property_spec_list
- struct cb_field * working_storage
- struct cb_field * local_storage
- struct cb_field * linkage_storage
- struct cb_field * screen_storage
- cb_tree local_file_list
- cb_tree global_file_list
- struct handler_struct global_handler [5]
- cb_tree collating_sequence
- cb_tree cursor_pos
- cb_tree crt_status
- cb_tree returning
- struct cb_word * word_table [CB_WORD_HASH_SIZE]
- int loop_counter
- int decimal_index
- int decimal_index_max
- unsigned char decimal_point
- unsigned char currency_symbol
- unsigned char numeric_separator
- unsigned char nested_level
- unsigned char flag_main
- unsigned char flag_common
- unsigned char flag_initial
- unsigned char flag_recursive
- unsigned char flag_screen
- unsigned char flag_validated
- unsigned char flag_chained
- unsigned char flag_global_use

- unsigned char gen_decset
- unsigned char gen_udecset
- unsigned char gen_ptrmanip
- unsigned char gen_file_error
- unsigned char prog_type
- unsigned char spare [3]

6.32.1 Detailed Description

Definition at line 1154 of file tree.h.

6.32.2 Member Data Documentation

6.32.2.1 cb_tree cb_program::alphabet_name_list

Definition at line 1167 of file tree.h.

6.32.2.2 cb_tree cb_program::cb_call_params

Definition at line 1175 of file tree.h.

6.32.2.3 cb_tree cb_program::cb_return_code

Definition at line 1173 of file tree.h.

6.32.2.4 cb_tree cb_program::cb_sort_return

Definition at line 1174 of file tree.h.

6.32.2.5 cb_tree cb_program::class_name_list

Definition at line 1168 of file tree.h.

6.32.2.6 cb_tree cb_program::class_spec_list

Definition at line 1176 of file tree.h.

6.32.2.7 cb_tree cb_program::collating_sequence

Definition at line 1188 of file tree.h.

6.32.2.8 cb_tree cb_program::crt_status

Definition at line 1190 of file tree.h.

6.32.2.9 unsigned char cb_program::currency_symbol

Definition at line 1198 of file tree.h.

6.32.2.10 cb_tree cb_program::cursor_pos

Definition at line 1189 of file tree.h.

6.32.2.11 int cb_program::decimal_index

Definition at line 1195 of file tree.h.

6.32.2.12 int cb_program::decimal_index_max

Definition at line 1196 of file tree.h.

6.32.2.13 unsigned char cb_program::decimal_point

Definition at line 1197 of file tree.h.

6.32.2.14 cb_tree cb_program::entry_list

Definition at line 1162 of file tree.h.

6.32.2.15 cb_tree cb_program::exec_list

Definition at line 1164 of file tree.h.

6.32.2.16 cb_tree cb_program::file_list

Definition at line 1163 of file tree.h.

6.32.2.17 unsigned char cb_program::flag_chained

Definition at line 1207 of file tree.h.

6.32.2.18 unsigned char cb_program::flag_common

Definition at line 1202 of file tree.h.

6.32.2.19 unsigned char cb_program::flag_global_use

Definition at line 1208 of file tree.h.

6.32.2.20 unsigned char cb_program::flag_initial

Definition at line 1203 of file tree.h.

6.32.2.21 unsigned char cb_program::flag_main

Definition at line 1201 of file tree.h.

6.32.2.22 unsigned char cb_program::flag_recursive

Definition at line 1204 of file tree.h.

6.32.2.23 unsigned char cb_program::flag_screen

Definition at line 1205 of file tree.h.

6.32.2.24 unsigned char cb_program::flag_validated

Definition at line 1206 of file tree.h.

6.32.2.25 cb_tree cb_program::function_spec_list

Definition at line 1178 of file tree.h.

6.32.2.26 unsigned char cb_program::gen_decset

Definition at line 1209 of file tree.h.

6.32.2.27 unsigned char cb_program::gen_file_error

Definition at line 1212 of file tree.h.

6.32.2.28 unsigned char cb_program::gen_ptrmanip

Definition at line 1211 of file tree.h.

6.32.2.29 unsigned char cb_program::gen_udecset

Definition at line 1210 of file tree.h.

6.32.2.30 cb_tree cb_program::global_file_list

Definition at line 1186 of file tree.h.

6.32.2.31 struct handler_struct cb_program::global_handler[5]

Definition at line 1187 of file tree.h.

6.32.2.32 cb_tree cb_program::global_list

Definition at line 1172 of file tree.h.

6.32.2.33 cb_tree cb_program::interface_spec_list

Definition at line 1177 of file tree.h.

6.32.2.34 cb_tree cb_program::label_list

Definition at line 1165 of file tree.h.

6.32.2.35 struct cb_field* cb_program::linkage_storage

Definition at line 1183 of file tree.h.

6.32.2.36 cb_tree cb_program::local_file_list

Definition at line 1185 of file tree.h.

6.32.2.37 struct cb_field* cb_program::local_storage

Definition at line 1182 of file tree.h.

6.32.2.38 FILE* cb_program::local_storage_file

Definition at line 1160 of file tree.h.

6.32.2.39 char* cb_program::local_storage_name

Definition at line 1161 of file tree.h.

6.32.2.40 cb_tree cb_program::locale_list

Definition at line 1170 of file tree.h.

6.32.2.41 int cb_program::loop_counter

Definition at line 1194 of file tree.h.

6.32.2.42 unsigned char cb_program::nested_level

Definition at line 1200 of file tree.h.

6.32.2.43 struct cb_program* cb_program::next_program

Definition at line 1156 of file tree.h.

6.32.2.44 unsigned char cb_program::numeric_separator

Definition at line 1199 of file tree.h.

6.32.2.45 char* cb_program::orig_source_name

Definition at line 1159 of file tree.h.

6.32.2.46 cb_tree cb_program::parameter_list

Definition at line 1169 of file tree.h.

6.32.2.47 unsigned char cb_program::prog_type

Definition at line 1213 of file tree.h.

6.32.2.48 const char* cb_program::program_id

Definition at line 1157 of file tree.h.

6.32.2.49 cb_tree cb_program::program_spec_list

Definition at line 1179 of file tree.h.

6.32.2.50 cb_tree cb_program::property_spec_list

Definition at line 1180 of file tree.h.

6.32.2.51 cb_tree cb_program::reference_list

Definition at line 1166 of file tree.h.

6.32.2.52 cb_tree cb_program::returning

Definition at line 1191 of file tree.h.

6.32.2.53 struct cb_field* cb_program::screen_storage

Definition at line 1184 of file tree.h.

6.32.2.54 char* cb_program::source_name

Definition at line 1158 of file tree.h.

6.32.2.55 unsigned char cb_program::spare[3]

Definition at line 1214 of file tree.h.

6.32.2.56 cb_tree cb_program::symbolic_list

Definition at line 1171 of file tree.h.

6.32.2.57 struct cb_word* cb_program::word_table[CB_WORD_HASH_SIZE]

Definition at line 1192 of file tree.h.

6.32.2.58 struct cb_field* cb_program::working_storage

Definition at line 1181 of file tree.h.

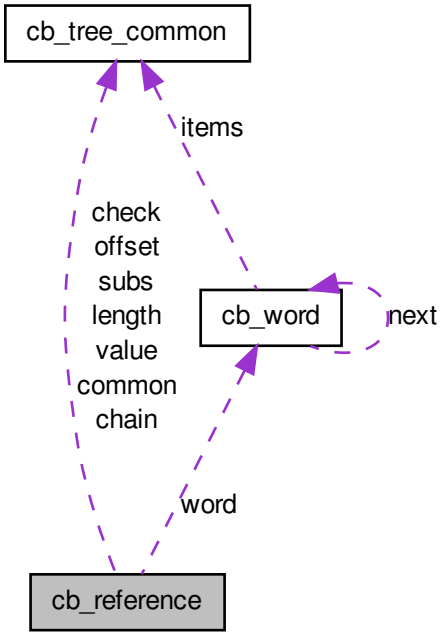
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.33 cb_reference Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_reference:



Public Attributes

- struct cb_tree_common common
- struct cb_word * word
- enum cb_operand_type type

- cb_tree value
- cb_tree subs
- cb_tree offset
- cb_tree length
- cb_tree check
- cb_tree chain
- int all

6.33.1 Detailed Description

Definition at line 656 of file tree.h.

6.33.2 Member Data Documentation

6.33.2.1 int cb_reference::all

Definition at line 666 of file tree.h.

6.33.2.2 cb_tree cb_reference::chain

Definition at line 665 of file tree.h.

6.33.2.3 cb_tree cb_reference::check

Definition at line 664 of file tree.h.

6.33.2.4 struct cb_tree_common cb_reference::common

Definition at line 657 of file tree.h.

6.33.2.5 cb_tree cb_reference::length

Definition at line 663 of file tree.h.

6.33.2.6 cb_tree cb_reference::offset

Definition at line 662 of file tree.h.

6.33.2.7 cb_tree cb_reference::subs

Definition at line 661 of file tree.h.

6.33.2.8 enum cb_operand_type cb_reference::type

Definition at line 659 of file tree.h.

6.33.2.9 cb_tree cb_reference::value

Definition at line 660 of file tree.h.

6.33.2.10 struct cb_word* cb_reference::word

Definition at line 658 of file tree.h.

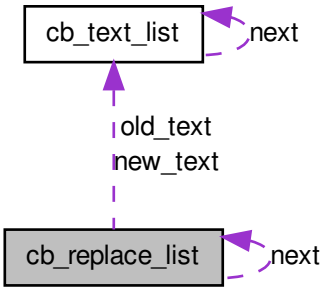
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.34 cb_replace_list Struct Reference

`#include <cobc.h>`

Collaboration diagram for cb_replace_list:



Public Attributes

- struct cb_text_list * old_text
- struct cb_text_list * new_text
- struct cb_replace_list * next

6.34.1 Detailed Description

Definition at line 71 of file cobc.h.

6.34.2 Member Data Documentation

6.34.2.1 struct cb_text_list* cb_replace_list::new_text

Definition at line 73 of file cobc.h.

6.34.2.2 struct cb_replace_list* cb_replace_list::next

Definition at line 74 of file cobc.h.

6.34.2.3 struct cb_text_list* cb_replace_list::old_text

Definition at line 72 of file cobc.h.

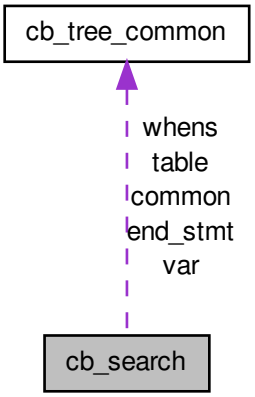
The documentation for this struct was generated from the following file:

- cobc/cobc.h

6.35 cb_search Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_search:



Public Attributes

- struct cb_tree_common common
- int flag_all
- cb_tree table
- cb_tree var
- cb_tree end_stmt
- cb_tree whens

6.35.1 Detailed Description

Definition at line 951 of file tree.h.

6.35.2 Member Data Documentation

6.35.2.1 struct cb_tree_common cb_search::common

Definition at line 952 of file tree.h.

6.35.2.2 cb_tree cb_search::end_stmt

Definition at line 956 of file tree.h.

6.35.2.3 int cb_search::flag_all

Definition at line 953 of file tree.h.

6.35.2.4 cb_tree cb_search::table

Definition at line 954 of file tree.h.

6.35.2.5 cb_tree cb_search::var

Definition at line 955 of file tree.h.

6.35.2.6 cb_tree cb_search::whens

Definition at line 957 of file tree.h.

The documentation for this struct was generated from the following file:

- cobc/tree.h

6.36 cb_statement Struct Reference

```
#include <tree.h>
```

Collaboration diagram for cb_statement:

Public Attributes

- struct cb_tree_common common
- const char * name
- cb_tree body
- cb_tree file
- cb_tree handler1
- cb_tree handler2
- cb_tree handler3
- cb_tree null_check
- int handler_id
- int need_terminator

6.36.1 Detailed Description

Definition at line 1067 of file tree.h.

6.36.2 Member Data Documentation

6.36.2.1 cb_tree cb_statement::body

Definition at line 1070 of file tree.h.

6.36.2.2 struct cb_tree_common cb_statement::common

Definition at line 1068 of file tree.h.

6.36.2.3 cb_tree cb_statement::file

Definition at line 1071 of file tree.h.

6.36.2.4 cb_tree cb_statement::handler1

Definition at line 1072 of file tree.h.

6.36.2.5 cb_tree cb_statement::handler2

Definition at line 1073 of file tree.h.

6.36.2.6 cb_tree cb_statement::handler3

Definition at line 1074 of file tree.h.

6.36.2.7 int cb_statement::handler_id

Definition at line 1076 of file tree.h.

6.36.2.8 const char* cb_statement::name

Definition at line 1069 of file tree.h.

6.36.2.9 int cb_statement::need_terminator

Definition at line 1077 of file tree.h.

6.36.2.10 cb_tree cb_statement::null_check

Definition at line 1075 of file tree.h.

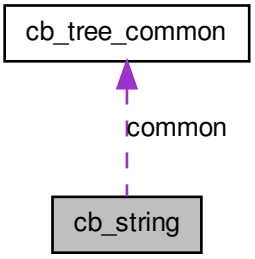
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.37 cb_string Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_string:



Public Attributes

- struct cb_tree_common common
- size_t size
- const unsigned char * data

6.37.1 Detailed Description

Definition at line 310 of file tree.h.

6.37.2 Member Data Documentation

6.37.2.1 struct cb_tree_common cb_string::common

Definition at line 311 of file tree.h.

6.37.2.2 const unsigned char* cb_string::data

Definition at line 313 of file tree.h.

6.37.2.3 size_t cb_string::size

Definition at line 312 of file tree.h.

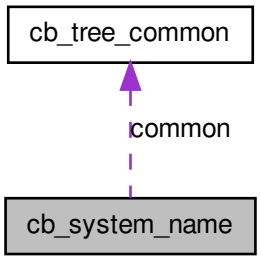
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.38 cb_system_name Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_system_name:



Public Attributes

- struct cb_tree_common common
- enum cb_system_name_category category
- int token

6.38.1 Detailed Description

Definition at line 381 of file tree.h.

6.38.2 Member Data Documentation

6.38.2.1 enum cb_system_name_category cb_system_name::category

Definition at line 383 of file tree.h.

6.38.2.2 struct cb_tree_common cb_system_name::common

Definition at line 382 of file tree.h.

6.38.2.3 int cb_system_name::token

Definition at line 384 of file tree.h.

The documentation for this struct was generated from the following file:

- cobc/tree.h

6.39 cb_text_list Struct Reference

`#include <cobc.h>`

Collaboration diagram for cb_text_list:

Public Attributes

- const char * text
- struct cb_text_list * next

6.39.1 Detailed Description

Definition at line 66 of file cobc.h.

6.39.2 Member Data Documentation

6.39.2.1 struct cb_text_list* cb_text_list::next

Definition at line 68 of file cobc.h.

6.39.2.2 const char* cb_text_list::text

Definition at line 67 of file cobc.h.

The documentation for this struct was generated from the following file:

- cobc/cobc.h

6.40 cb_tree_common Struct Reference

`#include <tree.h>`

Public Attributes

- enum cb_tag tag
- enum cb_category category
- unsigned char * source_file
- int source_line

6.40.1 Detailed Description

Definition at line 210 of file tree.h.

6.40.2 Member Data Documentation

6.40.2.1 enum cb_category cb_tree_common::category

Definition at line 212 of file tree.h.

6.40.2.2 unsigned char* cb_tree_common::source_file

Definition at line 213 of file tree.h.

6.40.2.3 int cb_tree_common::source_line

Definition at line 214 of file tree.h.

6.40.2.4 enum cb_tag cb_tree_common::tag

Definition at line 211 of file tree.h.

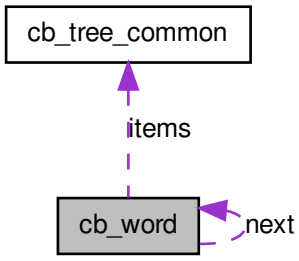
The documentation for this struct was generated from the following file:

- cobc/tree.h

6.41 cb_word Struct Reference

`#include <tree.h>`

Collaboration diagram for cb_word:



Public Attributes

- struct cb_word * next
- const char * name
- cb_tree items
- int count
- int error

6.41.1 Detailed Description

Definition at line 648 of file tree.h.

6.41.2 Member Data Documentation

6.41.2.1 int cb_word::count

Definition at line 652 of file tree.h.

6.41.2.2 int cb_word::error

Definition at line 653 of file tree.h.

6.41.2.3 cb_tree cb_word::items

Definition at line 651 of file tree.h.

6.41.2.4 const char* cb_word::name

Definition at line 650 of file tree.h.

6.41.2.5 struct cb_word* cb_word::next

Definition at line 649 of file tree.h.

The documentation for this struct was generated from the following file:

- cobc/tree.h

6.42 cob_alloc_cache Struct Reference

Collaboration diagram for cob_alloc_cache:

Public Attributes

- struct cob_alloc_cache * next
- void * cob_pointer
- size_t size

6.42.1 Detailed Description

Definition at line 69 of file common.c.

6.42.2 Member Data Documentation

6.42.2.1 void* cob_alloc_cache::cob_pointer

Definition at line 71 of file common.c.

6.42.2.2 struct cob_alloc_cache* cob_alloc_cache::next

Definition at line 70 of file common.c.

6.42.2.3 size_t cob_alloc_cache::size

Definition at line 72 of file common.c.

The documentation for this struct was generated from the following file:

- libcob/common.c

6.43 cob_decimal Struct Reference

```
#include <numeric.h>
```

Public Attributes

- mpz_t value
- int scale

6.43.1 Detailed Description

Definition at line 36 of file numeric.h.

6.43.2 Member Data Documentation

6.43.2.1 int cob_decimal::scale

Definition at line 38 of file numeric.h.

6.43.2.2 mpz_t cob_decimal::value

Definition at line 37 of file numeric.h.

The documentation for this struct was generated from the following file:

- libcob/numeric.h

6.44 cob_exception Struct Reference

Public Attributes

- const char * name
- const int code
- const int critical

6.44.1 Detailed Description

Definition at line 63 of file common.c.

6.44.2 Member Data Documentation

6.44.2.1 const int cob_exception::code

Definition at line 65 of file common.c.

6.44.2.2 const int cob_exception::critical

Definition at line 66 of file common.c.

6.44.2.3 const char* cob_exception::name

Definition at line 64 of file common.c.

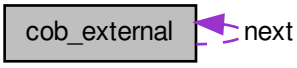
The documentation for this struct was generated from the following file:

- libcob/common.c

6.45 cob_external Struct Reference

`#include <common.h>`

Collaboration diagram for cob_external:



Public Attributes

- struct cob_external * next
- char * ext_alloc
- char * ename
- int esize

6.45.1 Detailed Description

Definition at line 98 of file common.h.

6.45.2 Member Data Documentation

6.45.2.1 char* cob_external::ename

Definition at line 101 of file common.h.

6.45.2.2 int cob_external::esize

Definition at line 102 of file common.h.

6.45.2.3 char* cob_external::ext_alloc

Definition at line 100 of file common.h.

6.45.2.4 struct cob_external* cob_external::next

Definition at line 99 of file common.h.

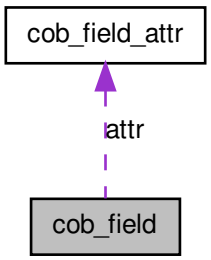
The documentation for this struct was generated from the following file:

- libcob/common.h

6.46 cob_field Struct Reference

```
#include <common.h>
```

Collaboration diagram for cob_field:



Public Attributes

- size_t size
- unsigned char * data
- const cob_field_attr * attr

6.46.1 Detailed Description

Definition at line 165 of file common.h.

6.46.2 Member Data Documentation

6.46.2.1 const cob_field_attr* cob_field::attr

Definition at line 168 of file common.h.

6.46.2.2 unsigned char* cob_field::data

Definition at line 167 of file common.h.

6.46.2.3 size_t cob_field::size

Definition at line 166 of file common.h.

The documentation for this struct was generated from the following file:

- libcob/common.h

6.47 cob_field_attr Struct Reference

`#include <common.h>`

Public Attributes

- unsigned char type
- unsigned char digits
- signed char scale
- unsigned char flags
- const char * pic

6.47.1 Detailed Description

Definition at line 155 of file common.h.

6.47.2 Member Data Documentation

6.47.2.1 unsigned char cob_field_attr::digits

Definition at line 157 of file common.h.

6.47.2.2 unsigned char cob_field_attr::flags

Definition at line 159 of file common.h.

6.47.2.3 const char* cob_field_attr::pic

Definition at line 160 of file common.h.

6.47.2.4 signed char cob_field_attr::scale

Definition at line 158 of file common.h.

6.47.2.5 unsigned char cob_field_attr::type

Definition at line 156 of file common.h.

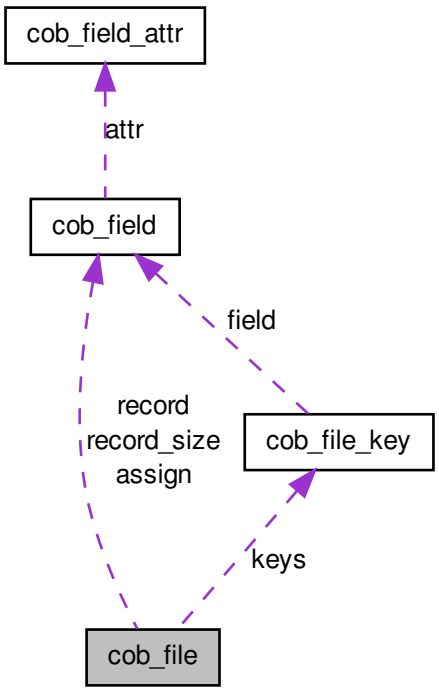
The documentation for this struct was generated from the following file:

- libcob/common.h

6.48 cob_file Struct Reference

`#include <fileio.h>`

Collaboration diagram for cob_file:



Public Attributes

- const char * select_name
- unsigned char * file_status
- cob_field * assign
- cob_field * record
- cob_field * record_size
- struct cob_file_key * keys
- void * file
- void * linorkeyptr
- const unsigned char * sort_collating
- void * extfh_ptr

- size_t record_min
- size_t record_max
- size_t nkeys
- char organization
- char access_mode
- char lock_mode
- char open_mode
- char flag_optional
- char last_open_mode
- char special
- char flag_nonexistent
- char flag_end_of_file
- char flag_begin_of_file
- char flag_first_read
- char flag_read_done
- char flag_select_features
- char flag_needs_nl
- char flag_needs_top
- char file_version

6.48.1 Detailed Description

Definition at line 171 of file fileio.h.

6.48.2 Member Data Documentation

6.48.2.1 char cob_file::access_mode

Definition at line 186 of file fileio.h.

6.48.2.2 cob_field* cob_file::assign

Definition at line 174 of file fileio.h.

6.48.2.3 void* cob_file::extfh_ptr

Definition at line 181 of file fileio.h.

6.48.2.4 void* cob_file::file

Definition at line 178 of file fileio.h.

6.48.2.5 unsigned char* cob_file::file_status

Definition at line 173 of file fileio.h.

6.48.2.6 char cob_file::file_version

Definition at line 200 of file fileio.h.

6.48.2.7 char cob_file::flag_begin_of_file

Definition at line 194 of file fileio.h.

6.48.2.8 char cob_file::flag_end_of_file

Definition at line 193 of file fileio.h.

6.48.2.9 char cob_file::flag_first_read

Definition at line 195 of file fileio.h.

6.48.2.10 char cob_file::flag_needs_nl

Definition at line 198 of file fileio.h.

6.48.2.11 char cob_file::flag_needs_top

Definition at line 199 of file fileio.h.

6.48.2.12 char cob_file::flag_nonexistent

Definition at line 192 of file fileio.h.

6.48.2.13 char cob_file::flag_optional

Definition at line 189 of file fileio.h.

6.48.2.14 char cob_file::flag_read_done

Definition at line 196 of file fileio.h.

6.48.2.15 char cob_file::flag_select_features

Definition at line 197 of file fileio.h.

6.48.2.16 struct cob_file_key* cob_file::keys

Definition at line 177 of file fileio.h.

6.48.2.17 char cob_file::last_open_mode

Definition at line 190 of file fileio.h.

6.48.2.18 void* cob_file::linorkeyptr

Definition at line 179 of file fileio.h.

6.48.2.19 char cob_file::lock_mode

Definition at line 187 of file fileio.h.

6.48.2.20 size_t cob_file::nkeys

Definition at line 184 of file fileio.h.

6.48.2.21 char cob_file::open_mode

Definition at line 188 of file fileio.h.

6.48.2.22 char cob_file::organization

Definition at line 185 of file fileio.h.

6.48.2.23 cob_field* cob_file::record

Definition at line 175 of file fileio.h.

6.48.2.24 size_t cob_file::record_max

Definition at line 183 of file fileio.h.

6.48.2.25 size_t cob_file::record_min

Definition at line 182 of file fileio.h.

6.48.2.26 cob_field* cob_file::record_size

Definition at line 176 of file fileio.h.

6.48.2.27 const char* cob_file::select_name

Definition at line 172 of file fileio.h.

6.48.2.28 const unsigned char* cob_file::sort_collating

Definition at line 180 of file fileio.h.

6.48.2.29 char cob_file::special

Definition at line 191 of file fileio.h.

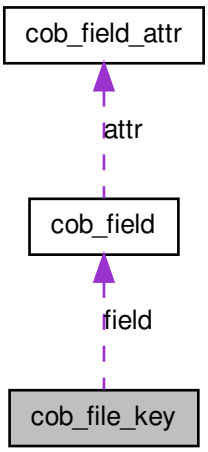
The documentation for this struct was generated from the following file:

- libcob/fileio.h

6.49 cob_file_key Struct Reference

`#include <fileio.h>`

Collaboration diagram for cob_file_key:



Public Attributes

- cob_field * field
- int flag
- size_t offset

6.49.1 Detailed Description

Definition at line 152 of file fileio.h.

6.49.2 Member Data Documentation

6.49.2.1 cob_field* cob_file_key::field

Definition at line 153 of file fileio.h.

6.49.2.2 int cob_file_key::flag

Definition at line 154 of file fileio.h.

6.49.2.3 size_t cob_file_key::offset

Definition at line 156 of file fileio.h.

The documentation for this struct was generated from the following file:

- libcob/fileio.h

6.50 cob_fileio_funcs Struct Reference

`#include <fileio.h>`

Public Attributes

- int(* open)(cob_file *, char *, const int, const int)
- int(* close)(cob_file *, const int)
- int(* start)(cob_file *, const int, cob_field *)
- int(* read)(cob_file *, cob_field *, int)
- int(* read_next)(cob_file *, int)
- int(* write)(cob_file *, const int)
- int(* rewrite)(cob_file *, const int)
- int(* fdelete)(cob_file *)

6.50.1 Detailed Description

Definition at line 216 of file fileio.h.

6.50.2 Member Data Documentation

6.50.2.1 int(* cob_fileio_funcs::close)(cob_file *, const int)

Definition at line 218 of file fileio.h.

6.50.2.2 int(* cob_fileio_funcs::fdelete)(cob_file *)

Definition at line 224 of file fileio.h.

6.50.2.3 int(* cob_fileio_funcs::open)(cob_file *, char *, const int, const int)

Definition at line 217 of file fileio.h.

6.50.2.4 int(∗ cob_fileio_funcs::read)(cob_file ∗, cob_field ∗, int)

Definition at line 220 of file fileio.h.

6.50.2.5 int(∗ cob_fileio_funcs::read_next)(cob_file ∗, int)

Definition at line 221 of file fileio.h.

6.50.2.6 int(∗ cob_fileio_funcs::rewrite)(cob_file ∗, const int)

Definition at line 223 of file fileio.h.

6.50.2.7 int(∗ cob_fileio_funcs::start)(cob_file ∗, const int, cob_field ∗)

Definition at line 219 of file fileio.h.

6.50.2.8 int(∗ cob_fileio_funcs::write)(cob_file ∗, const int)

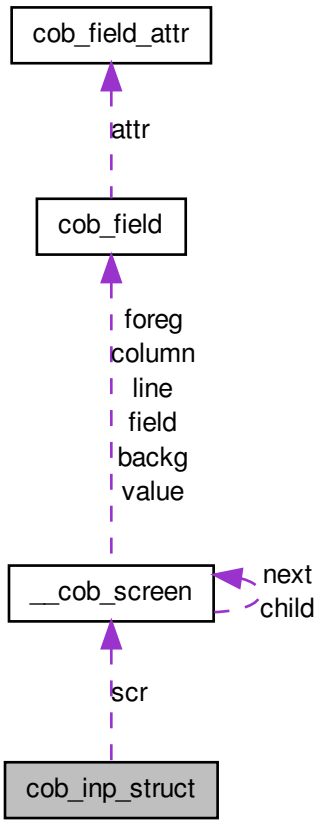
Definition at line 222 of file fileio.h.

The documentation for this struct was generated from the following file:

- libcob/fileio.h

6.51 cob_inp_struct Struct Reference

Collaboration diagram for cob_inp_struct:



Public Attributes

- cob_screen * scr
- size_t up_index
- size_t down_index
- int this_y
- int this_x

6.51.1 Detailed Description

Definition at line 56 of file screenio.c.

6.51.2 Member Data Documentation

6.51.2.1 size_t cob_inp_struct::down_index

Definition at line 59 of file screenio.c.

6.51.2.2 cob_screen* cob_inp_struct::scr

Definition at line 57 of file screenio.c.

6.51.2.3 int cob_inp_struct::this_x

Definition at line 61 of file screenio.c.

6.51.2.4 int cob_inp_struct::this_y

Definition at line 60 of file screenio.c.

6.51.2.5 size_t cob_inp_struct::up_index

Definition at line 58 of file screenio.c.

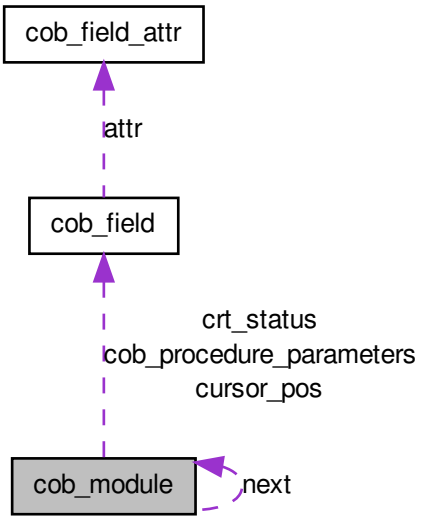
The documentation for this struct was generated from the following file:

- libcob/screenio.c

6.52 cob_module Struct Reference

```
#include <common.h>
```

Collaboration diagram for cob_module:



Public Attributes

- struct cob_module * next
- const unsigned char * collating_sequence
- cob_field * crt_status
- cob_field * cursor_pos
- cob_field ** cob_procedure_parameters
- const unsigned char display_sign
- const unsigned char decimal_point
- const unsigned char currency_symbol
- const unsigned char numeric_separator
- const unsigned char flag_filename_mapping
- const unsigned char flag_binary_truncate
- const unsigned char flag_pretty_display
- const unsigned char spare8

6.52.1 Detailed Description

Definition at line 201 of file common.h.

6.52.2 Member Data Documentation

6.52.2.1 cob_field** cob_module::cob_procedure_parameters

Definition at line 206 of file common.h.

6.52.2.2 const unsigned char* cob_module::collating_sequence

Definition at line 203 of file common.h.

6.52.2.3 cob_field* cob_module::crt_status

Definition at line 204 of file common.h.

6.52.2.4 const unsigned char cob_module::currency_symbol

Definition at line 209 of file common.h.

6.52.2.5 cob_field* cob_module::cursor_pos

Definition at line 205 of file common.h.

6.52.2.6 const unsigned char cob_module::decimal_point

Definition at line 208 of file common.h.

6.52.2.7 const unsigned char cob_module::display_sign

Definition at line 207 of file common.h.

6.52.2.8 const unsigned char cob_module::flag_binary_truncate

Definition at line 212 of file common.h.

6.52.2.9 const unsigned char cob_module::flag_filename_mapping

Definition at line 211 of file common.h.

6.52.2.10 const unsigned char cob_module::flag_pretty_display

Definition at line 213 of file common.h.

6.52.2.11 struct cob_module* cob_module::next

Definition at line 202 of file common.h.

6.52.2.12 const unsigned char cob_module::numeric_separator

Definition at line 210 of file common.h.

6.52.2.13 const unsigned char cob_module::spare8

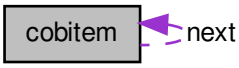
Definition at line 214 of file common.h.

The documentation for this struct was generated from the following file:

- libcob/common.h

6.53 cobitem Struct Reference

Collaboration diagram for cobitem:



Public Attributes

- struct cobitem * next
- size_t end_of_block
- unsigned char block_byte
- unsigned char unique [sizeof(size_t)]
- unsigned char item [1]

6.53.1 Detailed Description

Definition at line 158 of file fileio.c.

6.53.2 Member Data Documentation

6.53.2.1 unsigned char cobitem::block_byte

Definition at line 161 of file fileio.c.

6.53.2.2 size_t cobitem::end_of_block

Definition at line 160 of file fileio.c.

6.53.2.3 unsigned char cobitem::item[1]

Definition at line 163 of file fileio.c.

6.53.2.4 struct cobitem* cobitem::next

Definition at line 159 of file fileio.c.

6.53.2.5 unsigned char cobitem::unique[sizeof(size_t)]

Definition at line 162 of file fileio.c.

The documentation for this struct was generated from the following file:

- libcob/fileio.c

6.54 cobjmp_buf Struct Reference

```
#include <call.h>
```

Public Attributes

- int cbj_int [4]
- void * cbj_ptr [4]
- jmp_buf cbj_jmp_buf
- void * cbj_ptr_rest [2]

6.54.1 Detailed Description

Definition at line 27 of file call.h.

6.54.2 Member Data Documentation

6.54.2.1 int cobjmp_buf::cbj_int[4]

Definition at line 28 of file call.h.

6.54.2.2 jmp_buf cobjmp_buf::cbj_jmp_buf

Definition at line 30 of file call.h.

6.54.2.3 void* cobjmp_buf::cbj_ptr[4]

Definition at line 29 of file call.h.

6.54.2.4 void* cobjmp_buf::cbj_ptr_rest[2]

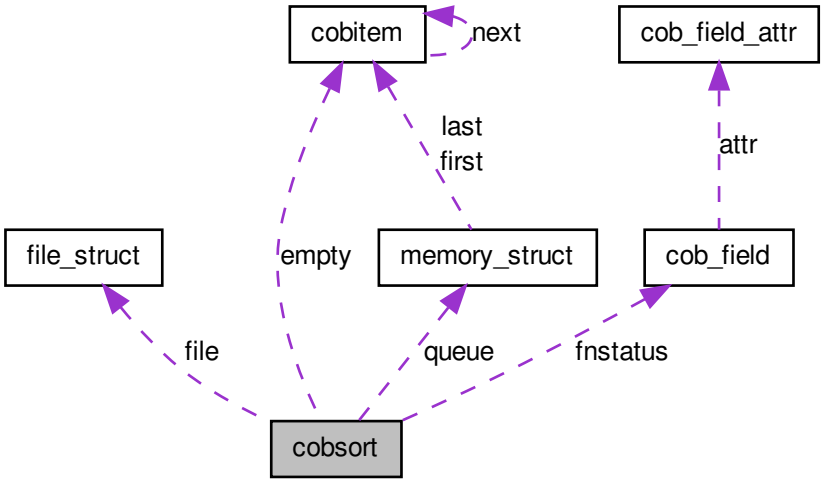
Definition at line 31 of file call.h.

The documentation for this struct was generated from the following file:

- libcob/call.h

6.55 cobsort Struct Reference

Collaboration diagram for cobsort:



Public Attributes

- void * pointer
- struct cobitem * empty
- void * sort_return
- cob_field * fnstatus
- size_t unique
- size_t retrieving
- size_t files_used
- size_t size
- size_t r_size
- size_t w_size
- size_t memory
- int destination_file
- int retrieval_queue
- struct memory_struct queue [4]
- struct file_struct file [4]

6.55.1 Detailed Description

Definition at line 177 of file fileio.c.

6.55.2 Member Data Documentation

6.55.2.1 int cobsort::destination_file

Definition at line 189 of file fileio.c.

6.55.2.2 struct cobitem* cobsort::empty

Definition at line 179 of file fileio.c.

6.55.2.3 struct file_struct cobsort::file[4]

Definition at line 192 of file fileio.c.

6.55.2.4 size_t cobsort::files_used

Definition at line 184 of file fileio.c.

6.55.2.5 cob_field* cobsort::fnstatus

Definition at line 181 of file fileio.c.

6.55.2.6 size_t cobsort::memory

Definition at line 188 of file fileio.c.

6.55.2.7 void* cobsort::pointer

Definition at line 178 of file fileio.c.

6.55.2.8 struct memory_struct cobsort::queue[4]

Definition at line 191 of file fileio.c.

6.55.2.9 size_t cobsort::r_size

Definition at line 186 of file fileio.c.

6.55.2.10 int cobsort::retrieval_queue

Definition at line 190 of file fileio.c.

6.55.2.11 size_t cobsort::retrieving

Definition at line 183 of file fileio.c.

6.55.2.12 size_t cobsort::size

Definition at line 185 of file fileio.c.

6.55.2.13 void* cobsort::sort_return

Definition at line 180 of file fileio.c.

6.55.2.14 size_t cobsort::unique

Definition at line 182 of file fileio.c.

6.55.2.15 size_t cobsort::w_size

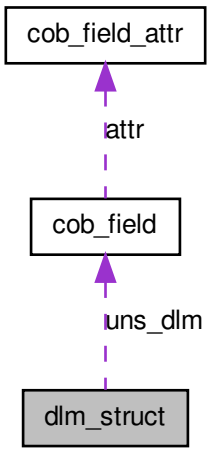
Definition at line 187 of file fileio.c.

The documentation for this struct was generated from the following file:

- libcob/fileio.c

6.56 dlm_struct Struct Reference

Collaboration diagram for dlm_struct:



Public Attributes

- cob_field * uns_dlm
- int uns_all

6.56.1 Detailed Description

Definition at line 41 of file strings.c.

6.56.2 Member Data Documentation

6.56.2.1 int dlm_struct::uns_all

Definition at line 43 of file strings.c.

6.56.2.2 cob_field* dlm_struct::uns_dlm

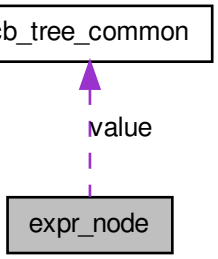
Definition at line 42 of file strings.c.

The documentation for this struct was generated from the following file:

- libcob/strings.c

6.57 expr_node Struct Reference

Collaboration diagram for expr_node:



Public Attributes

- int token
- cb_tree value

6.57.1 Detailed Description

Definition at line 48 of file typeck.c.

6.57.2 Member Data Documentation

6.57.2.1 int expr_node::token

Definition at line 56 of file typeck.c.

6.57.2.2 cb_tree expr_node::value

Definition at line 58 of file typeck.c.

The documentation for this struct was generated from the following file:

- cobc/typeck.c

6.58 file_struct Struct Reference

Public Attributes

- FILE * fp
- size_t count

6.58.1 Detailed Description

Definition at line 172 of file fileio.c.

6.58.2 Member Data Documentation

6.58.2.1 size_t file_struct::count

Definition at line 174 of file fileio.c.

6.58.2.2 FILE* file_struct::fp

Definition at line 173 of file fileio.c.

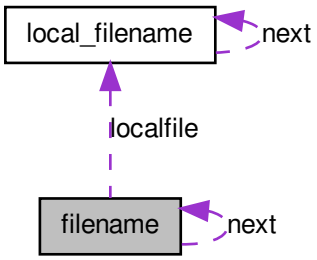
The documentation for this struct was generated from the following file:

- libcob/fileio.c

6.59 filename Struct Reference

```
#include <cobc.h>
```

Collaboration diagram for filename:



Public Attributes

- struct filename * next
- char * source
- char * preprocess
- char * translate
- char * trstorage
- char * object
- char * demangle_source
- struct local_filename * localfile
- int need_preprocess
- int need_translate
- int need_assemble

6.59.1 Detailed Description

Definition at line 83 of file cobc.h.

6.59.2 Member Data Documentation

6.59.2.1 char* filename::demangle_source

Definition at line 90 of file cobc.h.

6.59.2.2 struct local_filename* filename::localfile

Definition at line 91 of file cobc.h.

6.59.2.3 int filename::need_assemble

Definition at line 94 of file cobc.h.

6.59.2.4 int filename::need_preprocess

Definition at line 92 of file cobc.h.

6.59.2.5 int filename::need_translate

Definition at line 93 of file cobc.h.

6.59.2.6 struct filename* filename::next

Definition at line 84 of file cobc.h.

6.59.2.7 char* filename::object

Definition at line 89 of file cobc.h.

6.59.2.8 char* filename::preprocess

Definition at line 86 of file cobc.h.

6.59.2.9 char* filename::source

Definition at line 85 of file cobc.h.

6.59.2.10 char* filename::translate

Definition at line 87 of file cobc.h.

6.59.2.11 char* filename::trstorage

Definition at line 88 of file cobc.h.

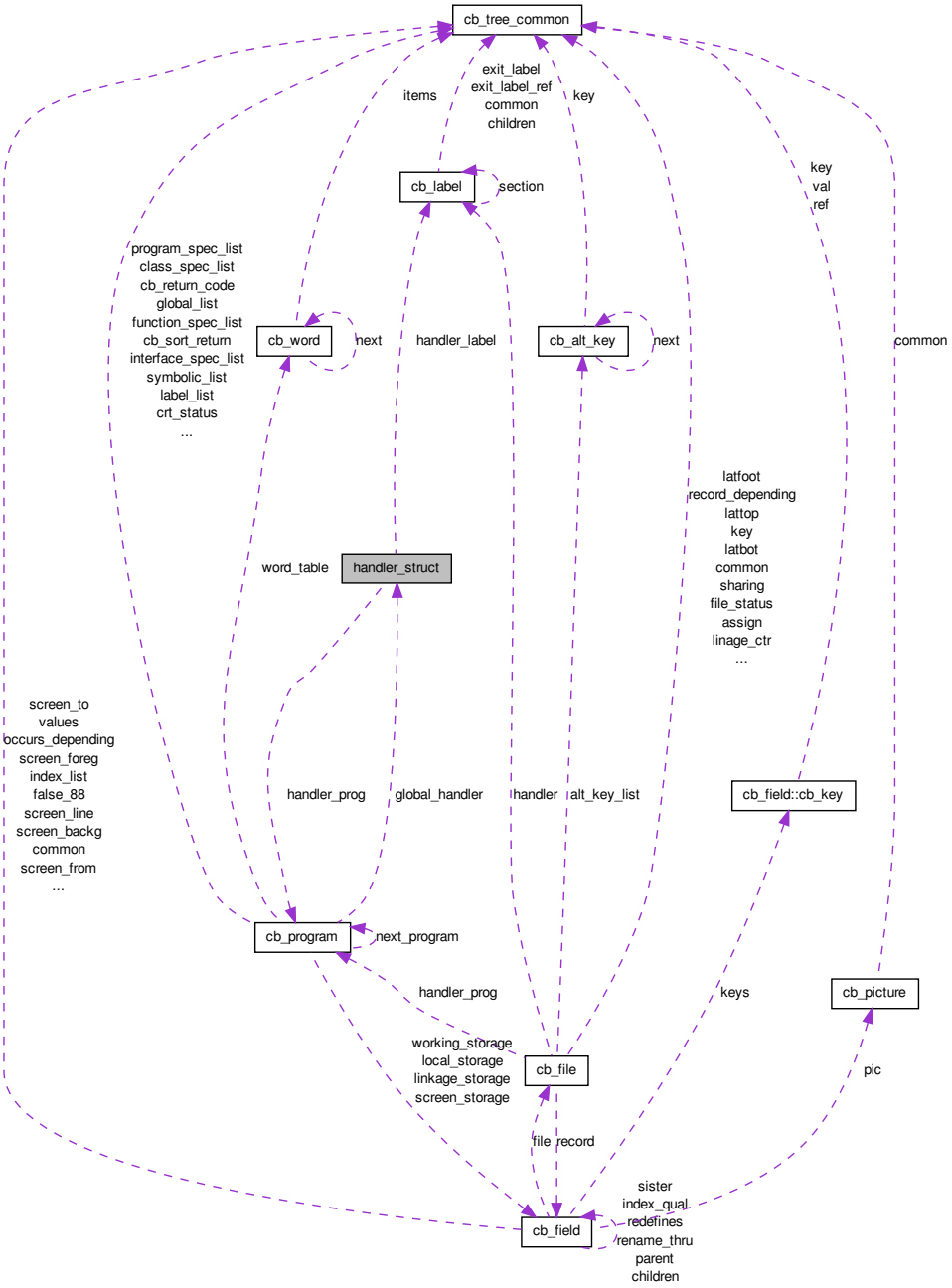
The documentation for this struct was generated from the following file:

- cobc/cobc.h

6.60 handler_struct Struct Reference

```
#include <tree.h>
```

Collaboration diagram for handler_struct:



Public Attributes

- struct cb_label * handler_label
- struct cb_program * handler_prog

6.60.1 Detailed Description

Definition at line 583 of file tree.h.

6.60.2 Member Data Documentation

6.60.2.1 struct cb_label* handler_struct::handler_label

Definition at line 584 of file tree.h.

6.60.2.2 struct cb_program* handler_struct::handler_prog

Definition at line 585 of file tree.h.

The documentation for this struct was generated from the following file:

- cobc/tree.h

6.61 indexed_file Struct Reference

Public Attributes

- size_t key_index
- unsigned char * last_key
- unsigned char * temp_key
- DB ** db
- DBT key
- DBT data
- unsigned char ** last_readkey
- unsigned int * last_dupno
- int * rewrite_sec_key
- DBC ** cursor
- DB_LOCK bdb_file_lock
- char * filename
- DB_LOCK bdb_record_lock
- int write_cursor_open
- unsigned int bdb_lock_id
- int record_locked
- int filenamelen

6.61.1 Detailed Description

Definition at line 307 of file fileio.c.

6.61.2 Member Data Documentation

6.61.2.1 DB_LOCK indexed_file::bdb_file_lock

Definition at line 319 of file fileio.c.

6.61.2.2 unsigned int indexed_file::bdb_lock_id

Definition at line 323 of file fileio.c.

6.61.2.3 DB_LOCK indexed_file::bdb_record_lock

Definition at line 321 of file fileio.c.

6.61.2.4 DBC** indexed_file::cursor

Definition at line 318 of file fileio.c.

6.61.2.5 DBT indexed_file::data

Definition at line 313 of file fileio.c.

6.61.2.6 DB** indexed_file::db

Definition at line 311 of file fileio.c.

6.61.2.7 char* indexed_file::filename

Definition at line 320 of file fileio.c.

6.61.2.8 int indexed_file::filenamelen

Definition at line 325 of file fileio.c.

6.61.2.9 DBT indexed_file::key

Definition at line 312 of file fileio.c.

6.61.2.10 size_t indexed_file::key_index

Definition at line 308 of file fileio.c.

6.61.2.11 unsigned int* indexed_file::last_dupno

Definition at line 315 of file fileio.c.

6.61.2.12 unsigned char* indexed_file::last_key

Definition at line 309 of file fileio.c.

6.61.2.13 unsigned char** indexed_file::last_readkey

Definition at line 314 of file fileio.c.

6.61.2.14 int indexed_file::record_locked

Definition at line 324 of file fileio.c.

6.61.2.15 int* indexed_file::rewrite_sec_key

Definition at line 316 of file fileio.c.

6.61.2.16 unsigned char* indexed_file::temp_key

Definition at line 310 of file fileio.c.

6.61.2.17 int indexed_file::write_cursor_open

Definition at line 322 of file fileio.c.

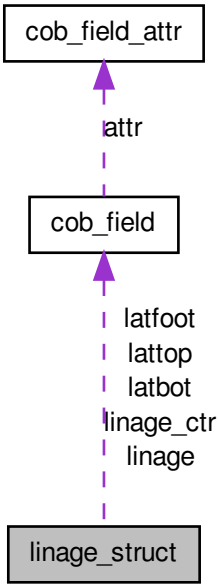
The documentation for this struct was generated from the following file:

- libcob/fileio.c

6.62 linage_struct Struct Reference

`#include <fileio.h>`

Collaboration diagram for linage_struct:



Public Attributes

- cob_field * linage
- cob_field * linage_ctr
- cob_field * latfoot
- cob_field * lattop
- cob_field * latbot
- int lin_lines
- int lin_foot
- int lin_top
- int lin_bot

6.62.1 Detailed Description

Definition at line 159 of file fileio.h.

6.62.2 Member Data Documentation

6.62.2.1 cob_field* linage_struct::latbot

Definition at line 164 of file fileio.h.

6.62.2.2 cob_field* linage_struct::latfoot

Definition at line 162 of file fileio.h.

6.62.2.3 cob_field* linage_struct::lattop

Definition at line 163 of file fileio.h.

6.62.2.4 int linage_struct::lin_bot

Definition at line 168 of file fileio.h.

6.62.2.5 int linage_struct::lin_foot

Definition at line 166 of file fileio.h.

6.62.2.6 int linage_struct::lin_lines

Definition at line 165 of file fileio.h.

6.62.2.7 int linage_struct::lin_top

Definition at line 167 of file fileio.h.

6.62.2.8 cob_field* linage_struct::linage

Definition at line 160 of file fileio.h.

6.62.2.9 cob_field* linage_struct::linage_ctr

Definition at line 161 of file fileio.h.

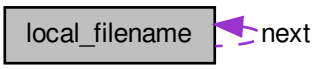
The documentation for this struct was generated from the following file:

- libcob/fileio.h

6.63 local_filename Struct Reference

`#include <cobc.h>`

Collaboration diagram for local_filename:



Public Attributes

- struct local_filename * next
- char * local_name
- FILE * local_fp

6.63.1 Detailed Description

Definition at line 77 of file cobc.h.

6.63.2 Member Data Documentation

6.63.2.1 FILE* local_filename::local_fp

Definition at line 80 of file cobc.h.

6.63.2.2 char* local_filename::local_name

Definition at line 79 of file cobc.h.

6.63.2.3 struct local_filename* local_filename::next

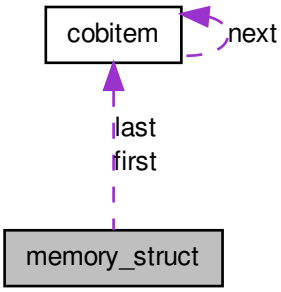
Definition at line 78 of file cobc.h.

The documentation for this struct was generated from the following file:

- cobc/cobc.h

6.64 memory_struct Struct Reference

Collaboration diagram for memory_struct:



Public Attributes

- struct cobitem * first
- struct cobitem * last
- size_t count

6.64.1 Detailed Description

Definition at line 166 of file fileio.c.

6.64.2 Member Data Documentation

6.64.2.1 size_t memory_struct::count

Definition at line 169 of file fileio.c.

6.64.2.2 struct cobitem* memory_struct::first

Definition at line 167 of file fileio.c.

6.64.2.3 struct cobitem* memory_struct::last

Definition at line 168 of file fileio.c.

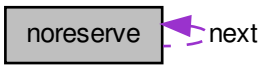
The documentation for this struct was generated from the following file:

- libcob/fileio.c

6.65 noreserve Struct Reference

`#include <cobc.h>`

Collaboration diagram for noreserve:



Public Attributes

- struct noreserve * next
- char * noresword

6.65.1 Detailed Description

Definition at line 151 of file cobc.h.

6.65.2 Member Data Documentation

6.65.2.1 struct noreserve* noreserve::next

Definition at line 152 of file cobc.h.

6.65.2.2 char* noreserve::noresword

Definition at line 153 of file cobc.h.

The documentation for this struct was generated from the following file:

- cobc/cobc.h

6.66 option Struct Reference

`#include <getopt.h>`

Public Attributes

- char * name
- int has_arg
- int * flag
- int val

6.66.1 Detailed Description

Definition at line 94 of file getopt.h.

6.66.2 Member Data Documentation

6.66.2.1 int* option::flag

Definition at line 104 of file getopt.h.

6.66.2.2 int option::has_arg

Definition at line 103 of file getopt.h.

6.66.2.3 char* option::name

Definition at line 99 of file getopt.h.

6.66.2.4 int option::val

Definition at line 105 of file getopt.h.

The documentation for this struct was generated from the following file:

- lib/getopt.h

6.67 reserved Struct Reference

Public Attributes

- const char * name
- const int token

6.67.1 Detailed Description

Definition at line 69 of file reserved.c.

6.67.2 Member Data Documentation

6.67.2.1 const char* reserved::name

Definition at line 70 of file reserved.c.

6.67.2.2 const int reserved::token

Definition at line 71 of file reserved.c.

The documentation for this struct was generated from the following file:

- cobc/reserved.c

6.68 sort_list Struct Reference

Collaboration diagram for sort_list:

Public Attributes

- struct sort_list * next

6.68.1 Detailed Description

Definition at line 121 of file codegen.c.

6.68.2 Member Data Documentation

6.68.2.1 struct sort_list* sort_list::next

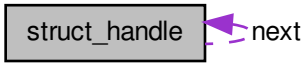
Definition at line 122 of file codegen.c.

The documentation for this struct was generated from the following file:

- cobc/codegen.c

6.69 struct_handle Struct Reference

Collaboration diagram for struct_handle:



Public Attributes

- struct struct_handle * next
- lt_dlhandle preload_handle

6.69.1 Detailed Description

Definition at line 119 of file call.c.

6.69.2 Member Data Documentation

6.69.2.1 struct struct_handle* struct_handle::next

Definition at line 120 of file call.c.

6.69.2.2 lt_dlhandle struct_handle::preload_handle

Definition at line 121 of file call.c.

The documentation for this struct was generated from the following file:

- libcob/call.c

6.70 system_table Struct Reference

Public Attributes

- const char * syst_name
- const char * syst_call
- const int syst_params
- void * syst_call

6.70.1 Detailed Description

Definition at line 125 of file codegen.c.

6.70.2 Member Data Documentation

6.70.2.1 const char* system_table::syst_call

Definition at line 127 of file codegen.c.

6.70.2.2 void* system_table::syst_call

Definition at line 148 of file call.c.

6.70.2.3 const char * system_table::syst_name

Definition at line 126 of file codegen.c.

6.70.2.4 const int system_table::syst_params

Definition at line 45 of file typeck.c.

The documentation for this struct was generated from the following files:

- cobc/codegen.c
- cobc/typeck.c
- libcob/call.c

6.71 yy_buffer_state Struct Reference

Public Attributes

- FILE * yy_input_file
- char * yy_ch_buf

- char * yy_buf_pos
- yy_size_t yy_buf_size
- int yy_n_chars
- int yy_is_our_buffer
- int yy_is_interactive
- int yy_at_bol
- int yy_fill_buffer
- int yy_buffer_status

6.71.1 Detailed Description

Definition at line 162 of file pplex.c.

6.71.2 Member Data Documentation

6.71.2.1 int yy_buffer_state::yy_at_bol

Definition at line 196 of file pplex.c.

6.71.2.2 char * yy_buffer_state::yy_buf_pos

Definition at line 167 of file pplex.c.

6.71.2.3 yy_size_t yy_buffer_state::yy_buf_size

Definition at line 172 of file pplex.c.

6.71.2.4 int yy_buffer_state::yy_buffer_status

Definition at line 203 of file pplex.c.

6.71.2.5 char * yy_buffer_state::yy_ch_buf

Definition at line 166 of file pplex.c.

6.71.2.6 int yy_buffer_state::yy_fill_buffer

Definition at line 201 of file pplex.c.

6.71.2.7 FILE * yy_buffer_state::yy_input_file

Definition at line 164 of file pplex.c.

6.71.2.8 int yy_buffer_state::yy_is_interactive

Definition at line 190 of file pplex.c.

6.71.2.9 int yy_buffer_state::yy_is_our_buffer

Definition at line 183 of file pplex.c.

6.71.2.10 int yy_buffer_state::yy_n_chars

Definition at line 177 of file pplex.c.

The documentation for this struct was generated from the following files:

- cobc/pplex.c
- cobc/scanner.c

6.72 yyalloc Union Reference

Public Attributes

- short yyss
- YYSTYPE yyvs

6.72.1 Detailed Description

Definition at line 1283 of file parser.c.

6.72.2 Member Data Documentation

6.72.2.1 short yyalloc::yyss

Definition at line 1285 of file parser.c.

6.72.2.2 YYSTYPE yyalloc::yyvs

Definition at line 1286 of file parser.c.

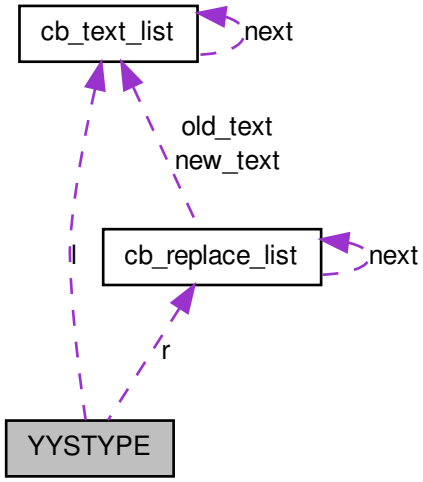
The documentation for this union was generated from the following files:

- cobc/parser.c
- cobc/ppparse.c

6.73 YYSTYPE Union Reference

`#include <ppparse.h>`

Collaboration diagram for YYSTYPE:



Public Attributes

- char * s
- struct cb_text_list * l
- struct cb_replace_list * r

6.73.1 Detailed Description

Definition at line 134 of file ppparse.c.

6.73.2 Member Data Documentation

6.73.2.1 struct cb_text_list * YYSTYPE::l

Definition at line 136 of file ppparse.c.

6.73.2.2 struct cb_replace_list * YYSTYPE::r

Definition at line 137 of file ppparse.c.

6.73.2.3 char * YYSTYPE::s

Definition at line 135 of file ppparse.c.

The documentation for this union was generated from the following files:

- cobc/ppparse.c
- cobc/ppparse.h

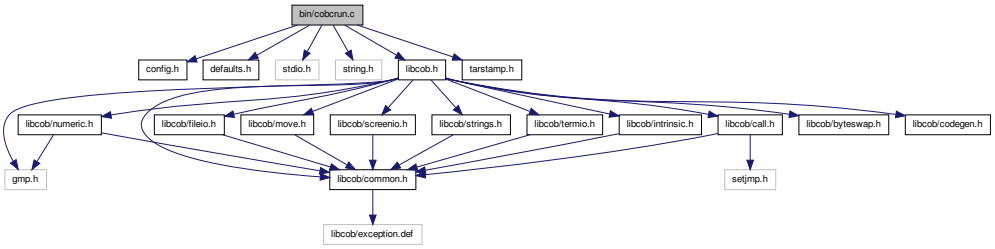
Chapter 7

File Documentation

7.1 bin/cobcrun.c File Reference

```
#include "config.h"
#include "defaults.h"
#include <stdio.h>
#include <string.h>
#include <libcob.h>
#include <tarstamp.h>
```

Include dependency graph for cobcrun.c:



Functions

- int main (int argc, char **argv)

7.1.1 Function Documentation

7.1.1.1 int main (int *argc*, char ** *argv*)

Definition at line 63 of file cobcrun.c.

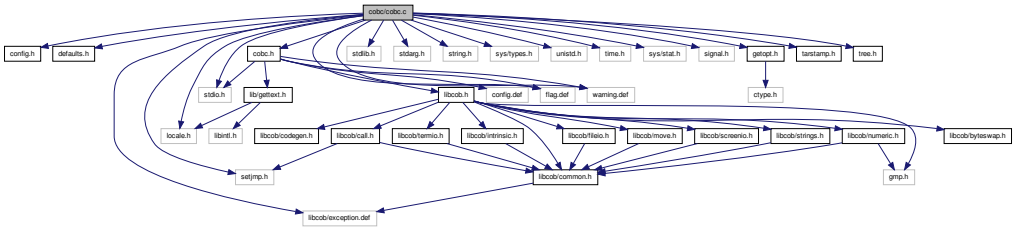
```
{
        union {
                int     (*func)();
                void    *func_void;
        } unifunc;

        if (argc <= 1) {
                print_usage ();
                return 1;
        }
        /* Quick check without getopt */
        if (!strncmp (argv[1], "--version", 10) ||
            !strncmp (argv[1], "-V", 4)) {
                print_version ();
                return 0;
        }
        if (!strncmp (argv[1], "--help", 10) ||
            !strncmp (argv[1], "-h", 4)) {
                print_usage ();
                return 0;
        }
        if (strlen (argv[1]) > 31) {
                fprintf (stderr, "Invalid PROGRAM name\n");
                return 1;
        }
        cob_init (argc - 1, &argv[1]);
        unifunc.func_void = cob_resolve (argv[1]);
        if (unifunc.func_void == NULL) {
                cob_call_error ();
        }
        cob_stop_run ( unifunc.func() );
}
```

7.2 cobc/cobc.c File Reference

```
#include "config.h"
#include "defaults.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <setjmp.h>
#include <sys/types.h>
#include <unistd.h>
#include <time.h>
#include <sys/stat.h>
```

```
#include <signal.h>
#include <getopt.h>
#include <locale.h>
#include <tarstamp.h>
#include "cobc.h"
#include "tree.h"
#include <libcob/exception.def>
#include "flag.def"
#include "warning.def"
```

Include dependency graph for cobc.c:



Defines

- #define COB_EXCEPTION(code, tag, name, critical) {name, 0x##code, 0},
- #define CB_FLAG(var, name, doc) int var = 0;
- #define CB_WARNDEF(var, name, wall, doc) int var = 0;
- #define PATHSEPS ":"
- #define COB_NUM_CSYNS sizeof(cob_csyns) / sizeof(char *)
- #define CB_FLAG(var, name, doc)
- #define CB_WARNDEF(var, name, wall, doc)
- #define CB_WARNDEF(var, name, wall, doc)
- #define CB_FLAG(var, name, doc)
- #define CB_WARNDEF(var, name, wall, doc) var = 0;
- #define CB_WARNDEF(var, name, wall, doc) if (wall) var = 1;
- #define CB_WARNDEF(var, name, wall, doc) var = 1;

Typedefs

- typedef void(* cob_sighandler_t)(int)

Enumerations

- enum cb_compile_level {

  CB_LEVEL_PREPROCESS = 1, CB_LEVEL_TRANSLATE, CB_LEVEL_COMPILE, CB_LEVEL_ASSEMBLE,

  CB_LEVEL_MODULE, CB_LEVEL_LIBRARY, CB_LEVEL_EXECUTABLE }

Functions

- void cobc_abort (const char *filename, const int linenum)
- void cobc_tree_cast_error (cb_tree x, const char *filen, const int linenum, const int tagnum)
- void * cobc_malloc (const size_t size)
- void * cobc_realloc (void *prevptr, const size_t size)
- struct cb_text_list * cb_text_list_add (struct cb_text_list *list, const char *text)
- size_t cobc_check_valid_name (char *name)
- int main (int argc, char *argv[])

Variables

- int cb_source_format = CB_FORMAT_FIXED
- int cb_display_sign = COB_DISPLAY_SIGN_ASCII
- struct cb_exception cb_exception_table []
- int cb_id = 1
- int cb_attr_id = 1
- int cb_literal_id = 1
- int cb_field_id = 1
- int cb_storage_id = 1
- int cb_flag_main = 0
- int errorcount = 0
- int warningcount = 0
- int alt_ebcdic = 0
- int optimize_flag = 0
- char * cb_source_file = NULL
- char * cb_oc_build_stamp = NULL
- char * source_name
- char * demangle_name
- int cb_source_line = 0
- FILE * cb_storage_file
- char * cb_storage_file_name
- FILE * cb_listing_file = NULL
- FILE * cb_depend_file = NULL
- char * cb_depend_target = NULL
- struct cb_text_list * cb_depend_list = NULL
- struct cb_text_list * cb_include_list = NULL

- struct cb_text_list * cb_extension_list = NULL
- int cb_saveargc
- char ** cb_saveargv
- const char * cob_config_dir

7.2.1 Define Documentation

7.2.1.1 #define CB_FLAG(var, name, doc) int var = 0;

Definition at line 97 of file cobc.c.

7.2.1.2 #define CB_FLAG(var, name, doc)

Value:

```
if (strcmp (name, "static-call"))       \
                printf ("  -f%-19s %s\n", name, gettext (doc));
```

Definition at line 97 of file cobc.c.

7.2.1.3 #define CB_FLAG(var, name, doc)

Value:

```
{"f"name, no_argument, &var, 1},        \
        {"fno-"name, no_argument, &var, 0},
```

Definition at line 97 of file cobc.c.

7.2.1.4 #define CB_WARNDEF(var, name, wall, doc) var = 1;

Definition at line 102 of file cobc.c.

7.2.1.5 #define CB_WARNDEF(var, name, wall, doc)

Value:

```
printf ("  -W%-19s %s", name, gettext (doc)); \
        if (!wall) { \
                puts (_(" (NOT set with -Wall)")); \
        } else { \
                printf ("\n"); \
        }
```

Definition at line 102 of file cobc.c.

7.2.1.6 #define CB_WARNDEF(var, name, wall, doc) var = 0;

Definition at line 102 of file cobc.c.

7.2.1.7 #define CB_WARNDEF(var, name, wall, doc)

Value:

```
{"W"name, no_argument, &var, 1},        \
        {"Wno-"name, no_argument, &var, 0},
```

Definition at line 102 of file cobc.c.

7.2.1.8 #define CB_WARNDEF(var, name, wall, doc) int var = 0;

Definition at line 102 of file cobc.c.

7.2.1.9 #define CB_WARNDEF(var, name, wall, doc) if (wall) var = 1;

Definition at line 102 of file cobc.c.

7.2.1.10 #define COB_EXCEPTION(code, tag, name, critical) {name, 0x##code, 0},

Definition at line 88 of file cobc.c.

7.2.1.11 #define COB_NUM_CSYNS sizeof(cob_csyns) / sizeof(char *)

Definition at line 201 of file cobc.c.

7.2.1.12 #define PATHSEPS ":"

Definition at line 109 of file cobc.c.

7.2.2 Typedef Documentation

7.2.2.1 typedef void(* cob_sighandler_t)(int)

Definition at line 249 of file cobc.c.

7.2.3 Enumeration Type Documentation

7.2.3.1 enum cb_compile_level

Enumerator:

CB_LEVEL_PREPROCESS

CB_LEVEL_TRANSLATE

CB_LEVEL_COMPILE

CB_LEVEL_ASSEMBLE

CB_LEVEL_MODULE

CB_LEVEL_LIBRARY

CB_LEVEL_EXECUTABLE

Definition at line 65 of file cobc.c.

```
                {
        CB_LEVEL_PREPROCESS = 1,
        CB_LEVEL_TRANSLATE,
        CB_LEVEL_COMPILE,
        CB_LEVEL_ASSEMBLE,
        CB_LEVEL_MODULE,
        CB_LEVEL_LIBRARY,
        CB_LEVEL_EXECUTABLE
};
```

7.2.4 Function Documentation

7.2.4.1 struct cb_text_list* cb_text_list_add (struct cb_text_list * *list*, const char * *text*) [read]

Definition at line 355 of file cobc.c.

```
{
        struct cb_text_list *p;
        struct cb_text_list *l;

        p = cobc_malloc (sizeof (struct cb_text_list));
        p->text = strdup (text);
        p->next = NULL;
        if (!list) {
                return p;
        } else {
                for (l = list; l->next; l = l->next) { ; }
                l->next = p;
                return list;
        }
}
```

7.2.4.2 void cobc_abort (const char * *filename*, const int *linenum*)

Definition at line 310 of file cobc.c.

```
{
        fprintf (stderr, "%s:%d: Internal compiler error\n", filename, linenum);
        (void)longjmp (cob_jmpbuf, 1);
}
```

7.2.4.3 size_t cobc_check_valid_name (char * *name*)

Definition at line 373 of file cobc.c.

```
{
        size_t  n;

        for (n = 0; n < COB_NUM_CSYNS; ++n) {
                if (!strcmp (name, cob_csyns[n])) {
                        return 1;
                }
        }
        return 0;
}
```

7.2.4.4 void* cobc_malloc (const size_t *size*)

Definition at line 327 of file cobc.c.

```
{
        void *mptr;

        mptr = calloc (1, size);
        if (!mptr) {
                fprintf (stderr, "Cannot allocate %d bytes of memory - Aborting\n
      ", (int)size);
                fflush (stderr);
                (void)longjmp (cob_jmpbuf, 1);
        }
        return mptr;
}
```

7.2.4.5 void* cobc_realloc (void * *prevptr,* const size_t *size*)

Definition at line 341 of file cobc.c.

```
{
        void *mptr;

        mptr = realloc (prevptr, size);
        if (!mptr) {
                fprintf (stderr, "Cannot reallocate %d bytes of memory - Aborting
      \n", (int)size);
                fflush (stderr);
                (void)longjmp (cob_jmpbuf, 1);
        }
        return mptr;
}
```

7.2.4.6 void cobc_tree_cast_error (cb_tree *x*, const char * *filen*, const int *linenum*, const int *tagnum*)

Definition at line 317 of file cobc.c.

```
{
        fprintf (stderr, "%s:%d: Invalid type cast from '%s'\n",
                filen, linenum, x ? cb_name (x) : "null");
        fprintf (stderr, "Tag 1 %d Tag 2 %d\n", x ? CB_TREE_TAG(x) : 0,
                tagnum);
        (void)longjmp (cob_jmpbuf, 1);
}
```

7.2.4.7 int main (int *argc*, char * *argv[]*)

Definition at line 1686 of file cobc.c.

```
{
        struct filename         *fn;
        char                    *p;
        int                     status = 1;
        int                     year;
        int                     day;
        char                    month[32];
        char                    buff[COB_SMALL_BUFF];

#ifdef  HAVE_SIGNAL_H
        if ((intsig = signal (SIGINT, cobc_sig_handler)) == SIG_IGN) {
                (void)signal (SIGINT, SIG_IGN);
        }
#ifdef  SIGHUP
        if ((hupsig = signal (SIGHUP, cobc_sig_handler)) == SIG_IGN) {
                (void)signal (SIGHUP, SIG_IGN);
        }
#endif
#ifdef  SIGQUIT
        if ((qutsig = signal (SIGQUIT, cobc_sig_handler)) == SIG_IGN) {
                (void)signal (SIGQUIT, SIG_IGN);
        }
#endif
#endif

        cb_saveargc = argc;
        cb_saveargv = argv;

#ifdef  HAVE_SETLOCALE
        setlocale (LC_ALL, "");
#endif

#ifdef  ENABLE_NLS
        bindtextdomain (PACKAGE, LOCALEDIR);
        textdomain (PACKAGE);
#endif

#ifndef _WIN32
        cob_process_id = getpid ();
#endif
```

```
/* Initialize global variables */
memset (buff, 0, sizeof(buff));
memset (month, 0, sizeof(month));
day = 0;
year = 0;
sscanf (__DATE__, "%s %d %d", month, &day, &year);
if (day && year) {
        sprintf (buff, "%s %2.2d %4.4d %s", month, day, year, __TIME__);
} else {
        sprintf (buff, "%s %s", __DATE__, __TIME__);
}
cb_oc_build_stamp = cobc_malloc (strlen (buff) + 1);
strcpy (cb_oc_build_stamp, buff);

output_name = NULL;

if ((p = getenv ("TMPDIR")) != NULL) {
        cob_tmpdir = p;
} else if ((p = getenv ("TMP")) != NULL) {
        cob_tmpdir = p;
        sprintf (buff, "TMPDIR=%s", p);
        p = strdup (buff);
        putenv (p);
} else {
        cob_tmpdir = "/tmp";
        putenv ((char *)"TMPDIR=/tmp");
}
cob_cc = getenv ("COB_CC");
if (cob_cc == NULL) {
        cob_cc = COB_CC;
}
cobc_init_var (cob_cflags, "COB_CFLAGS", COB_CFLAGS);
cobc_init_var (cob_libs, "COB_LIBS", COB_LIBS);
cob_ldflags = getenv ("COB_LDFLAGS");
if (cob_ldflags == NULL) {
        cob_ldflags = COB_LDFLAGS;
}
cob_config_dir = getenv ("COB_CONFIG_DIR");
if (cob_config_dir == NULL) {
        cob_config_dir = COB_CONFIG_DIR;
}
cob_copy_dir = getenv ("COB_COPY_DIR");
if (cob_copy_dir == NULL) {
        cob_copy_dir = COB_COPY_DIR;
}
memset (cob_define_flags, 0, sizeof (cob_define_flags));

p = getenv ("COB_LDADD");
if (p) {
        strcat (cob_libs, " ");
        strcat (cob_libs, p);
}
p = getenv ("COB_EBCDIC");
if (p && (*p == 'F' || *p == 'f')) {
        alt_ebcdic = 1;
}

/* Process command line arguments */
iargs = process_command_line (argc, argv);

/* Check the filename */
if (iargs == argc) {
```

```
            fprintf (stderr, "cobc: No input files\n");
            exit (1);
    }

    /* processes COBCPY environment variable */
    process_env_copy_path ();

    cb_include_list = cb_text_list_add (cb_include_list, cob_copy_dir);

    file_list = NULL;

    if (setjmp (cob_jmpbuf) != 0) {
            fprintf (stderr, "Aborting compile of %s at line %d\n",
                     cb_source_file, cb_source_line);
            fflush (stderr);
            if (yyout) {
                    fflush (yyout);
            }
            if (cb_storage_file) {
                    fflush (cb_storage_file);
            }
            status = 1;
            cobc_clean_up (status);
            return status;
    }

    /* Defaults are set here */
    if (!cb_flag_syntax_only) {
            if (!wants_nonfinal) {
                    if (cb_flag_main) {
                            cb_compile_level = CB_LEVEL_EXECUTABLE;
                    }
                    if (cb_flag_module) {
                            cb_compile_level = CB_LEVEL_MODULE;
                    }
                    if (cb_flag_library) {
                            cb_compile_level = CB_LEVEL_LIBRARY;
                    }
            }
            if (cb_compile_level == 0 && !wants_nonfinal) {
                    cb_compile_level = CB_LEVEL_MODULE;
                    cb_flag_module = 1;
            }
            if (wants_nonfinal && cb_compile_level != CB_LEVEL_PREPROCESS &&
                !cb_flag_main && !cb_flag_module && !cb_flag_library) {
                    cb_flag_module = 1;
            }
    } else {
                    cb_compile_level = CB_LEVEL_TRANSLATE;
    }

    if (output_name && cb_compile_level < CB_LEVEL_LIBRARY &&
        (argc - iargs) > 1) {
            fprintf (stderr, "cobc: -o option invalid in this combination\n")
;
            exit (1);
    }
    if (cb_flag_sign_ascii && cb_flag_sign_ebcdic) {
            fprintf (stderr, "Only one of -fsign-ascii or -fsign-ebcdic may b
e specified\n");
            exit (1);
    }
```

```
  if (cb_flag_sign_ascii) {
          cb_display_sign = COB_DISPLAY_SIGN_ASCII;
  }
  if (cb_flag_sign_ebcdic) {
          cb_display_sign = COB_DISPLAY_SIGN_EBCDIC;
  }
  if (cb_flag_notrunc) {
          cb_binary_truncate = 0;
          cb_pretty_display = 0;
  }

  while (iargs < argc) {
          fn = process_filename (argv[iargs++]);
          if (!fn) {
                  status = 1;
                  cobc_clean_up (status);
                  return status;
          }
          /* Preprocess */
          if (cb_compile_level >= CB_LEVEL_PREPROCESS && fn->
need_preprocess) {
                  if (preprocess (fn) != 0) {
                          cobc_clean_up (status);
                          return status;
                  }
          }
  }
  for (fn = file_list; fn; fn = fn->next) {
          cb_id = 1;
          cb_attr_id = 1;
          cb_literal_id = 1;
          cb_field_id = 1;
          cb_storage_id = 1;
          iparams++;
          demangle_name = fn->demangle_source;
          if (iparams > 1 && cb_compile_level == CB_LEVEL_EXECUTABLE &&
              !cb_flag_syntax_only) {
                  local_level = cb_compile_level;
                  cb_flag_main = 0;
                  cb_compile_level = CB_LEVEL_ASSEMBLE;
          }
          /* Translate */
          if (cb_compile_level >= CB_LEVEL_TRANSLATE && fn->need_translate)
 {
                  if (process_translate (fn) != 0) {
                          cobc_clean_up (status);
                          return status;
                  }
          }
          if (cb_flag_syntax_only) {
                  continue;
          }

          /* Compile */
          if (cb_compile_level == CB_LEVEL_COMPILE) {
                  if (process_compile (fn) != 0) {
                          cobc_clean_up (status);
                          return status;
                  }
          }

          /* Build module */
```

```
                if (cb_compile_level == CB_LEVEL_MODULE && fn->need_assemble) {
                        if (process_module_direct (fn) != 0) {
                                cobc_clean_up (status);
                                return status;
                        }
                } else {
                        /* Assemble */
                        if (cb_compile_level >= CB_LEVEL_ASSEMBLE && fn->
      need_assemble) {
                                if (process_assemble (fn) != 0) {
                                        cobc_clean_up (status);
                                        return status;
                                }
                        }

                        /* Build module */
                        if (cb_compile_level == CB_LEVEL_MODULE) {
                                if (process_module (fn) != 0) {
                                        cobc_clean_up (status);
                                        return status;
                                }
                        }
                }
        }

        if (!cb_flag_syntax_only) {
                /* Link */
                if (local_level == CB_LEVEL_EXECUTABLE) {
                        cb_compile_level = CB_LEVEL_EXECUTABLE;
                        cb_flag_main = 1;
                }
                if (cb_compile_level == CB_LEVEL_LIBRARY) {
                        if (process_library (file_list) != 0) {
                                cobc_clean_up (status);
                                return status;
                        }
                } else if (cb_compile_level == CB_LEVEL_EXECUTABLE) {
                        if (process_link (file_list) != 0) {
                                cobc_clean_up (status);
                                return status;
                        }
                }
        }

        /* We have successfully completed */
        status = 0;
        cobc_clean_up (status);

        return status;
}
```

7.2.5 Variable Documentation

7.2.5.1 int alt_ebcdic = 0

Definition at line 121 of file cobc.c.

7.2.5.2 int cb_attr_id = 1

Definition at line 113 of file cobc.c.

7.2.5.3 FILE* cb_depend_file = NULL

Definition at line 134 of file cobc.c.

7.2.5.4 struct cb_text_list* cb_depend_list = NULL

Definition at line 136 of file cobc.c.

7.2.5.5 char* cb_depend_target = NULL

Definition at line 135 of file cobc.c.

7.2.5.6 int cb_display_sign = COB_DISPLAY_SIGN_ASCII

Definition at line 84 of file cobc.c.

7.2.5.7 struct cb_exception cb_exception_table[ ]

Definition at line 89 of file cobc.c.

7.2.5.8 struct cb_text_list* cb_extension_list = NULL

Definition at line 138 of file cobc.c.

7.2.5.9 int cb_field_id = 1

Definition at line 115 of file cobc.c.

7.2.5.10 int cb_flag_main = 0

Definition at line 117 of file cobc.c.

7.2.5.11 int cb_id = 1

Definition at line 112 of file cobc.c.

7.2.5.12 struct cb_text_list* cb_include_list = NULL

Definition at line 137 of file cobc.c.

7.2.5.13 FILE* cb_listing_file = NULL

Definition at line 133 of file cobc.c.

7.2.5.14 int cb_literal_id = 1

Definition at line 114 of file cobc.c.

7.2.5.15 char* cb_oc_build_stamp = NULL

Definition at line 125 of file cobc.c.

7.2.5.16 int cb_saveargc

Definition at line 140 of file cobc.c.

7.2.5.17 char** cb_saveargv

Definition at line 141 of file cobc.c.

7.2.5.18 char* cb_source_file = NULL

Definition at line 124 of file cobc.c.

7.2.5.19 int cb_source_format = CB_FORMAT_FIXED

Definition at line 79 of file cobc.c.

7.2.5.20 int cb_source_line = 0

Definition at line 128 of file cobc.c.

7.2.5.21 FILE* cb_storage_file

Definition at line 130 of file cobc.c.

7.2.5.22 char* cb_storage_file_name

Definition at line 131 of file cobc.c.

7.2.5.23 int cb_storage_id = 1

Definition at line 116 of file cobc.c.

7.2.5.24 const char* cob_config_dir

Definition at line 143 of file cobc.c.

7.2.5.25 char* demangle_name

Definition at line 127 of file cobc.c.

7.2.5.26 int errorcount = 0

Definition at line 119 of file cobc.c.

7.2.5.27 int optimize_flag = 0

Definition at line 122 of file cobc.c.

7.2.5.28 char* source_name

Definition at line 126 of file cobc.c.

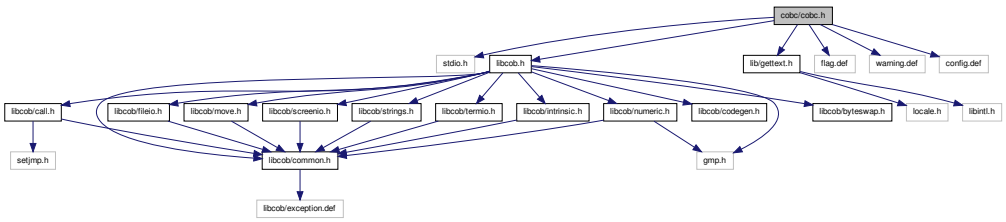
7.2.5.29 int warningcount = 0

Definition at line 120 of file cobc.c.

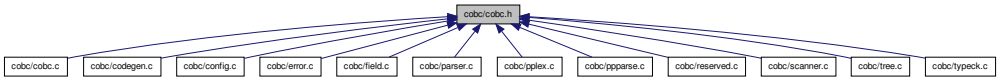
7.3 cobc/cobc.h File Reference

```
#include <stdio.h>
#include <libcob.h>
#include "lib/gettext.h"
#include "flag.def"
#include "warning.def"
#include "config.def"
```

Include dependency graph for cobc.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct cb_exception
- struct cb_text_list
- struct cb_replace_list
- struct local_filename
- struct filename
- struct noreserve

Defines

- #define COB_NON_ALIGNED
- #define ABORT() cobc_abort(__FILE__,__LINE__)
- #define CB_FORMAT_FIXED 0
- #define CB_FORMAT_FREE 1
- #define CB_EXCEPTION_NAME(id) cb_exception_table[id].name
- #define CB_EXCEPTION_CODE(id) cb_exception_table[id].code
- #define CB_EXCEPTION_ENABLE(id) cb_exception_table[id].enable
- #define CB_FLAG(var, name, doc) extern int var;
- #define CB_WARNDEF(var, name, wall, doc) extern int var;
- #define CB_CONFIG_ANY(type, var, name) extern type var;
- #define CB_CONFIG_INT(var, name) extern int var;
- #define CB_CONFIG_STRING(var, name) extern const char *var;
- #define CB_CONFIG_BOOLEAN(var, name) extern int var;
- #define CB_CONFIG_SUPPORT(var, name) extern enum cb_support var;

Enumerations

- enum cb_assign_clause { CB_ASSIGN_COBOL2002, CB_ASSIGN_MF, CB_ASSIGN_IBM }
- enum cb_binary_byteorder { CB_BYTEORDER_NATIVE, CB_BYTEORDER_BIG_ENDIAN }
- enum cb_binary_size { CB_BINARY_SIZE_2_4_8, CB_BINARY_SIZE_1_2_4_8, CB_BINARY_SIZE_1__8 }
- enum cb_operation_type { CB_OPERATION_READ, CB_OPERATION_WRITE, CB_OPERATION_ASSIGN }
- enum cb_support {

  CB_OK, CB_WARNING, CB_ARCHAIC, CB_OBSOLETE,

  CB_SKIP, CB_IGNORE, CB_ERROR, CB_UNCONFORMABLE }

Functions

- struct cb_text_list * cb_text_list_add (struct cb_text_list *list, const char *name)
- void * cobc_malloc (const size_t size)
- void * cobc_realloc (void *prevptr, const size_t size)
- void cobc_abort (const char *filename, const int linenum)
- size_t cobc_check_valid_name (char *name)
- int cb_load_std (const char *name)
- int cb_load_conf (const char *fname, const int check_nodef, const int prefix_dir)
- int pplex (void)
- int ppparse (void)
- int ppopen (const char *name, struct cb_replace_list *replace_list)
- int ppcopy (const char *name, const char *lib, struct cb_replace_list *replace_list)
- void pp_set_replace_list (struct cb_replace_list *replace_list)
- int yylex (void)
- int yyparse (void)
- void cb_warning (const char *fmt,...)
- void cb_error (const char *fmt,...)
- int cb_verify (const enum cb_support tag, const char *feature)

Variables

- int cb_source_format
- int cb_display_sign
- struct cb_exception cb_exception_table []
- int cb_id
- int cb_attr_id
- int cb_literal_id
- int cb_field_id
- int cb_storage_id

- int cb_flag_main
- int errorcount
- int warningcount
- int alt_ebcdic
- int optimize_flag
- int has_external
- char * cb_oc_build_stamp
- char * cb_source_file
- int cb_source_line
- const char * cob_config_dir
- char * source_name
- char * demangle_name
- FILE * cb_storage_file
- char * cb_storage_file_name
- char ** cb_saveargv
- int cb_saveargc
- FILE * cb_listing_file
- FILE * cb_depend_file
- char * cb_depend_target
- struct cb_text_list * cb_depend_list
- struct cb_text_list * cb_include_list
- struct cb_text_list * cb_extension_list
- struct cb_program * current_program
- struct cb_statement * current_statement
- struct cb_label * current_section
- struct cb_label * current_paragraph
- size_t functions_are_all
- struct noreserve * norestab
- FILE * ppin
- FILE * ppout
- FILE * yyin
- FILE * yyout
- size_t sending_id
- size_t suppress_warn

7.3.1 Define Documentation

7.3.1.1 #define ABORT( ) cobc_abort(__FILE__,__LINE__)

Definition at line 37 of file cobc.h.

7.3.1.2 #define CB_CONFIG_ANY(type, var, name) extern type var;

Definition at line 198 of file cobc.h.

7.3.1.3 #define CB_CONFIG_BOOLEAN(var, name) extern int var;

Definition at line 201 of file cobc.h.

7.3.1.4 #define CB_CONFIG_INT(var, name) extern int var;

Definition at line 199 of file cobc.h.

7.3.1.5 #define CB_CONFIG_STRING(var, name) extern const char *var;

Definition at line 200 of file cobc.h.

7.3.1.6 #define CB_CONFIG_SUPPORT(var, name) extern enum cb_support var;

Definition at line 202 of file cobc.h.

7.3.1.7 #define CB_EXCEPTION_CODE(id) cb_exception_table[id].code

Definition at line 53 of file cobc.h.

7.3.1.8 #define CB_EXCEPTION_ENABLE(id) cb_exception_table[id].enable

Definition at line 54 of file cobc.h.

7.3.1.9 #define CB_EXCEPTION_NAME(id) cb_exception_table[id].name

Definition at line 52 of file cobc.h.

7.3.1.10 #define CB_FLAG(var, name, doc) extern int var;

Definition at line 57 of file cobc.h.

7.3.1.11 #define CB_FORMAT_FIXED 0

Definition at line 39 of file cobc.h.

7.3.1.12 #define CB_FORMAT_FREE 1

Definition at line 40 of file cobc.h.

7.3.1.13 #define CB_WARNDEF(var, name, *wall, doc*) extern int var;

Definition at line 62 of file cobc.h.

7.3.1.14 #define COB_NON_ALIGNED

Definition at line 30 of file cobc.h.

7.3.2 Enumeration Type Documentation

7.3.2.1 enum cb_assign_clause

Enumerator:

CB_ASSIGN_COBOL2002

CB_ASSIGN_MF

CB_ASSIGN_IBM

Definition at line 158 of file cobc.h.

```
                {
        CB_ASSIGN_COBOL2002,            /* COBOL 2002 standard */
        CB_ASSIGN_MF,                   /* Micro Focus COBOL compatibility */
        CB_ASSIGN_IBM                   /* IBM COBOL compatibility */
};
```

7.3.2.2 enum cb_binary_byteorder

Enumerator:

CB_BYTEORDER_NATIVE

CB_BYTEORDER_BIG_ENDIAN

Definition at line 164 of file cobc.h.

```
                    {
        CB_BYTEORDER_NATIVE,
        CB_BYTEORDER_BIG_ENDIAN
};
```

7.3.2.3 enum cb_binary_size

Enumerator:

CB_BINARY_SIZE_2_4_8

CB_BINARY_SIZE_1_2_4_8

CB_BINARY_SIZE_1__8

Definition at line 169 of file cobc.h.

```
                {
        CB_BINARY_SIZE_2_4_8,           /* 2,4,8 bytes */
        CB_BINARY_SIZE_1_2_4_8,         /* 1,2,4,8 bytes */
        CB_BINARY_SIZE_1__8             /* 1,2,3,4,5,6,7,8 bytes */
};
```

7.3.2.4 enum cb_operation_type

Enumerator:

CB_OPERATION_READ
CB_OPERATION_WRITE
CB_OPERATION_ASSIGN

Definition at line 175 of file cobc.h.

```
                  {
        CB_OPERATION_READ,
        CB_OPERATION_WRITE,
        CB_OPERATION_ASSIGN
};
```

7.3.2.5 enum cb_support

Enumerator:

CB_OK
CB_WARNING
CB_ARCHAIC
CB_OBSOLETE
CB_SKIP
CB_IGNORE
CB_ERROR
CB_UNCONFORMABLE

Definition at line 181 of file cobc.h.

```
           {
        CB_OK,
        CB_WARNING,
        CB_ARCHAIC,
        CB_OBSOLETE,
        CB_SKIP,
        CB_IGNORE,
        CB_ERROR,
        CB_UNCONFORMABLE
};
```

7.3.3 Function Documentation

7.3.3.1 void cb_error (const char * *fmt*, ...)

Definition at line 85 of file error.c.

```
{
        va_list ap;

        va_start (ap, fmt);
        print_error (NULL, 0, "Error: ", fmt, ap);
        va_end (ap);

        errorcount++;

}
```

7.3.3.2 int cb_load_conf (const char * *fname*, const int *check_nodef*, const int *prefix_dir*)

Definition at line 112 of file config.c.

```
{
        char                    *s;
        char                    *e;
        const char              *name;
        const char              *val;
        void                    *var;
        FILE                    *fp;
        char                    *nores;
        struct noreserve        *noresptr;
        int                     i;
        int                     j;
        int                     ret;
        int                     saveret;
        int                     line;
        char                    buff[COB_SMALL_BUFF];

        /* initialize the config table */
        if (check_nodef) {
                for (i = 0; config_table[i].name; i++) {
                        config_table[i].val = NULL;
                }
        }

        if (prefix_dir) {
                snprintf (buff, COB_SMALL_MAX, "%s/%s", cob_config_dir, fname);
                name = buff;
        } else {
                name = fname;
        }
        /* open the config file */
        fp = fopen (name, "r");
        if (fp == NULL) {
                perror (name);
                return -1;
        }
```

```
	/* read the config file */
	ret = 0;
	line = 0;
	while (fgets (buff, COB_SMALL_BUFF, fp)) {
		line++;

		/* skip comments */
		if (buff[0] == '#') {
			continue;
		}

		/* skip blank lines */
		for (s = buff; *s; s++) {
			if (isgraph (*s)) {
				break;
			}
		}
		if (!*s) {
			continue;
		}

		/* get the tag */
		s = strpbrk (buff, " \t:=");
		if (!s) {
			fprintf (stderr, "%s:%d: invalid line\n", fname, line);
			ret = -1;
			continue;
		}
		*s = 0;

		/* find the entry */
		for (i = 0; config_table[i].name; i++) {
			if (strcmp (buff, config_table[i].name) == 0) {
				break;
			}
		}
		if (!config_table[i].name) {
			fprintf (stderr, "%s:%d: unknown tag '%s'\n", fname, line
, buff);
			ret = -1;
			continue;
		}

		/* get the value */
		for (s++; *s && strchr (" \t:=", *s); s++) {
			;
		}
		e = s + strlen (s) - 1;
		for (; e >= s && strchr (" \t\r\n", *e); e--) {
			;
		}
		e[1] = 0;
		config_table[i].val = s;

		/* set the value */
		name = config_table[i].name;
		var = config_table[i].var;
		val = config_table[i].val;
		switch (config_table[i].type) {
		case ANY:
			if (strcmp (name, "assign-clause") == 0) {
				if (strcmp (val, "cobol2002") == 0) {
```

```
                                        unsupported_value (fname, line, val);
                                        ret = -1;
                                } else if (strcmp (val, "mf") == 0) {
                                        cb_assign_clause = CB_ASSIGN_MF;
                                } else if (strcmp (val, "ibm") == 0) {
                                        cb_assign_clause = CB_ASSIGN_IBM;
                                } else {
                                        invalid_value (fname, line, name);
                                        ret = -1;
                                }
                        } else if (strcmp (name, "binary-size") == 0) {
                                if (strcmp (val, "2-4-8") == 0) {
                                        cb_binary_size = CB_BINARY_SIZE_2_4_8;
                                } else if (strcmp (val, "1-2-4-8") == 0) {
                                        cb_binary_size = CB_BINARY_SIZE_1_2_4_8;
                                } else if (strcmp (val, "1--8") == 0) {
                                        cb_binary_size = CB_BINARY_SIZE_1__8;
                                } else {
                                        invalid_value (fname, line, name);
                                        ret = -1;
                                }
                        } else if (strcmp (name, "binary-byteorder") == 0) {
                                if (strcmp (val, "native") == 0) {
                                        cb_binary_byteorder =
CB_BYTEORDER_NATIVE;
                                } else if (strcmp (val, "big-endian") == 0) {
                                        cb_binary_byteorder =
CB_BYTEORDER_BIG_ENDIAN;
                                } else {
                                        invalid_value (fname, line, name);
                                        ret = -1;
                                }
                        }
                        break;
                case INT:
                        for (j = 0; val[j]; j++) {
                                if (!isdigit (val[j])) {
                                        invalid_value (fname, line, name);
                                        ret = -1;
                                        break;
                                }
                        }
                        *((int *)var) = atoi (val);
                        break;
                case STRING:
                        val = read_string (val);

                        if (strcmp (name, "include") == 0) {
                                /* include another conf file */
                                saveret = ret;
                                if (cb_load_conf (val, 0, 1) != 0) {
                                        return -1;
                                }
                                ret = saveret;
                        } else if (strcmp (name, "not-reserved") == 0) {
                                nores = read_string (val);
                                noresptr = cobc_malloc (sizeof (struct noreserve)
);
                                noresptr->noresword = cobc_malloc (strlen (nores)
 + 1);
                                strcpy (noresptr->noresword, nores);
                                noresptr->next = norestab;
```

```
                                norestab = noresptr;
                        } else {
                                *((const char **)var) = val;
                        }
                        break;
                case BOOLEAN:
                        if (strcmp (val, "yes") == 0) {
                                *((int *)var) = 1;
                        } else if (strcmp (val, "no") == 0) {
                                *((int *)var) = 0;
                        } else {
                                invalid_value (fname, line, name);
                                ret = -1;
                        }
                        break;
                case SUPPORT:
                        if (strcmp (val, "ok") == 0) {
                                *((enum cb_support *)var) = CB_OK;
                        } else if (strcmp (val, "warning") == 0) {
                                *((enum cb_support *)var) = CB_WARNING;
                        } else if (strcmp (val, "archaic") == 0) {
                                *((enum cb_support *)var) = CB_ARCHAIC;
                        } else if (strcmp (val, "obsolete") == 0) {
                                *((enum cb_support *)var) = CB_OBSOLETE;
                        } else if (strcmp (val, "skip") == 0) {
                                *((enum cb_support *)var) = CB_SKIP;
                        } else if (strcmp (val, "ignore") == 0) {
                                *((enum cb_support *)var) = CB_IGNORE;
                        } else if (strcmp (val, "error") == 0) {
                                *((enum cb_support *)var) = CB_ERROR;
                        } else if (strcmp (val, "unconformable") == 0) {
                                *((enum cb_support *)var) = CB_UNCONFORMABLE;
                        } else {
                                invalid_value (fname, line, name);
                                ret = -1;
                        }
                        break;
                default:
                        fprintf (stderr, _("%s:%d: invalid type for '%s'\n"),
                                 fname, line, name);
                        ret = -1;
                        break;
                }
        }
        fclose (fp);

        /* checks for no definition */
        if (check_nodef) {
                for (i = 2; config_table[i].name; i++) {
                        if (config_table[i].val == NULL) {
                                fprintf (stderr, "%s: no definition of '%s'\n",
                                         fname, config_table[i].name);
                                ret = -1;
                        }
                }
        }

        return ret;
}
```

7.3.3.3 int cb_load_std (const char * *name*)

Definition at line 106 of file config.c.

```
{
        return cb_load_conf (name, 1, 1);
}
```

7.3.3.4 struct cb_text_list* cb_text_list_add (struct cb_text_list * *list,* const char * *name*) [read]

Definition at line 355 of file cobc.c.

```
{
        struct cb_text_list *p;
        struct cb_text_list *l;

        p = cobc_malloc (sizeof (struct cb_text_list));
        p->text = strdup (text);
        p->next = NULL;
        if (!list) {
                return p;
        } else {
                for (l = list; l->next; l = l->next) { ; }
                l->next = p;
                return list;
        }
}
```

7.3.3.5 int cb_verify (const enum cb_support *tag,* const char * *feature*)

Definition at line 122 of file error.c.

```
{
        switch (tag) {
        case CB_OK:
                return 1;
        case CB_WARNING:
                return 1;
        case CB_ARCHAIC:
                if (cb_warn_archaic) {
                        cb_warning (_("%s is archaic in %s"), feature, cb_config_
      name);
                }
                return 1;
        case CB_OBSOLETE:
                if (cb_warn_obsolete) {
                        cb_warning (_("%s is obsolete in %s"), feature, cb_config
      _name);
                }
                return 1;
        case CB_SKIP:
                return 0;
        case CB_IGNORE:
```

```
                cb_warning (_("%s ignored"), feature);
                return 0;
        case CB_ERROR:
                return 0;
        case CB_UNCONFORMABLE:
                cb_error (_("%s does not conform to %s"), feature, cb_config_name
      );
                return 0;
        }
        return 0;
}
```

7.3.3.6 void cb_warning (const char * *fmt,* ...)

Definition at line 73 of file error.c.

```
{
        va_list ap;

        va_start (ap, fmt);
        print_error (NULL, 0, "Warning: ", fmt, ap);
        va_end (ap);

        warningcount++;
}
```

7.3.3.7 void cobc_abort (const char * *filename,* const int *linenum*)

Definition at line 310 of file cobc.c.

```
{
        fprintf (stderr, "%s:%d: Internal compiler error\n", filename, linenum);
        (void)longjmp (cob_jmpbuf, 1);
}
```

7.3.3.8 size_t cobc_check_valid_name (char * *name*)

Definition at line 373 of file cobc.c.

```
{
        size_t  n;

        for (n = 0; n < COB_NUM_CSYNS; ++n) {
                if (!strcmp (name, cob_csyns[n])) {
                        return 1;
                }
        }
        return 0;
}
```

7.3.3.9 void* cobc_malloc (const size_t *size*)

Definition at line 327 of file cobc.c.

```
{
        void *mptr;

        mptr = calloc (1, size);
        if (!mptr) {
                fprintf (stderr, "Cannot allocate %d bytes of memory - Aborting\n
      ", (int)size);
                fflush (stderr);
                (void)longjmp (cob_jmpbuf, 1);
        }
        return mptr;
}
```

7.3.3.10 void* cobc_realloc (void * *prevptr,* const size_t *size*)

Definition at line 341 of file cobc.c.

```
{
        void *mptr;

        mptr = realloc (prevptr, size);
        if (!mptr) {
                fprintf (stderr, "Cannot reallocate %d bytes of memory - Aborting
      \n", (int)size);
                fflush (stderr);
                (void)longjmp (cob_jmpbuf, 1);
        }
        return mptr;
}
```

7.3.3.11 void pp_set_replace_list (struct cb_replace_list * *replace_list*)

7.3.3.12 int ppcopy (const char * *name,* const char * *lib,* struct cb_replace_list * *replace_list*)

7.3.3.13 int pplex (void)

7.3.3.14 int ppopen (const char * *name,* struct cb_replace_list * *replace_list*)

7.3.3.15 int ppparse (void)

7.3.3.16 int yylex (void)

7.3.3.17 int yyparse (void)

7.3.4 Variable Documentation

7.3.4.1 int alt_ebcdic

Definition at line 121 of file cobc.c.

7.3.4.2 int cb_attr_id

Definition at line 113 of file cobc.c.

7.3.4.3 FILE* cb_depend_file

Definition at line 134 of file cobc.c.

7.3.4.4 struct cb_text_list* cb_depend_list

Definition at line 136 of file cobc.c.

7.3.4.5 char* cb_depend_target

Definition at line 135 of file cobc.c.

7.3.4.6 int cb_display_sign

Definition at line 84 of file cobc.c.

7.3.4.7 struct cb_exception cb_exception_table[ ]

7.3.4.8 struct cb_text_list* cb_extension_list

Definition at line 138 of file cobc.c.

7.3.4.9 int cb_field_id

Definition at line 115 of file cobc.c.

7.3.4.10 int cb_flag_main

Definition at line 117 of file cobc.c.

7.3.4.11 int cb_id

Definition at line 112 of file cobc.c.

7.3.4.12 struct cb_text_list* cb_include_list

Definition at line 137 of file cobc.c.

7.3.4.13 FILE* cb_listing_file

Definition at line 133 of file cobc.c.

7.3.4.14 int cb_literal_id

Definition at line 114 of file cobc.c.

7.3.4.15 char* cb_oc_build_stamp

Definition at line 125 of file cobc.c.

7.3.4.16 int cb_saveargc

Definition at line 140 of file cobc.c.

7.3.4.17 char** cb_saveargv

Definition at line 141 of file cobc.c.

7.3.4.18 char* cb_source_file

Definition at line 124 of file cobc.c.

7.3.4.19 int cb_source_format

Definition at line 79 of file cobc.c.

7.3.4.20 int cb_source_line

Definition at line 128 of file cobc.c.

7.3.4.21 FILE* cb_storage_file

Definition at line 130 of file cobc.c.

7.3.4.22 char* cb_storage_file_name

Definition at line 131 of file cobc.c.

7.3.4.23 int cb_storage_id

Definition at line 116 of file cobc.c.

7.3.4.24 const char* cob_config_dir

Definition at line 143 of file cobc.c.

7.3.4.25 struct cb_label* current_paragraph

Definition at line 986 of file parser.c.

7.3.4.26 struct cb_program* current_program

Definition at line 983 of file parser.c.

7.3.4.27 struct cb_label* current_section

Definition at line 985 of file parser.c.

7.3.4.28 struct cb_statement* current_statement

Definition at line 984 of file parser.c.

7.3.4.29 char* demangle_name

Definition at line 127 of file cobc.c.

7.3.4.30 int errorcount

Definition at line 119 of file cobc.c.

7.3.4.31 size_t functions_are_all

Definition at line 987 of file parser.c.

7.3.4.32 int has_external

Definition at line 138 of file codegen.c.

7.3.4.33 struct noreserve* norestab

Definition at line 52 of file config.c.

7.3.4.34 int optimize_flag

Definition at line 122 of file cobc.c.

7.3.4.35 FILE* ppin

7.3.4.36 FILE* ppout

7.3.4.37 size_t sending_id

Definition at line 74 of file typeck.c.

7.3.4.38 char* source_name

Definition at line 126 of file cobc.c.

7.3.4.39 size_t suppress_warn

Definition at line 75 of file typeck.c.

7.3.4.40 int warningcount

Definition at line 120 of file cobc.c.

7.3.4.41 FILE* yyin

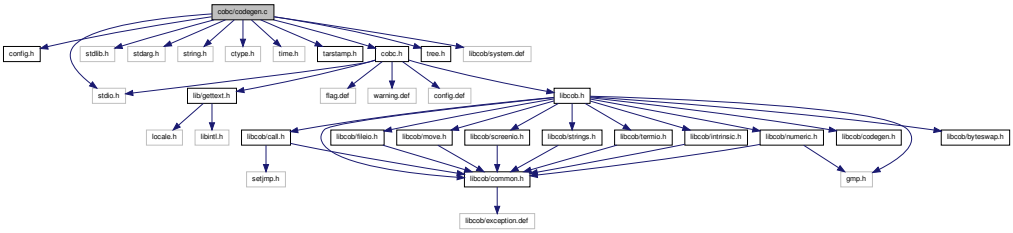
7.3.4.42 FILE* yyout

7.4 cobc/codegen.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
```

```
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <tarstamp.h>
#include "cobc.h"
#include "tree.h"
#include "libcob/system.def"
```

Include dependency graph for codegen.c:



Classes

- struct **label_list**
- struct **attr_list**
- struct **literal_list**
- struct **field_list**
- struct **call_list**
- struct **base_list**
- struct **local_list**
- struct sort_list
- struct system_table

Defines

- #define COB_USE_SETJMP 0
- #define COB_MAX_SUBSCRIPTS 16
- #define INITIALIZE_NONE 0
- #define INITIALIZE_ONE 1
- #define INITIALIZE_DEFAULT 2
- #define INITIALIZE_COMPOUND 3
- #define INITIALIZE_EXTERNAL 4
- #define COB_SYSTEM_GEN(x, y, z) { x, #z },

Functions

- void codegen (struct cb_program *prog, int nested)

Variables

- int has_external = 0

7.4.1 Define Documentation

7.4.1.1 #define COB_MAX_SUBSCRIPTS 16

Definition at line 36 of file codegen.c.

7.4.1.2 #define COB_SYSTEM_GEN(*x*, *y*, *z*) { x, #z },

7.4.1.3 #define COB_USE_SETJMP 0

Definition at line 35 of file codegen.c.

7.4.1.4 #define INITIALIZE_COMPOUND 3

Definition at line 41 of file codegen.c.

7.4.1.5 #define INITIALIZE_DEFAULT 2

Definition at line 40 of file codegen.c.

7.4.1.6 #define INITIALIZE_EXTERNAL 4

Definition at line 42 of file codegen.c.

7.4.1.7 #define INITIALIZE_NONE 0

Definition at line 38 of file codegen.c.

7.4.1.8 #define INITIALIZE_ONE 1

Definition at line 39 of file codegen.c.

7.4.2 Function Documentation

7.4.2.1 void codegen (struct cb_program * *prog*, int *nested*)

Definition at line 4683 of file codegen.c.

```
{
        int                     i;
        cb_tree                 l;
        struct attr_list        *j;
        struct literal_list     *m;
        struct field_list       *k;
        struct call_list        *clp;
        struct base_list        *blp;
        unsigned char           *s;
        struct cb_program       *cp;
        cb_tree                 l1;
        cb_tree                 l2;
        const char              *prevprog;
        time_t                  loctime;
        char                    locbuff[48];

        current_prog = prog;
        param_id = 0;
        stack_id = 0;
        num_cob_fields = 0;
        progid = 0;
        loop_counter = 0;
        output_indent_level = 0;
        last_line = 0;
        needs_exit_prog = 0;
        gen_custom = 0;
        call_cache = NULL;
        label_cache = NULL;
        local_cache = NULL;
        excp_current_program_id = prog->orig_source_name;
        excp_current_section = NULL;
        excp_current_paragraph = NULL;
        memset ((char *)i_counters, 0, sizeof (i_counters));

        output_target = yyout;

        if (!nested) {
                gen_ebcdic = 0;
                gen_ebcdic_ascii = 0;
                gen_full_ebcdic = 0;
                gen_native = 0;
                attr_cache = NULL;
                base_cache = NULL;
                literal_cache = NULL;
                field_cache = NULL;

                loctime = time (NULL);
                strftime (locbuff, sizeof(locbuff) - 1, "%b %d %Y %H:%M:%S %Z",
                        localtime (&loctime));
                output_header (output_target, locbuff);
                output_header (cb_storage_file, locbuff);
                for (cp = prog; cp; cp = cp->next_program) {
                        output_header (cp->local_storage_file, locbuff);
                }
```

```
                output_storage ("/* Frame stack declaration */\n");
                output_storage ("struct cob_frame {\n");
                output_storage ("\tint\tperform_through;\n");
#ifndef __GNUC__
                output_storage ("\tint\treturn_address;\n");
#elif   COB_USE_SETJMP
                output_storage ("\tjmp_buf\treturn_address;\n");
#else
                output_storage ("\tvoid\t*return_address;\n");
#endif
                output_storage ("};\n\n");
                output_storage ("/* Union for CALL statement */\n");
                output_storage ("union cob_call_union {\n");
                output_storage ("\tvoid *(*funcptr)();\n");
                output_storage ("\tint  (*funcint)();\n");
                output_storage ("\tvoid *func_void;\n");
                output_storage ("};\n");
                output_storage ("union cob_call_union\tcob_unifunc;\n\n");

                output ("#define  __USE_STRING_INLINES 1\n");
#ifdef  _XOPEN_SOURCE_EXTENDED
                output ("#ifndef        _XOPEN_SOURCE_EXTENDED\n");
                output ("#define  _XOPEN_SOURCE_EXTENDED 1\n");
                output ("#endif\n");
#endif
                output ("#include <stdio.h>\n");
                output ("#include <stdlib.h>\n");
                output ("#include <string.h>\n");
                output ("#include <math.h>\n");
#if     COB_USE_SETJMP
                output ("#include <setjmp.h>\n");
#endif
#ifdef  WORDS_BIGENDIAN
                output ("#define WORDS_BIGENDIAN 1\n");
#endif
#ifdef  HAVE_BUILTIN_EXPECT
                output ("#define HAVE_BUILTIN_EXPECT\n");
#endif
                if (optimize_flag) {
                        output ("#define COB_LOCAL_INLINE\n");
                }
                output ("#include <libcob.h>\n\n");

                output ("#define COB_SOURCE_FILE               \"%s\"\n",
     cb_source_file);
                output ("#define COB_PACKAGE_VERSION    \"%s\"\n",
     PACKAGE_VERSION);
                output ("#define COB_PATCH_LEVEL               %d\n\n",
     PATCH_LEVEL);
                output ("/* Global variables */\n");
                output ("#include \"%s\"\n\n", cb_storage_file_name);

                for (cp = prog; cp; cp = cp->next_program) {
                        if (cp->gen_decset) {
                                output("static void\n");
                                output("cob_decimal_set_int (cob_decimal *d, cons
     t int n)\n");
                                output("{\n");
                                output("        mpz_set_si (d->value, n);\n");
                                output("        d->scale = 0;\n");
                                output("}\n\n");
                                break;
```

```
                }
        }
        for (cp = prog; cp; cp = cp->next_program) {
                if (cp->gen_udecset) {
                        output("static void\n");
                        output("cob_decimal_set_uint (cob_decimal *d, con
st unsigned int n)\n");
                        output("{\n");
                        output("        mpz_set_ui (d->value, n);\n");
                        output("        d->scale = 0;\n");
                        output("}\n\n");
                        break;
                }
        }
        for (cp = prog; cp; cp = cp->next_program) {
                if (cp->gen_ptrmanip) {
                        output("static void\n");
                        output("cob_pointer_manip (cob_field *f1, cob_fie
ld *f2, size_t addsub)\n");
                        output("{\n");
                        output("        unsigned char   *tmptr;\n");
                        output("        memcpy (&tmptr, f1->data, sizeof(
void *));\n");
                        output("        if (addsub) {\n");
                        output("                tmptr -= cob_get_int (f2)
;\n");
                        output("        } else {\n");
                        output("                tmptr += cob_get_int (f2)
;\n");
                        output("        }\n");
                        output("        memcpy (f1->data, &tmptr, sizeof(
void *));\n");
                        output("}\n\n");
                        break;
                }
        }
        output ("/* Function prototypes */\n\n");
        for (cp = prog; cp; cp = cp->next_program) {
                /* Build parameter list */
                for (l = cp->entry_list; l; l = CB_CHAIN (l)) {
                        for (l1 = CB_VALUE (l); l1; l1 = CB_CHAIN (l1)) {

                                for (l2 = cp->parameter_list; l2; l2 =
CB_CHAIN (l2)) {
                                        if (strcasecmp (cb_field (
CB_VALUE (l1))->name,
                                                        cb_field (
CB_VALUE (l2))->name) == 0) {
                                                break;
                                        }
                                }
                                if (l2 == NULL) {
                                        cp->parameter_list = cb_list_add
(cp->parameter_list, CB_VALUE (l1));
                                }
                        }
                }
                if (cp->flag_main) {
                        output ("int %s ();\n", cp->program_id);
                } else {
                        for (l = cp->entry_list; l; l = CB_CHAIN (l)) {
                                output_entry_function (cp, l, cp->
```

```
parameter_list, 0);
                        }
                }
                output ("static int %s_ (const int", cp->program_id);
                if (!cp->flag_chained) {
                        for (l = cp->parameter_list; l; l = CB_CHAIN (l))
 {
                                output (", unsigned char *");
                        }
                }
                output (");\n");
        }
        output ("\n");
  }

  /* Class-names */
  if (!prog->nested_level && prog->class_name_list) {
        output ("/* Class names */\n");
        for (l = prog->class_name_list; l; l = CB_CHAIN (l)) {
                output_class_name_definition (CB_CLASS_NAME (CB_VALUE (l)
));
        }
  }

  /* Main function */
  if (prog->flag_main) {
        output_main_function (prog);
  }

  /* Functions */
  if (!nested) {
        output ("/* Functions */\n\n");
  }
  for (l = prog->entry_list; l; l = CB_CHAIN (l)) {
        output_entry_function (prog, l, prog->parameter_list, 1);
  }
  output_internal_function (prog, prog->parameter_list);

  if (!prog->next_program) {
        output ("/* End functions */\n\n");
  }

  if (gen_native || gen_full_ebcdic || gen_ebcdic_ascii || prog->
alphabet_name_list) {
        (void)lookup_attr (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL, 0);
  }

  output_target = cb_storage_file;

  /* Program local stuff */
  if (call_cache) {
        output_local ("\n/* Call pointers */\n");
        for (clp = call_cache; clp; clp = clp->next) {
                output_local ("static union cob_call_union\tcall_%s = { N
ULL };\n", clp->callname);
        }
        output_local ("\n");
  }

  for (i = 0; i < COB_MAX_SUBSCRIPTS; i++) {
        if (i_counters[i]) {
                output_local ("int\t\ti%d;\n", i);
```

```
                }
        }

        if (num_cob_fields) {
                output_local ("\n/* Local cob_field items */\n");
                for (i = 0; i < num_cob_fields; i++) {
                        output_local ("cob_field\tf%d;\n", i);
                }
                output_local ("\n");
        }

        /* Skip to next nested program */

        if (prog->next_program) {
                codegen (prog->next_program, 1);
                return;
        }

        /* Finalize the storage file */

        if (base_cache) {
                output_storage ("\n/* Storage */\n");
                base_cache = list_cache_sort (base_cache, &base_cache_cmp);
                prevprog = NULL;
                for (blp = base_cache; blp; blp = blp->next) {
                        if (blp->curr_prog != prevprog) {
                                prevprog = blp->curr_prog;
                                output_storage ("\n/* PROGRAM-ID : %s */\n", prev
      prog);
                        }
#ifdef HAVE_ATTRIBUTE_ALIGNED
                        output_storage ("static unsigned char %s%d[%d] __attribut
      e__((aligned));",
#else
                        output_storage ("static unsigned char %s%d[%d];",
#endif
                                        CB_PREFIX_BASE, blp->f->id,
                                        blp->f->memory_size);
                        output_storage ("\t/* %s */\n", blp->f->name);
                }
                output_storage ("\n/* End of storage */\n\n");
        }

        if (attr_cache) {
                output_storage ("\n/* Attributes */\n\n");
                attr_cache = attr_list_reverse (attr_cache);
                for (j = attr_cache; j; j = j->next) {
                        output_storage ("static const cob_field_attr %s%d = ",
                                        CB_PREFIX_ATTR, j->id);
                        output_storage ("{%d, %d, %d, %d, ", j->type, j->digits,
                                        j->scale, j->flags);
                        if (j->pic) {
                                output_storage ("\"");
                                for (s = j->pic; *s; s += 5) {
                                        output_storage ("%c\\%03o\\%03o\\%03o\\%0
      3o",
                                                        s[0], s[1], s[2], s[3], s[4]);
                                }
                                output_storage ("\"");
                        } else {
                                output_storage ("NULL");
                        }
```

```
                output_storage ("};\n");
        }
  }

  if (field_cache) {
        output_storage ("\n/* Fields */\n");
        field_cache = list_cache_sort (field_cache, &field_cache_cmp);
        prevprog = NULL;
        for (k = field_cache; k; k = k->next) {
                if (k->curr_prog != prevprog) {
                        prevprog = k->curr_prog;
                        output_storage ("\n/* PROGRAM-ID : %s */\n", prev
prog);
                }
                output ("static cob_field %s%d\t= ", CB_PREFIX_FIELD, k->
f->id);
                if (!k->f->flag_local && !k->f->flag_item_external) {
                        output_field (k->x);
                } else {
                        output ("{");
                        output_size (k->x);
                        output (", NULL, ");
                        output_attr (k->x);
                        output ("}");
                }
                output (";\t/* %s */\n", k->f->name);
        }
        output_storage ("\n/* End of fields */\n\n");
  }
  if (literal_cache) {
        output_storage ("/* Constants */\n");
        literal_cache = literal_list_reverse (literal_cache);
        for (m = literal_cache; m; m = m->next) {
                output ("static cob_field %s%d\t= ", CB_PREFIX_CONST, m->
id);
                output_field (m->x);
                output (";\n");
        }
        output ("\n");
  }

  if (gen_ebcdic) {
        output_storage ("/* EBCDIC translate table */\n");
        output ("static const unsigned char\tcob_a2e[256] = {\n");
        if (alt_ebcdic) {
                output ("\t0x00, 0x01, 0x02, 0x03, 0x37, 0x2D, 0x2E, 0x2F
,\n");
                output ("\t0x16, 0x05, 0x25, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
,\n");
                output ("\t0x10, 0x11, 0x12, 0x13, 0x3C, 0x3D, 0x32, 0x26
,\n");
                output ("\t0x18, 0x19, 0x3F, 0x27, 0x1C, 0x1D, 0x1E, 0x1F
,\n");
                output ("\t0x40, 0x5A, 0x7F, 0x7B, 0x5B, 0x6C, 0x50, 0x7D
,\n");
                output ("\t0x4D, 0x5D, 0x5C, 0x4E, 0x6B, 0x60, 0x4B, 0x61
,\n");
                output ("\t0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7
,\n");
                output ("\t0xF8, 0xF9, 0x7A, 0x5E, 0x4C, 0x7E, 0x6E, 0x6F
,\n");
                output ("\t0x7C, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7
```

```
,\n");
                output ("\t0xC8, 0xC9, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6
,\n");
                output ("\t0xD7, 0xD8, 0xD9, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6
,\n");
                output ("\t0xE7, 0xE8, 0xE9, 0xAD, 0xE0, 0xBD, 0x5F, 0x6D
,\n");
                output ("\t0x79, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87
,\n");
                output ("\t0x88, 0x89, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96
,\n");
                output ("\t0x97, 0x98, 0x99, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6
,\n");
                output ("\t0xA7, 0xA8, 0xA9, 0xC0, 0x6A, 0xD0, 0xA1, 0x07
,\n");
                output ("\t0x68, 0xDC, 0x51, 0x42, 0x43, 0x44, 0x47, 0x48
,\n");
                output ("\t0x52, 0x53, 0x54, 0x57, 0x56, 0x58, 0x63, 0x67
,\n");
                output ("\t0x71, 0x9C, 0x9E, 0xCB, 0xCC, 0xCD, 0xDB, 0xDD
,\n");
                output ("\t0xDF, 0xEC, 0xFC, 0xB0, 0xB1, 0xB2, 0x3E, 0xB4
,\n");
                output ("\t0x45, 0x55, 0xCE, 0xDE, 0x49, 0x69, 0x9A, 0x9B
,\n");
                output ("\t0xAB, 0x9F, 0xBA, 0xB8, 0xB7, 0xAA, 0x8A, 0x8B
,\n");
                output ("\t0xB6, 0xB5, 0x62, 0x4F, 0x64, 0x65, 0x66, 0x20
,\n");
                output ("\t0x21, 0x22, 0x70, 0x23, 0x72, 0x73, 0x74, 0xBE
,\n");
                output ("\t0x76, 0x77, 0x78, 0x80, 0x24, 0x15, 0x8C, 0x8D
,\n");
                output ("\t0x8E, 0x41, 0x06, 0x17, 0x28, 0x29, 0x9D, 0x2A
,\n");
                output ("\t0x2B, 0x2C, 0x09, 0x0A, 0xAC, 0x4A, 0xAE, 0xAF
,\n");
                output ("\t0x1B, 0x30, 0x31, 0xFA, 0x1A, 0x33, 0x34, 0x35
,\n");
                output ("\t0x36, 0x59, 0x08, 0x38, 0xBC, 0x39, 0xA0, 0xBF
,\n");
                output ("\t0xCA, 0x3A, 0xFE, 0x3B, 0x04, 0xCF, 0xDA, 0x14
,\n");
                output ("\t0xE1, 0x8F, 0x46, 0x75, 0xFD, 0xEB, 0xEE, 0xED
,\n");
                output ("\t0x90, 0xEF, 0xB3, 0xFB, 0xB9, 0xEA, 0xBB, 0xFF
\n");
        } else {
                /* MF */
                output ("\t0x00, 0x01, 0x02, 0x03, 0x1D, 0x19, 0x1A, 0x1B
,\n");
                output ("\t0x0F, 0x04, 0x16, 0x06, 0x07, 0x08, 0x09, 0x0A
,\n");
                output ("\t0x0B, 0x0C, 0x0D, 0x0E, 0x1E, 0x1F, 0x1C, 0x17
,\n");
                output ("\t0x10, 0x11, 0x20, 0x18, 0x12, 0x13, 0x14, 0x15
,\n");
                output ("\t0x21, 0x27, 0x3A, 0x36, 0x28, 0x30, 0x26, 0x38
,\n");
                output ("\t0x24, 0x2A, 0x29, 0x25, 0x2F, 0x2C, 0x22, 0x2D
,\n");
                output ("\t0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A
```

```
,\n");
                output ("\t0x7B, 0x7C, 0x35, 0x2B, 0x23, 0x39, 0x32, 0x33
,\n");
                output ("\t0x37, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D
,\n");
                output ("\t0x5E, 0x5F, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66
,\n");
                output ("\t0x67, 0x68, 0x69, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F
,\n");
                output ("\t0x70, 0x71, 0x72, 0x7D, 0x6A, 0x7E, 0x7F, 0x31
,\n");
                output ("\t0x34, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41
,\n");
                output ("\t0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49
,\n");
                output ("\t0x4A, 0x4B, 0x4C, 0x4E, 0x4F, 0x50, 0x51, 0x52
,\n");
                output ("\t0x53, 0x54, 0x55, 0x56, 0x2E, 0x60, 0x4D, 0x05
,\n");
                output ("\t0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87
,\n");
                output ("\t0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E, 0x8F
,\n");
                output ("\t0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97
,\n");
                output ("\t0x98, 0x99, 0x9A, 0x9B, 0x9C, 0x9D, 0x9E, 0x9F
,\n");
                output ("\t0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7
,\n");
                output ("\t0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF
,\n");
                output ("\t0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7
,\n");
                output ("\t0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF
,\n");
                output ("\t0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7
,\n");
                output ("\t0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF
,\n");
                output ("\t0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7
,\n");
                output ("\t0xD8, 0xD9, 0xDA, 0xDB, 0xDC, 0xDD, 0xDE, 0xDF
,\n");
                output ("\t0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7
,\n");
                output ("\t0xE8, 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 0xEE, 0xEF
,\n");
                output ("\t0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7
,\n");
                output ("\t0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF
\n");
        }
        output ("};\n");
        output_storage ("\n");
  }
  if (gen_full_ebcdic) {
        output ("static const unsigned char\tcob_ebcdic[256] = {\n");
        output ("\t0x00, 0x01, 0x02, 0x03, 0x37, 0x2D, 0x2E, 0x2F,\n");
        output ("\t0x16, 0x05, 0x25, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,\n");
        output ("\t0x10, 0x11, 0x12, 0x13, 0x3C, 0x3D, 0x32, 0x26,\n");
        output ("\t0x18, 0x19, 0x3F, 0x27, 0x1C, 0x1D, 0x1E, 0x1F,\n");
        output ("\t0x40, 0x5A, 0x7F, 0x7B, 0x5B, 0x6C, 0x50, 0x7D,\n");
```

```
            output ("\t0x4D, 0x5D, 0x5C, 0x4E, 0x6B, 0x60, 0x4B, 0x61,\n");
            output ("\t0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,\n");
            output ("\t0xF8, 0xF9, 0x7A, 0x5E, 0x4C, 0x7E, 0x6E, 0x6F,\n");
            output ("\t0x7C, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7,\n");
            output ("\t0xC8, 0xC9, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6,\n");
            output ("\t0xD7, 0xD8, 0xD9, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6,\n");
            output ("\t0xE7, 0xE8, 0xE9, 0xAD, 0xE0, 0xBD, 0x5F, 0x6D,\n");
            output ("\t0x79, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,\n");
            output ("\t0x88, 0x89, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96,\n");
            output ("\t0x97, 0x98, 0x99, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6,\n");
            output ("\t0xA7, 0xA8, 0xA9, 0xC0, 0x6A, 0xD0, 0xA1, 0x07,\n");
            output ("\t0x68, 0xDC, 0x51, 0x42, 0x43, 0x44, 0x47, 0x48,\n");
            output ("\t0x52, 0x53, 0x54, 0x57, 0x56, 0x58, 0x63, 0x67,\n");
            output ("\t0x71, 0x9C, 0x9E, 0xCB, 0xCC, 0xCD, 0xDB, 0xDD,\n");
            output ("\t0xDF, 0xEC, 0xFC, 0xB0, 0xB1, 0xB2, 0x3E, 0xB4,\n");
            output ("\t0x45, 0x55, 0xCE, 0xDE, 0x49, 0x69, 0x9A, 0x9B,\n");
            output ("\t0xAB, 0x9F, 0xBA, 0xB8, 0xB7, 0xAA, 0x8A, 0x8B,\n");
            output ("\t0xB6, 0xB5, 0x62, 0x4F, 0x64, 0x65, 0x66, 0x20,\n");
            output ("\t0x21, 0x22, 0x70, 0x23, 0x72, 0x73, 0x74, 0xBE,\n");
            output ("\t0x76, 0x77, 0x78, 0x80, 0x24, 0x15, 0x8C, 0x8D,\n");
            output ("\t0x8E, 0x41, 0x06, 0x17, 0x28, 0x29, 0x9D, 0x2A,\n");
            output ("\t0x2B, 0x2C, 0x09, 0x0A, 0xAC, 0x4A, 0xAE, 0xAF,\n");
            output ("\t0x1B, 0x30, 0x31, 0xFA, 0x1A, 0x33, 0x34, 0x35,\n");
            output ("\t0x36, 0x59, 0x08, 0x38, 0xBC, 0x39, 0xA0, 0xBF,\n");
            output ("\t0xCA, 0x3A, 0xFE, 0x3B, 0x04, 0xCF, 0xDA, 0x14,\n");
            output ("\t0xE1, 0x8F, 0x46, 0x75, 0xFD, 0xEB, 0xEE, 0xED,\n");
            output ("\t0x90, 0xEF, 0xB3, 0xFB, 0xB9, 0xEA, 0xBB, 0xFF\n");
            output ("};\n");
            i = lookup_attr (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL, 0);
            output
                ("static cob_field f_ebcdic = { 256, (unsigned char *)cob_ebc
dic, &%s%d };\n",
                 CB_PREFIX_ATTR, i);
            output_storage ("\n");
  }
  if (gen_ebcdic_ascii) {
            output ("static const unsigned char\tcob_ebcdic_ascii[256] = {\n"
);
            output ("\t0x00, 0x01, 0x02, 0x03, 0xEC, 0x09, 0xCA, 0x7F,\n");
            output ("\t0xE2, 0xD2, 0xD3, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,\n");
            output ("\t0x10, 0x11, 0x12, 0x13, 0xEF, 0xC5, 0x08, 0xCB,\n");
            output ("\t0x18, 0x19, 0xDC, 0xD8, 0x1C, 0x1D, 0x1E, 0x1F,\n");
            output ("\t0xB7, 0xB8, 0xB9, 0xBB, 0xC4, 0x0A, 0x17, 0x1B,\n");
            output ("\t0xCC, 0xCD, 0xCF, 0xD0, 0xD1, 0x05, 0x06, 0x07,\n");
            output ("\t0xD9, 0xDA, 0x16, 0xDD, 0xDE, 0xDF, 0xE0, 0x04,\n");
            output ("\t0xE3, 0xE5, 0xE9, 0xEB, 0x14, 0x15, 0x9E, 0x1A,\n");
            output ("\t0x20, 0xC9, 0x83, 0x84, 0x85, 0xA0, 0xF2, 0x86,\n");
            output ("\t0x87, 0xA4, 0xD5, 0x2E, 0x3C, 0x28, 0x2B, 0xB3,\n");
            output ("\t0x26, 0x82, 0x88, 0x89, 0x8A, 0xA1, 0x8C, 0x8B,\n");
            output ("\t0x8D, 0xE1, 0x21, 0x24, 0x2A, 0x29, 0x3B, 0x5E,\n");
            output ("\t0x2D, 0x2F, 0xB2, 0x8E, 0xB4, 0xB5, 0xB6, 0x8F,\n");
            output ("\t0x80, 0xA5, 0x7C, 0x2C, 0x25, 0x5F, 0x3E, 0x3F,\n");
            output ("\t0xBA, 0x90, 0xBC, 0xBD, 0xBE, 0xF3, 0xC0, 0xC1,\n");
            output ("\t0xC2, 0x60, 0x3A, 0x23, 0x40, 0x27, 0x3D, 0x22,\n");
            output ("\t0xC3, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,\n");
            output ("\t0x68, 0x69, 0xAE, 0xAF, 0xC6, 0xC7, 0xC8, 0xF1,\n");
            output ("\t0xF8, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F, 0x70,\n");
            output ("\t0x71, 0x72, 0xA6, 0xA7, 0x91, 0xCE, 0x92, 0xA9,\n");
            output ("\t0xE6, 0x7E, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,\n");
            output ("\t0x79, 0x7A, 0xAD, 0xA8, 0xD4, 0x5B, 0xD6, 0xD7,\n");
            output ("\t0x9B, 0x9C, 0x9D, 0xFA, 0x9F, 0xB1, 0xB0, 0xAC,\n");
            output ("\t0xAB, 0xFC, 0xAA, 0xFE, 0xE4, 0x5D, 0xBF, 0xE7,\n");
```

```
                output ("\t0x7B, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47,\n");
                output ("\t0x48, 0x49, 0xE8, 0x93, 0x94, 0x95, 0xA2, 0xED,\n");
                output ("\t0x7D, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F, 0x50,\n");
                output ("\t0x51, 0x52, 0xEE, 0x96, 0x81, 0x97, 0xA3, 0x98,\n");
                output ("\t0x5C, 0xF0, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,\n");
                output ("\t0x59, 0x5A, 0xFD, 0xF5, 0x99, 0xF7, 0xF6, 0xF9,\n");
                output ("\t0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,\n");
                output ("\t0x38, 0x39, 0xDB, 0xFB, 0x9A, 0xF4, 0xEA, 0xFF\n");
                output ("};\n");
                i = lookup_attr (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL, 0);
                output
                    ("static cob_field f_ebcdic_ascii = { 256, (unsigned char *)c
      ob_ebcdic_ascii, &%s%d };\n",
                     CB_PREFIX_ATTR, i);
                output_storage ("\n");
        }
        if (gen_native) {
                output ("static const unsigned char\tcob_native[256] = {\n");
                output ("\t0, 1, 2, 3, 4, 5, 6, 7,\n");
                output ("\t8, 9, 10, 11, 12, 13, 14, 15,\n");
                output ("\t16, 17, 18, 19, 20, 21, 22, 23,\n");
                output ("\t24, 25, 26, 27, 28, 29, 30, 31,\n");
                output ("\t32, 33, 34, 35, 36, 37, 38, 39,\n");
                output ("\t40, 41, 42, 43, 44, 45, 46, 47,\n");
                output ("\t48, 49, 50, 51, 52, 53, 54, 55,\n");
                output ("\t56, 57, 58, 59, 60, 61, 62, 63,\n");
                output ("\t64, 65, 66, 67, 68, 69, 70, 71,\n");
                output ("\t72, 73, 74, 75, 76, 77, 78, 79,\n");
                output ("\t80, 81, 82, 83, 84, 85, 86, 87,\n");
                output ("\t88, 89, 90, 91, 92, 93, 94, 95,\n");
                output ("\t96, 97, 98, 99, 100, 101, 102, 103,\n");
                output ("\t104, 105, 106, 107, 108, 109, 110, 111,\n");
                output ("\t112, 113, 114, 115, 116, 117, 118, 119,\n");
                output ("\t120, 121, 122, 123, 124, 125, 126, 127,\n");
                output ("\t128, 129, 130, 131, 132, 133, 134, 135,\n");
                output ("\t136, 137, 138, 139, 140, 141, 142, 143,\n");
                output ("\t144, 145, 146, 147, 148, 149, 150, 151,\n");
                output ("\t152, 153, 154, 155, 156, 157, 158, 159,\n");
                output ("\t160, 161, 162, 163, 164, 165, 166, 167,\n");
                output ("\t168, 169, 170, 171, 172, 173, 174, 175,\n");
                output ("\t176, 177, 178, 179, 180, 181, 182, 183,\n");
                output ("\t184, 185, 186, 187, 188, 189, 190, 191,\n");
                output ("\t192, 193, 194, 195, 196, 197, 198, 199,\n");
                output ("\t200, 201, 202, 203, 204, 205, 206, 207,\n");
                output ("\t208, 209, 210, 211, 212, 213, 214, 215,\n");
                output ("\t216, 217, 218, 219, 220, 221, 222, 223,\n");
                output ("\t224, 225, 226, 227, 228, 229, 230, 231,\n");
                output ("\t232, 233, 234, 235, 236, 237, 238, 239,\n");
                output ("\t240, 241, 242, 243, 244, 245, 246, 247,\n");
                output ("\t248, 249, 250, 251, 252, 253, 254, 255\n");
                output ("};\n");
                i = lookup_attr (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL, 0);
                output
                    ("static cob_field f_native = { 256, (unsigned char *)cob_nat
      ive, &%s%d };\n",
                     CB_PREFIX_ATTR, i);
                output_storage ("\n");
        }
}
```

7.4.3 Variable Documentation

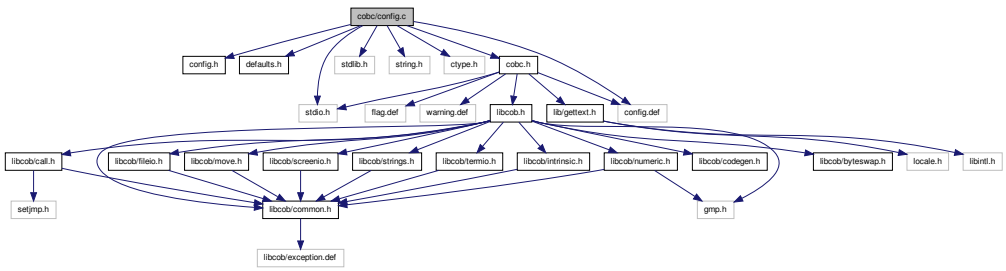
7.4.3.1 int has_external = 0

Definition at line 138 of file codegen.c.

7.5 cobc/config.c File Reference

```
#include "config.h"
#include "defaults.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "cobc.h"
#include "config.def"
```

Include dependency graph for config.c:



Defines

- #define CB_CONFIG_ANY(type, var, name) type var;
- #define CB_CONFIG_INT(var, name) int var;
- #define CB_CONFIG_STRING(var, name) const char *var;
- #define CB_CONFIG_BOOLEAN(var, name) int var;
- #define CB_CONFIG_SUPPORT(var, name) enum cb_support var;
- #define CB_CONFIG_ANY(type, var, name) {ANY, name, &var, NULL},
- #define CB_CONFIG_INT(var, name) {INT, name, &var, NULL},
- #define CB_CONFIG_STRING(var, name) {STRING, name, &var, NULL},
- #define CB_CONFIG_BOOLEAN(var, name) {BOOLEAN, name, &var, NULL},
- #define CB_CONFIG_SUPPORT(var, name) {SUPPORT, name, &var, NULL},

Enumerations

- enum cb_config_type {
  ANY, INT, STRING, BOOLEAN,
  SUPPORT }

Functions

- int cb_load_std (const char *name)
- int cb_load_conf (const char *fname, const int check_nodef, const int prefix_dir)

Variables

- struct noreserve * norestab = NULL

7.5.1 Define Documentation

7.5.1.1 #define CB_CONFIG_ANY(type, var, name) type var;

Definition at line 36 of file config.c.

7.5.1.2 #define CB_CONFIG_ANY(type, var, name) {ANY, name, &var, NULL},

Definition at line 36 of file config.c.

7.5.1.3 #define CB_CONFIG_BOOLEAN(var, name) int var;

Definition at line 39 of file config.c.

7.5.1.4 #define CB_CONFIG_BOOLEAN(var, name) {BOOLEAN, name, &var, NULL},

Definition at line 39 of file config.c.

7.5.1.5 #define CB_CONFIG_INT(var, name) int var;

Definition at line 37 of file config.c.

7.5.1.6 #define CB_CONFIG_INT(var, name) {INT, name, &var, NULL},

Definition at line 37 of file config.c.

7.5.1.7 #define CB_CONFIG_STRING(var, name) {STRING, name, &var, NULL},

Definition at line 38 of file config.c.

7.5.1.8 #define CB_CONFIG_STRING(var, name) const char *var;

Definition at line 38 of file config.c.

7.5.1.9 #define CB_CONFIG_SUPPORT(var, name) {SUPPORT, name, &var, NULL},

Definition at line 40 of file config.c.

7.5.1.10 #define CB_CONFIG_SUPPORT(var, name) enum cb_support var;

Definition at line 40 of file config.c.

7.5.2 Enumeration Type Documentation

7.5.2.1 enum cb_config_type

Enumerator:

ANY

INT

STRING

BOOLEAN

SUPPORT

Definition at line 43 of file config.c.

```
                {
        ANY,
        INT,                    /* integer */
        STRING,                 /* "..." */
        BOOLEAN,                /* 'yes', 'no' */
        SUPPORT                 /* 'ok', 'archaic', 'obsolete',
                                   'skip', 'ignore', 'unconformable' */
};
```

7.5.3 Function Documentation

7.5.3.1 int cb_load_conf (const char * *fname*, const int *check_nodef*, const int *prefix_dir*)

Definition at line 112 of file config.c.

```
{
        char                    *s;
        char                    *e;
        const char              *name;
        const char              *val;
        void                    *var;
        FILE                    *fp;
        char                    *nores;
        struct noreserve        *noresptr;
        int                     i;
        int                     j;
        int                     ret;
        int                     saveret;
        int                     line;
        char                    buff[COB_SMALL_BUFF];

        /* initialize the config table */
        if (check_nodef) {
                for (i = 0; config_table[i].name; i++) {
                        config_table[i].val = NULL;
                }
        }

        if (prefix_dir) {
                snprintf (buff, COB_SMALL_MAX, "%s/%s", cob_config_dir, fname);
                name = buff;
        } else {
                name = fname;
        }
        /* open the config file */
        fp = fopen (name, "r");
        if (fp == NULL) {
                perror (name);
                return -1;
        }

        /* read the config file */
        ret = 0;
        line = 0;
        while (fgets (buff, COB_SMALL_BUFF, fp)) {
                line++;

                /* skip comments */
                if (buff[0] == '#') {
                        continue;
                }

                /* skip blank lines */
                for (s = buff; *s; s++) {
                        if (isgraph (*s)) {
                                break;
                        }
                }
                if (!*s) {
                        continue;
                }

                /* get the tag */
                s = strpbrk (buff, " \t:=");
                if (!s) {
                        fprintf (stderr, "%s:%d: invalid line\n", fname, line);
                        ret = -1;
```

```
                continue;
        }
        *s = 0;

        /* find the entry */
        for (i = 0; config_table[i].name; i++) {
                if (strcmp (buff, config_table[i].name) == 0) {
                        break;
                }
        }
        if (!config_table[i].name) {
                fprintf (stderr, "%s:%d: unknown tag '%s'\n", fname, line
, buff);
                ret = -1;
                continue;
        }

        /* get the value */
        for (s++; *s && strchr (" \t:=", *s); s++) {
                ;
        }
        e = s + strlen (s) - 1;
        for (; e >= s && strchr (" \t\r\n", *e); e--) {
                ;
        }
        e[1] = 0;
        config_table[i].val = s;

        /* set the value */
        name = config_table[i].name;
        var = config_table[i].var;
        val = config_table[i].val;
        switch (config_table[i].type) {
        case ANY:
                if (strcmp (name, "assign-clause") == 0) {
                        if (strcmp (val, "cobol2002") == 0) {
                                unsupported_value (fname, line, val);
                                ret = -1;
                        } else if (strcmp (val, "mf") == 0) {
                                cb_assign_clause = CB_ASSIGN_MF;
                        } else if (strcmp (val, "ibm") == 0) {
                                cb_assign_clause = CB_ASSIGN_IBM;
                        } else {
                                invalid_value (fname, line, name);
                                ret = -1;
                        }
                } else if (strcmp (name, "binary-size") == 0) {
                        if (strcmp (val, "2-4-8") == 0) {
                                cb_binary_size = CB_BINARY_SIZE_2_4_8;
                        } else if (strcmp (val, "1-2-4-8") == 0) {
                                cb_binary_size = CB_BINARY_SIZE_1_2_4_8;
                        } else if (strcmp (val, "1--8") == 0) {
                                cb_binary_size = CB_BINARY_SIZE_1__8;
                        } else {
                                invalid_value (fname, line, name);
                                ret = -1;
                        }
                } else if (strcmp (name, "binary-byteorder") == 0) {
                        if (strcmp (val, "native") == 0) {
                                cb_binary_byteorder =
CB_BYTEORDER_NATIVE;
                        } else if (strcmp (val, "big-endian") == 0) {
```

```
                                        cb_binary_byteorder =
CB_BYTEORDER_BIG_ENDIAN;
                                } else {
                                        invalid_value (fname, line, name);
                                        ret = -1;
                                }
                        }
                        break;
                case INT:
                        for (j = 0; val[j]; j++) {
                                if (!isdigit (val[j])) {
                                        invalid_value (fname, line, name);
                                        ret = -1;
                                        break;
                                }
                        }
                        *((int *)var) = atoi (val);
                        break;
                case STRING:
                        val = read_string (val);

                        if (strcmp (name, "include") == 0) {
                                /* include another conf file */
                                saveret = ret;
                                if (cb_load_conf (val, 0, 1) != 0) {
                                        return -1;
                                }
                                ret = saveret;
                        } else if (strcmp (name, "not-reserved") == 0) {
                                nores = read_string (val);
                                noresptr = cobc_malloc (sizeof (struct noreserve)
);
                                noresptr->noresword = cobc_malloc (strlen (nores)
 + 1);
                                strcpy (noresptr->noresword, nores);
                                noresptr->next = norestab;
                                norestab = noresptr;
                        } else {
                                *((const char **)var) = val;
                        }
                        break;
                case BOOLEAN:
                        if (strcmp (val, "yes") == 0) {
                                *((int *)var) = 1;
                        } else if (strcmp (val, "no") == 0) {
                                *((int *)var) = 0;
                        } else {
                                invalid_value (fname, line, name);
                                ret = -1;
                        }
                        break;
                case SUPPORT:
                        if (strcmp (val, "ok") == 0) {
                                *((enum cb_support *)var) = CB_OK;
                        } else if (strcmp (val, "warning") == 0) {
                                *((enum cb_support *)var) = CB_WARNING;
                        } else if (strcmp (val, "archaic") == 0) {
                                *((enum cb_support *)var) = CB_ARCHAIC;
                        } else if (strcmp (val, "obsolete") == 0) {
                                *((enum cb_support *)var) = CB_OBSOLETE;
                        } else if (strcmp (val, "skip") == 0) {
                                *((enum cb_support *)var) = CB_SKIP;
```

```
                        } else if (strcmp (val, "ignore") == 0) {
                                *((enum cb_support *)var) = CB_IGNORE;
                        } else if (strcmp (val, "error") == 0) {
                                *((enum cb_support *)var) = CB_ERROR;
                        } else if (strcmp (val, "unconformable") == 0) {
                                *((enum cb_support *)var) = CB_UNCONFORMABLE;
                        } else {
                                invalid_value (fname, line, name);
                                ret = -1;
                        }
                        break;
                default:
                        fprintf (stderr, _("%s:%d: invalid type for '%s'\n"),
                                 fname, line, name);
                        ret = -1;
                        break;
                }
        }
        fclose (fp);

        /* checks for no definition */
        if (check_nodef) {
                for (i = 2; config_table[i].name; i++) {
                        if (config_table[i].val == NULL) {
                                fprintf (stderr, "%s: no definition of '%s'\n",
                                         fname, config_table[i].name);
                                ret = -1;
                        }
                }
        }

        return ret;
}
```

7.5.3.2 int cb_load_std (const char * *name*)

Definition at line 106 of file config.c.

```
{
        return cb_load_conf (name, 1, 1);
}
```

7.5.4 Variable Documentation

7.5.4.1 const char* name

Definition at line 56 of file config.c.

7.5.4.2 struct noreserve* norestab = NULL

Definition at line 52 of file config.c.

7.5.4.3 enum cb_config_type type

Definition at line 55 of file config.c.

7.5.4.4 char* val

Definition at line 58 of file config.c.

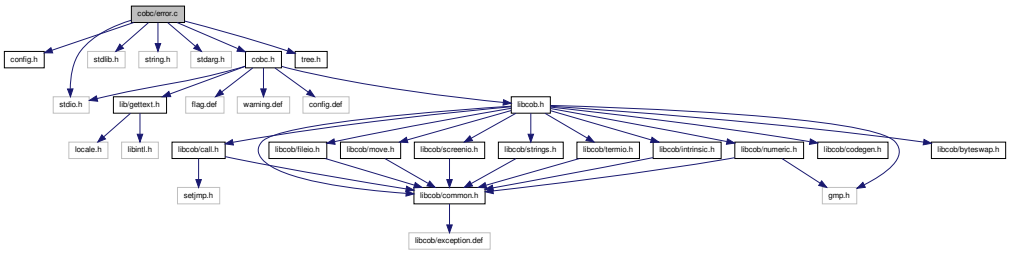
7.5.4.5 void* var

Definition at line 57 of file config.c.

7.6 cobc/error.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "cobc.h"
#include "tree.h"
```

Include dependency graph for error.c:



Functions

- char * check_filler_name (char *name)
- void cb_warning (const char *fmt,...)
- void cb_error (const char *fmt,...)
- void cb_warning_x (cb_tree x, const char *fmt,...)
- void cb_error_x (cb_tree x, const char *fmt,...)

- int cb_verify (const enum cb_support tag, const char *feature)
- void redefinition_error (cb_tree x)
- void redefinition_warning (cb_tree x, cb_tree y)
- void undefined_error (cb_tree x)
- void ambiguous_error (cb_tree x)
- void group_error (cb_tree x, const char *clause)
- void level_redundant_error (cb_tree x, const char *clause)
- void level_require_error (cb_tree x, const char *clause)
- void level_except_error (cb_tree x, const char *clause)

7.6.1 Function Documentation

7.6.1.1 void ambiguous_error (cb_tree *x*)

Definition at line 197 of file error.c.

```
{
        struct cb_word  *w;
        struct cb_field *p;
        struct cb_label *l2;
        cb_tree         l;
        cb_tree         y;

        w = CB_REFERENCE (x)->word;
        if (w->error == 0) {
                if (!errnamebuff) {
                        errnamebuff = cobc_malloc (COB_NORMAL_BUFF);
                }
                /* display error on the first time */
                snprintf (errnamebuff, COB_NORMAL_MAX, "'%s'", CB_NAME (x));
                for (l = CB_REFERENCE (x)->chain; l; l = CB_REFERENCE (l)->chain)
      {
                        strcat (errnamebuff, " in '");
                        strcat (errnamebuff, CB_NAME (l));
                        strcat (errnamebuff, "'");
                }
                cb_error_x (x, _("%s ambiguous; need qualification"), errnamebuff
    );
                w->error = 1;

                /* display all fields with the same name */
                for (l = w->items; l; l = CB_CHAIN (l)) {
                        y = CB_VALUE (l);
                        snprintf (errnamebuff, COB_NORMAL_MAX, "'%s' ", w->name);

                        switch (CB_TREE_TAG (y)) {
                        case CB_TAG_FIELD:
                                for (p = CB_FIELD (y)->parent; p; p = p->parent)
    {
                                        strcat (errnamebuff, "in '");
                                        strcat (errnamebuff, p->name);
                                        strcat (errnamebuff, "' ");
                                }
                                break;
                        case CB_TAG_LABEL:
                                l2 = CB_LABEL (y);
```

```
                                if (l2->section) {
                                        strcat (errnamebuff, "in '");
                                        strcat (errnamebuff, (const char *)(l2->
      section->name));
                                        strcat (errnamebuff, "' ");
                                }
                                break;
                        default:
                                break;
                        }
                        strcat (errnamebuff, _("defined here"));
                        cb_error_x (y, errnamebuff);
                }
        }
}
```

7.6.1.2 void cb_error (const char * *fmt*, ...)

Definition at line 85 of file error.c.

```
{
        va_list ap;

        va_start (ap, fmt);
        print_error (NULL, 0, "Error: ", fmt, ap);
        va_end (ap);

        errorcount++;

}
```

7.6.1.3 void cb_error_x (cb_tree *x*, const char * *fmt*, ...)

Definition at line 110 of file error.c.

```
{
        va_list ap;

        va_start (ap, fmt);
        print_error ((char *)(x->source_file), x->source_line, "Error: ", fmt, ap
      );
        va_end (ap);

        errorcount++;
}
```

7.6.1.4 int cb_verify (const enum cb_support *tag*, const char * *feature*)

Definition at line 122 of file error.c.

```
{
        switch (tag) {
```

```
        case CB_OK:
                return 1;
        case CB_WARNING:
                return 1;
        case CB_ARCHAIC:
                if (cb_warn_archaic) {
                        cb_warning (_("%s is archaic in %s"), feature, cb_config_
      name);
                }
                return 1;
        case CB_OBSOLETE:
                if (cb_warn_obsolete) {
                        cb_warning (_("%s is obsolete in %s"), feature, cb_config
      _name);
                }
                return 1;
        case CB_SKIP:
                return 0;
        case CB_IGNORE:
                cb_warning (_("%s ignored"), feature);
                return 0;
        case CB_ERROR:
                return 0;
        case CB_UNCONFORMABLE:
                cb_error (_("%s does not conform to %s"), feature, cb_config_name
      );
                return 0;
        }
        return 0;
}
```

7.6.1.5 void cb_warning (const char * *fmt*, ...)

Definition at line 73 of file error.c.

```
{
        va_list ap;

        va_start (ap, fmt);
        print_error (NULL, 0, "Warning: ", fmt, ap);
        va_end (ap);

        warningcount++;
}
```

7.6.1.6 void cb_warning_x (cb_tree *x*, const char * *fmt*, ...)

Definition at line 98 of file error.c.

```
{
        va_list ap;

        va_start (ap, fmt);
        print_error ((char *)(x->source_file), x->source_line, "Warning: ", fmt,
      ap);
```

```
        va_end (ap);

        warningcount++;
}
```

7.6.1.7 char* check_filler_name (char * *name*)

Definition at line 64 of file error.c.

```
{
        if (!memcmp (name, "WORK$", 5)) {
                name = (char *)"FILLER";
        }
        return name;
}
```

7.6.1.8 void group_error (cb_tree *x*, const char * *clause*)

Definition at line 250 of file error.c.

```
{
        cb_error_x (x, _("Group item '%s' cannot have %s clause"),
      check_filler_name (cb_name (x)), clause);
}
```

7.6.1.9 void level_except_error (cb_tree *x*, const char * *clause*)

Definition at line 270 of file error.c.

```
{
        cb_error_x (x, _("Level %02d item '%s' cannot have other than %s clause")
      ,
                    cb_field (x)->level, check_filler_name (cb_name (x)), clause)
      ;
}
```

7.6.1.10 void level_redundant_error (cb_tree *x*, const char * *clause*)

Definition at line 256 of file error.c.

```
{
        cb_error_x (x, _("Level %02d item '%s' cannot have %s clause"),
                    cb_field (x)->level, check_filler_name (cb_name (x)), clause)
      ;
}
```

7.6.1.11 void level_require_error (cb_tree *x*, const char * *clause*)

Definition at line 263 of file error.c.

```
{
        cb_error_x (x, _("Level %02d item '%s' requires %s clause"),
                    cb_field (x)->level, check_filler_name (cb_name (x)), clause)
      ;
}
```

7.6.1.12 void redefinition_error (cb_tree *x*)

Definition at line 154 of file error.c.

```
{
        struct cb_word  *w;

        w = CB_REFERENCE (x)->word;
        cb_error_x (x, _("Redefinition of '%s'"), w->name);
        cb_error_x (CB_VALUE (w->items), _("'%s' previously defined here"), w->
      name);
}
```

7.6.1.13 void redefinition_warning (cb_tree *x*, cb_tree *y*)

Definition at line 164 of file error.c.

```
{
        struct cb_word  *w;

        w = CB_REFERENCE (x)->word;
        cb_warning_x (x, _("Redefinition of '%s'"), w->name);
        if (y) {
                cb_warning_x (y, _("'%s' previously defined here"), w->name);
        } else {
                cb_warning_x (CB_VALUE (w->items), _("'%s' previously defined her
      e"), w->name);
        }
}
```

7.6.1.14 void undefined_error (cb_tree *x*)

Definition at line 178 of file error.c.

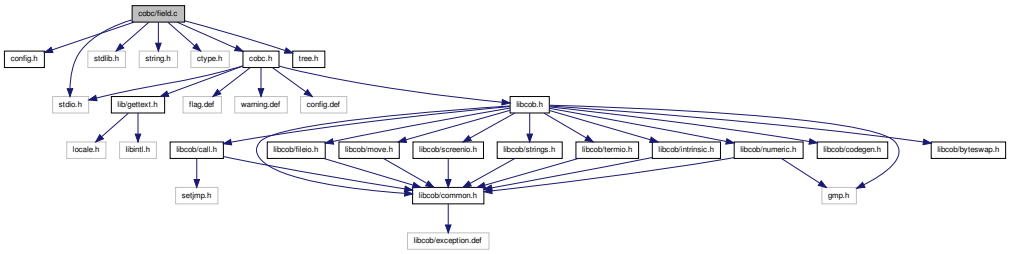
```
{
        struct cb_reference     *r;
        cb_tree                 c;

        if (!errnamebuff) {
                errnamebuff = cobc_malloc (COB_NORMAL_BUFF);
```

```
        }
        r = CB_REFERENCE (x);
        snprintf (errnamebuff, COB_NORMAL_MAX, "'%s'", CB_NAME (x));
        for (c = r->chain; c; c = CB_REFERENCE (c)->chain) {
                strcat (errnamebuff, " in '");
                strcat (errnamebuff, CB_NAME (c));
                strcat (errnamebuff, "'");
        }
        cb_error_x (x, _("%s undefined"), errnamebuff);
}
```

7.7 cobc/field.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "cobc.h"
#include "tree.h"
```

Include dependency graph for field.c:



Functions

- int cb_get_level (cb_tree x)
- cb_tree cb_build_field_tree (cb_tree level, cb_tree name, struct cb_field *last_field, enum cb_storage storage, struct cb_file *fn)
- struct cb_field * cb_resolve_redefines (struct cb_field *field, cb_tree redefines)
- void cb_validate_field (struct cb_field *f)
- void cb_validate_88_item (struct cb_field *f)
- struct cb_field * cb_validate_78_item (struct cb_field *f)
- void cb_clear_real_field (void)

Variables

- size_t cb_needs_01 = 0

7.7.1 Function Documentation

7.7.1.1 cb_tree cb_build_field_tree (cb_tree *level*, cb_tree *name*, struct cb_field * *last_field*, enum cb_storage *storage*, struct cb_file * *fn*)

Definition at line 78 of file field.c.

```
{
        struct cb_reference     *r;
        struct cb_field         *f;
        struct cb_field         *p;
        struct cb_field         *field_fill;
        cb_tree                 dummy_fill;
        cb_tree                 l;
        cb_tree                 x;
        int                     lv;

        if (level == cb_error_node || name == cb_error_node) {
                return cb_error_node;
        }

        /* check the level number */
        lv = cb_get_level (level);
        if (!lv) {
                return cb_error_node;
        }

        /* build the field */
        r = CB_REFERENCE (name);
        f = CB_FIELD (cb_build_field (name));
        f->storage = storage;
        last_real_field = last_field;
        if (lv == 78) {
                f->level = 01;
                f->flag_item_78 = 1;
                return CB_TREE (f);
        } else {
                f->level = lv;
        }
        if (f->level == 01 && storage == CB_STORAGE_FILE) {
                if (fn->external) {
                        f->flag_external = 1;
                        has_external = 1;
                } else if (fn->global) {
                        f->flag_is_global = 1;
                }
        }
        if (last_field) {
                if (last_field->level == 77 && f->level != 01 &&
                    f->level != 77 && f->level != 66 && f->level != 88) {
                        cb_error_x (name, _("Level number must begin with 01 or 7
      7"));
                        return cb_error_node;
                }
        }
```

```
        /* checks for redefinition */
        if (cb_warn_redefinition) {
                if (r->word->count > 1) {
                        if (f->level == 01 || f->level == 77) {
                                redefinition_warning (name, NULL);
                        } else {
                                for (l = r->word->items; l; l = CB_CHAIN (l)) {
                                        x = CB_VALUE (l);
                                        if (!CB_FIELD_P (x)
                                            || CB_FIELD (x)->level == 01
                                            || CB_FIELD (x)->level == 77
                                            || (f->level == last_field->level
                                                && CB_FIELD (x)->parent == last_f
      ield->parent)) {
                                                redefinition_warning (name, x);
                                                break;
                                        }
                                }
                        }
                }
        }

        if (last_field && last_field->level == 88) {
                last_field = last_field->parent;
        }

        /* link the field into the tree */
        if (f->level == 01 || f->level == 77) {
                /* top level */
                cb_needs_01 = 0;
                if (last_field) {
/*
                        cb_field_add (cb_field_founder (last_field), f);
*/
                        cb_field_founder (last_field)->sister = f;
                }
        } else if (!last_field || cb_needs_01) {
                /* invalid top level */
                cb_error_x (name, _("Level number must begin with 01 or 77"));
                return cb_error_node;
        } else if (f->level == 66) {
                /* level 66 */
                f->parent = cb_field_founder (last_field);
                for (p = f->parent->children; p && p->sister; p = p->sister) ;
                if (p) {
                        p->sister = f;
                }
        } else if (f->level == 88) {
                /* level 88 */
                f->parent = last_field;
        } else if (f->level > last_field->level) {
                /* lower level */
                last_field->children = f;
                f->parent = last_field;
        } else if (f->level == last_field->level) {
                /* same level */
same_level:
                last_field->sister = f;
                f->parent = last_field->parent;
        } else {
                /* upper level */
```

```
                for (p = last_field->parent; p; p = p->parent) {
                        if (p->level == f->level) {
                                last_field = p;
                                goto same_level;
                        }
                        if (cb_relax_level_hierarchy && p->level < f->level) {
                                break;
                        }
                }
                if (cb_relax_level_hierarchy) {
                        dummy_fill = cb_build_filler ();
                        field_fill = CB_FIELD (cb_build_field (dummy_fill));
                        cb_warning_x (name, _("No previous data item of level %02
      d"), f->level);
                        field_fill->level = f->level;
                        field_fill->storage = storage;
                        field_fill->children = p->children;
                        field_fill->parent = p;
                        for (p = p->children; p != NULL; p = p->sister) {
                                p->parent = field_fill;
                        }
                        field_fill->parent->children = field_fill;
                        field_fill->sister = f;
                        f->parent = field_fill->parent;
                        last_field = field_fill;
                } else {
                        cb_error_x (name, _("No previous data item of level %02d"
      ), f->level);
                        return cb_error_node;
                }
          }

          /* inherit parent's properties */
          if (f->parent) {
                  f->usage = f->parent->usage;
                  f->indexes = f->parent->indexes;
                  f->flag_sign_leading = f->parent->flag_sign_leading;
                  f->flag_sign_separate = f->parent->flag_sign_separate;
                  f->flag_is_global = f->parent->flag_is_global;
          }
          return CB_TREE (f);
}
```

7.7.1.2 void cb_clear_real_field (void)

Definition at line 1050 of file field.c.

```
{
        last_real_field = NULL;
}
```

7.7.1.3 int cb_get_level (cb_tree *x*)

Definition at line 41 of file field.c.

```
{
```

```
        const char      *p;
        const char      *name;
        int             level = 0;

        name = CB_NAME (x);
        /* get level */
        for (p = name; *p; p++) {
                if (!isdigit (*p)) {
                        goto level_error;
                }
                level = level * 10 + (*p - '0');
        }

        /* check level */
        switch (level) {
        case 66:
        case 77:
        case 78:
        case 88:
                break;
        default:
                if (level < 1 || level > 49) {
                        goto level_error;
                }
                break;
        }

        return level;

level_error:
        cb_error_x (x, _("Invalid level number '%s'"), name);
        return 0;
}
```

7.7.1.4 struct cb_field* cb_resolve_redefines (struct cb_field * *field,* cb_tree *redefines*) [read]

Definition at line 231 of file field.c.

```
{
        struct cb_field         *f;
        struct cb_reference     *r;
        const char              *name;
        cb_tree                 x;

        r = CB_REFERENCE (redefines);
        name = CB_NAME (redefines);
        x = CB_TREE (field);

        /* check qualification */
        if (r->chain) {
                cb_error_x (x, _("'%s' cannot be qualified here"), name);
                return NULL;
        }

        /* check subscripts */
        if (r->subs) {
                cb_error_x (x, _("'%s' cannot be subscripted here"), name);
                return NULL;
```

```
        }

        /* resolve the name in the current group (if any) */
        if (field->parent && field->parent->children) {
                for (f = field->parent->children; f; f = f->sister) {
                        if (strcasecmp (f->name, name) == 0) {
                                break;
                        }
                }
                if (f == NULL) {
                        cb_error_x (x, _("'%s' undefined in '%s'"), name, field->
      parent->name);
                        return NULL;
                }
        } else {
                if (cb_ref (redefines) == cb_error_node) {
                        return NULL;
                }
                f = cb_field (redefines);
        }

        /* check level number */
        if (f->level != field->level) {
                cb_error_x (x, _("Level number of REDEFINES entries must be ident
      ical"));
                return NULL;
        }
        if (f->level == 66 || f->level == 88) {
                cb_error_x (x, _("Level number of REDEFINES entry cannot be 66 or
       88"));
                return NULL;
        }

        if (!cb_indirect_redefines && f->redefines) {
                cb_error_x (x, _("'%s' not the original definition"), f->name);
                return NULL;
        }

        /* return the original definition */
        while (f->redefines) {
                f = f->redefines;
        }
        return f;
}
```

7.7.1.5 struct cb_field* cb_validate_78_item (struct cb_field * f) [read]

Definition at line 1033 of file field.c.

```
{
        cb_tree x;

        x = CB_TREE (f);
        if (!f->values) {
                level_require_error (x, "VALUE");
        }

        if (f->pic || f->flag_occurs) {
                level_except_error (x, "VALUE");
```

```
        }
        cb_add_78 (f);
        return last_real_field;
}
```

7.7.1.6 void cb_validate_88_item (struct cb_field * f)

Definition at line 1018 of file field.c.

```
{
        cb_tree x;

        x = CB_TREE (f);
        if (!f->values) {
                level_require_error (x, "VALUE");
        }

        if (f->pic || f->flag_occurs) {
                level_except_error (x, "VALUE");
        }
}
```

7.7.1.7 void cb_validate_field (struct cb_field * f)

Definition at line 973 of file field.c.

```
{
        struct cb_field         *c;

        if (validate_field_1 (f) != 0) {
                f->flag_invalid = 1;
                return;
        }
        /* RXW - Remove */
        if (f->flag_item_78) {
                f->flag_is_verified = 1;
                return;
        }

        /* setup parameters */
        if (f->storage == CB_STORAGE_LOCAL ||
            f->storage == CB_STORAGE_LINKAGE ||
            f->flag_item_based) {
                f->flag_local = 1;
        }
        if (f->storage == CB_STORAGE_LINKAGE || f->flag_item_based) {
                f->flag_base = 1;
        }
        setup_parameters (f);

        /* compute size */
        compute_size (f);
        if (!f->redefines) {
                f->memory_size = f->size * f->occurs_max;
        } else if (f->redefines->memory_size < f->size * f->occurs_max) {
```

```
                f->redefines->memory_size = f->size * f->occurs_max;
        }

        validate_field_value (f);
        if (f->flag_is_global) {
                f->count++;
                for (c = f->children; c; c = c->sister) {
                        c->flag_is_global = 1;
                        c->count++;
                }
        }
        f->flag_is_verified = 1;
}
```

7.7.2 Variable Documentation

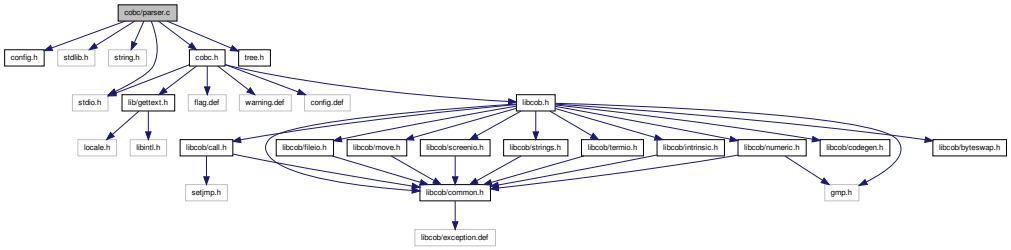
7.7.2.1 size_t cb_needs_01 = 0

Definition at line 33 of file field.c.

7.8 cobc/parser.c File Reference

```
#include "config.h"
#include <stdlib.h>
#include <string.h>
#include "cobc.h"
#include "tree.h"
#include <stdio.h>
```

Include dependency graph for parser.c:



Classes

- union yyalloc

Defines

- #define YYBISON 1
- #define YYSKELETON_NAME "yacc.c"
- #define YYPURE 0
- #define YYLSP_NEEDED 0
- #define YYTOKENTYPE
- #define TOKEN_EOF 0
- #define ACCEPT 258
- #define ACCESS 259
- #define ADD 260
- #define ADDRESS 261
- #define ADVANCING 262
- #define AFTER 263
- #define ALL 264
- #define ALLOCATE 265
- #define ALPHABET 266
- #define ALPHABETIC 267
- #define ALPHABETIC_LOWER 268
- #define ALPHABETIC_UPPER 269
- #define ALPHANUMERIC 270
- #define ALPHANUMERIC_EDITED 271
- #define ALSO 272
- #define ALTER 273
- #define ALTERNATE 274
- #define AND 275
- #define ANY 276
- #define ARE 277
- #define AREA 278
- #define ARGUMENT_NUMBER 279
- #define ARGUMENT_VALUE 280
- #define AS 281
- #define ASCENDING 282
- #define ASSIGN 283
- #define AT 284
- #define AUTO 285
- #define AUTOMATIC 286
- #define BACKGROUND_COLOR 287
- #define BASED 288
- #define BEFORE 289
- #define BELL 290
- #define BINARY 291
- #define BINARY_C_LONG 292
- #define BINARY_CHAR 293
- #define BINARY_DOUBLE 294
- #define BINARY_LONG 295
- #define BINARY_SHORT 296

- #define BLANK 297
- #define BLANK_LINE 298
- #define BLANK_SCREEN 299
- #define BLINK 300
- #define BLOCK 301
- #define BOTTOM 302
- #define BY 303
- #define BYTE_LENGTH 304
- #define CALL 305
- #define CANCEL 306
- #define CH 307
- #define CHAINING 308
- #define CHARACTER 309
- #define CHARACTERS 310
- #define CLASS 311
- #define CLOSE 312
- #define CODE 313
- #define CODE_SET 314
- #define COLLATING 315
- #define COL 316
- #define COLS 317
- #define COLUMN 318
- #define COLUMNS 319
- #define COMMA 320
- #define COMMAND_LINE 321
- #define COMMA_DELIM 322
- #define COMMIT 323
- #define COMMON 324
- #define COMP 325
- #define COMPUTE 326
- #define COMP_1 327
- #define COMP_2 328
- #define COMP_3 329
- #define COMP_4 330
- #define COMP_5 331
- #define COMP_X 332
- #define CONCATENATE_FUNC 333
- #define CONFIGURATION 334
- #define CONSTANT 335
- #define CONTAINS 336
- #define CONTENT 337
- #define CONTINUE 338
- #define CONTROL 339
- #define CONTROLS 340
- #define CONTROL_FOOTING 341
- #define CONTROL_HEADING 342

- #define CONVERTING 343
- #define CORRESPONDING 344
- #define COUNT 345
- #define CRT 346
- #define CURRENCY 347
- #define CURRENT_DATE_FUNC 348
- #define CURSOR 349
- #define CYCLE 350
- #define DATA 351
- #define DATE 352
- #define DAY 353
- #define DAY_OF_WEEK 354
- #define DE 355
- #define DEBUGGING 356
- #define DECIMAL_POINT 357
- #define DECLARATIVES 358
- #define DEFAULT 359
- #define DELETE 360
- #define DELIMITED 361
- #define DELIMITER 362
- #define DEPENDING 363
- #define DESCENDING 364
- #define DETAIL 365
- #define DISK 366
- #define DISPLAY 367
- #define DIVIDE 368
- #define DIVISION 369
- #define DOWN 370
- #define DUPLICATES 371
- #define DYNAMIC 372
- #define EBCDIC 373
- #define ELSE 374
- #define END 375
- #define END_ACCEPT 376
- #define END_ADD 377
- #define END_CALL 378
- #define END_COMPUTE 379
- #define END_DELETE 380
- #define END_DISPLAY 381
- #define END_DIVIDE 382
- #define END_EVALUATE 383
- #define END_FUNCTION 384
- #define END_IF 385
- #define END_MULTIPLY 386
- #define END_PERFORM 387
- #define END_PROGRAM 388

- #define END_READ 389
- #define END_RETURN 390
- #define END_REWRITE 391
- #define END_SEARCH 392
- #define END_START 393
- #define END_STRING 394
- #define END_SUBTRACT 395
- #define END_UNSTRING 396
- #define END_WRITE 397
- #define ENTRY 398
- #define ENVIRONMENT 399
- #define ENVIRONMENT_NAME 400
- #define ENVIRONMENT_VALUE 401
- #define EOL 402
- #define EOP 403
- #define EOS 404
- #define EQUAL 405
- #define EQUALS 406
- #define ERASE 407
- #define ERROR 408
- #define ESCAPE 409
- #define EVALUATE 410
- #define EVENT_STATUS 411
- #define EXCEPTION 412
- #define EXCLUSIVE 413
- #define EXIT 414
- #define EXTEND 415
- #define EXTERNAL 416
- #define FD 417
- #define FILE_CONTROL 418
- #define FILE_ID 419
- #define FILLER 420
- #define FINAL 421
- #define FIRST 422
- #define FOOTING 423
- #define FOR 424
- #define FOREGROUND_COLOR 425
- #define FOREVER 426
- #define FREE 427
- #define FROM 428
- #define FULL 429
- #define FUNCTION 430
- #define FUNCTION_ID 431
- #define FUNCTION_NAME 432
- #define GE 433
- #define GENERATE 434

- #define GIVING 435
- #define GLOBAL 436
- #define GO 437
- #define GOBACK 438
- #define GREATER 439
- #define GROUP 440
- #define HEADING 441
- #define HIGHLIGHT 442
- #define HIGH_VALUE 443
- #define IDENTIFICATION 444
- #define IF 445
- #define IGNORE 446
- #define IGNORING 447
- #define IN 448
- #define INDEX 449
- #define INDEXED 450
- #define INDICATE 451
- #define INITIALIZE 452
- #define INITIALIZED 453
- #define INITIATE 454
- #define INPUT 455
- #define INPUT_OUTPUT 456
- #define INSPECT 457
- #define INTO 458
- #define INTRINSIC 459
- #define INVALID 460
- #define INVALID_KEY 461
- #define IS 462
- #define I_O 463
- #define I_O_CONTROL 464
- #define JUSTIFIED 465
- #define KEY 466
- #define LABEL 467
- #define LAST 468
- #define LAST_DETAIL 469
- #define LE 470
- #define LEADING 471
- #define LEFT 472
- #define LENGTH 473
- #define LESS 474
- #define LIMIT 475
- #define LIMITS 476
- #define LINAGE 477
- #define LINAGE_COUNTER 478
- #define LINE 479
- #define LINES 480

- #define OPEN 527
- #define OPTIONAL 528
- #define OR 529
- #define ORDER 530
- #define ORGANIZATION 531
- #define OTHER 532
- #define OUTPUT 533
- #define OVERFLOW 534
- #define OVERLINE 535
- #define PACKED_DECIMAL 536
- #define PADDING 537
- #define PAGE 538
- #define PAGE_FOOTING 539
- #define PAGE_HEADING 540
- #define PARAGRAPH 541
- #define PERFORM 542
- #define PICTURE 543
- #define PLUS 544
- #define POINTER 545
- #define POSITION 546
- #define POSITIVE 547
- #define PRESENT 548
- #define PREVIOUS 549
- #define PRINTER 550
- #define PRINTING 551
- #define PROCEDURE 552
- #define PROCEDURES 553
- #define PROCEED 554
- #define PROGRAM 555
- #define PROGRAM_ID 556
- #define PROGRAM_NAME 557
- #define PROGRAM_POINTER 558
- #define PROMPT 559
- #define QUOTE 560
- #define RANDOM 561
- #define RD 562
- #define READ 563
- #define RECORD 564
- #define RECORDING 565
- #define RECORDS 566
- #define RECURSIVE 567
- #define REDEFINES 568
- #define REEL 569
- #define REFERENCE 570
- #define RELATIVE 571
- #define RELEASE 572

- #define REMAINDER 573
- #define REMOVAL 574
- #define RENAMES 575
- #define REPLACING 576
- #define REPORT 577
- #define REPORTING 578
- #define REPORTS 579
- #define REPORT_FOOTING 580
- #define REPORT_HEADING 581
- #define REPOSITORY 582
- #define REQUIRED 583
- #define RESERVE 584
- #define RETURN 585
- #define RETURNING 586
- #define REVERSE_FUNC 587
- #define REVERSE_VIDEO 588
- #define REWIND 589
- #define REWRITE 590
- #define RIGHT 591
- #define ROLLBACK 592
- #define ROUNDED 593
- #define RUN 594
- #define SAME 595
- #define SCREEN 596
- #define SCREEN_CONTROL 597
- #define SCROLL 598
- #define SD 599
- #define SEARCH 600
- #define SECTION 601
- #define SECURE 602
- #define SEGMENT_LIMIT 603
- #define SELECT 604
- #define SEMI_COLON 605
- #define SENTENCE 606
- #define SEPARATE 607
- #define SEQUENCE 608
- #define SEQUENTIAL 609
- #define SET 610
- #define SHARING 611
- #define SIGN 612
- #define SIGNED 613
- #define SIGNED_INT 614
- #define SIGNED_LONG 615
- #define SIGNED_SHORT 616
- #define SIZE 617
- #define SIZE_ERROR 618

- #define SORT 619
- #define SORT_MERGE 620
- #define SOURCE 621
- #define SOURCE_COMPUTER 622
- #define SPACE 623
- #define SPECIAL_NAMES 624
- #define STANDARD 625
- #define STANDARD_1 626
- #define STANDARD_2 627
- #define START 628
- #define STATUS 629
- #define STOP 630
- #define STRING 631
- #define SUBSTITUTE_FUNC 632
- #define SUBSTITUTE_CASE_FUNC 633
- #define SUBTRACT 634
- #define SUM 635
- #define SUPPRESS 636
- #define SYMBOLIC 637
- #define SYNCHRONIZED 638
- #define TALLYING 639
- #define TAPE 640
- #define TERMINATE 641
- #define TEST 642
- #define THAN 643
- #define THEN 644
- #define THRU 645
- #define TIME 646
- #define TIMES 647
- #define TO 648
- #define TOK_FALSE 649
- #define TOK_FILE 650
- #define TOK_INITIAL 651
- #define TOK_NULL 652
- #define TOK_TRUE 653
- #define TOP 654
- #define TRAILING 655
- #define TRANSFORM 656
- #define TRIM_FUNCTION 657
- #define TYPE 658
- #define UNDERLINE 659
- #define UNIT 660
- #define UNLOCK 661
- #define UNSIGNED 662
- #define UNSIGNED_INT 663
- #define UNSIGNED_LONG 664

- #define UNSIGNED_SHORT 665
- #define UNSTRING 666
- #define UNTIL 667
- #define UP 668
- #define UPDATE 669
- #define UPON 670
- #define UPON_ARGUMENT_NUMBER 671
- #define UPON_COMMAND_LINE 672
- #define UPON_ENVIRONMENT_NAME 673
- #define UPON_ENVIRONMENT_VALUE 674
- #define UPPER_CASE_FUNC 675
- #define USAGE 676
- #define USE 677
- #define USING 678
- #define VALUE 679
- #define VARYING 680
- #define WAIT 681
- #define WHEN 682
- #define WHEN_COMPILED_FUNC 683
- #define WITH 684
- #define WORD 685
- #define WORDS 686
- #define WORKING_STORAGE 687
- #define WRITE 688
- #define YYYYDDD 689
- #define YYYYMMDD 690
- #define ZERO 691
- #define UNARY_SIGN 692
- #define yyerror cb_error
- #define YYDEBUG 1
- #define YYERROR_VERBOSE 1
- #define PENDING(x) cb_warning (_("'%s' not implemented"), x)
- #define emit_statement(x) current_program->exec_list = cb_cons (x, current_program->exec_list)
- #define push_expr(type, node) current_expr = cb_build_list (cb_int (type), node, current_expr)
- #define TERM_NONE 0
- #define TERM_ACCEPT 1
- #define TERM_ADD 2
- #define TERM_CALL 3
- #define TERM_COMPUTE 4
- #define TERM_DELETE 5
- #define TERM_DISPLAY 6
- #define TERM_DIVIDE 7
- #define TERM_EVALUATE 8
- #define TERM_IF 9

- #define TERM_MULTIPLY 10
- #define TERM_PERFORM 11
- #define TERM_READ 12
- #define TERM_RECEIVE 13
- #define TERM_RETURN 14
- #define TERM_REWRITE 15
- #define TERM_SEARCH 16
- #define TERM_START 17
- #define TERM_STRING 18
- #define TERM_SUBTRACT 19
- #define TERM_UNSTRING 20
- #define TERM_WRITE 21
- #define TERM_MAX 22
- #define YYERROR_VERBOSE 1
- #define YYSTACK_ALLOC malloc
- #define YYSTACK_FREE free
- #define YYSTACK_GAP_MAXIMUM (sizeof (union yyalloc) - 1)
- #define YYSTACK_BYTES(N)
- #define YYCOPY(To, From, Count)
- #define YYSTACK_RELOCATE(Stack)
- #define YYFINAL 3
- #define YYLAST 5462
- #define YYNTOKENS 451
- #define YYNNTS 690
- #define YYNRULES 1518
- #define YYNSTATES 2240
- #define YYUNDEFTOK 2
- #define YYMAXUTOK 692
- #define YYTRANSLATE(YYX) ((unsigned int) (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)
- #define YYPACT_NINF -1922
- #define YYTABLE_NINF -1518
- #define YYSIZE_T unsigned int
- #define yyerrok (yyerrstatus = 0)
- #define yyclearin (yychar = YYEMPTY)
- #define YYEMPTY (-2)
- #define YYEOF 0
- #define YYACCEPT goto yyacceptlab
- #define YYABORT goto yyabortlab
- #define YYERROR goto yyerrlab1
- #define YYFAIL goto yyerrlab
- #define YYRECOVERING() (!!yyerrstatus)
- #define YYBACKUP(Token, Value)
- #define YYTERROR 1
- #define YYERRCODE 256
- #define YYLLOC_DEFAULT(Current, Rhs, N)

- #define YYLEX yylex ()
- #define YYFPRINTF fprintf
- #define YYDPRINTF(Args)
- #define YYDSYMPRINT(Args)
- #define YYDSYMPRINTF(Title, Token, Value, Location)
- #define YY_STACK_PRINT(Bottom, Top)
- #define YY_REDUCE_PRINT(Rule)
- #define YYINITDEPTH 200
- #define YYMAXDEPTH 10000
- #define YYPOPSTACK (yyvsp--, yyssp--)

Typedefs

- typedef short yysigned_char

Enumerations

- enum yytokentype {

  TOKEN_EOF = 0, ACCEPT = 258, ACCESS = 259, ADD = 260,

  ADDRESS = 261, ADVANCING = 262, AFTER = 263, ALL = 264,

  ALLOCATE = 265, ALPHABET = 266, ALPHABETIC = 267, ALPHABETIC_-
  LOWER = 268,

  ALPHABETIC_UPPER = 269, ALPHANUMERIC = 270, ALPHANUMERIC_EDITED
  = 271, ALSO = 272,

  ALTER = 273, ALTERNATE = 274, AND = 275, ANY = 276,

  ARE = 277, AREA = 278, ARGUMENT_NUMBER = 279, ARGUMENT_VALUE
  = 280,

  AS = 281, ASCENDING = 282, ASSIGN = 283, AT = 284,

  AUTO = 285, AUTOMATIC = 286, BACKGROUND_COLOR = 287, BASED = 288,

  BEFORE = 289, BELL = 290, BINARY = 291, BINARY_C_LONG = 292,

  BINARY_CHAR = 293, BINARY_DOUBLE = 294, BINARY_LONG = 295, BINARY_-
  SHORT = 296,

  BLANK = 297, BLANK_LINE = 298, BLANK_SCREEN = 299, BLINK = 300,

  BLOCK = 301, BOTTOM = 302, BY = 303, BYTE_LENGTH = 304,

  CALL = 305, CANCEL = 306, CH = 307, CHAINING = 308,

  CHARACTER = 309, CHARACTERS = 310, CLASS = 311, CLOSE = 312,

  CODE = 313, CODE_SET = 314, COLLATING = 315, COL = 316,

  COLS = 317, COLUMN = 318, COLUMNS = 319, COMMA = 320,

  COMMAND_LINE = 321, COMMA_DELIM = 322, COMMIT = 323, COMMON =
  324,

  COMP = 325, COMPUTE = 326, COMP_1 = 327, COMP_2 = 328,

COMP_3 = 329, COMP_4 = 330, COMP_5 = 331, COMP_X = 332,

CONCATENATE_FUNC = 333, CONFIGURATION = 334, CONSTANT = 335, CONTAINS = 336,

CONTENT = 337, CONTINUE = 338, CONTROL = 339, CONTROLS = 340,

CONTROL_FOOTING = 341, CONTROL_HEADING = 342, CONVERTING = 343, CORRESPONDING = 344,

COUNT = 345, CRT = 346, CURRENCY = 347, CURRENT_DATE_FUNC = 348,

CURSOR = 349, CYCLE = 350, DATA = 351, DATE = 352,

DAY = 353, DAY_OF_WEEK = 354, DE = 355, DEBUGGING = 356,

DECIMAL_POINT = 357, DECLARATIVES = 358, DEFAULT = 359, DELETE = 360,

DELIMITED = 361, DELIMITER = 362, DEPENDING = 363, DESCENDING = 364,

DETAIL = 365, DISK = 366, DISPLAY = 367, DIVIDE = 368,

DIVISION = 369, DOWN = 370, DUPLICATES = 371, DYNAMIC = 372,

EBCDIC = 373, ELSE = 374, END = 375, END_ACCEPT = 376,

END_ADD = 377, END_CALL = 378, END_COMPUTE = 379, END_DELETE = 380,

END_DISPLAY = 381, END_DIVIDE = 382, END_EVALUATE = 383, END_-FUNCTION = 384,

END_IF = 385, END_MULTIPLY = 386, END_PERFORM = 387, END_PROGRAM = 388,

END_READ = 389, END_RETURN = 390, END_REWRITE = 391, END_SEARCH = 392,

END_START = 393, END_STRING = 394, END_SUBTRACT = 395, END_UNSTRING = 396,

END_WRITE = 397, ENTRY = 398, ENVIRONMENT = 399, ENVIRONMENT_-NAME = 400,

ENVIRONMENT_VALUE = 401, EOL = 402, EOP = 403, EOS = 404,

EQUAL = 405, EQUALS = 406, ERASE = 407, ERROR = 408,

ESCAPE = 409, EVALUATE = 410, EVENT_STATUS = 411, EXCEPTION = 412,

EXCLUSIVE = 413, EXIT = 414, EXTEND = 415, EXTERNAL = 416,

FD = 417, FILE_CONTROL = 418, FILE_ID = 419, FILLER = 420,

FINAL = 421, FIRST = 422, FOOTING = 423, FOR = 424,

FOREGROUND_COLOR = 425, FOREVER = 426, FREE = 427, FROM = 428,

FULL = 429, FUNCTION = 430, FUNCTION_ID = 431, FUNCTION_NAME = 432,

GE = 433, GENERATE = 434, GIVING = 435, GLOBAL = 436,

GO = 437, GOBACK = 438, GREATER = 439, GROUP = 440,

HEADING = 441, HIGHLIGHT = 442, HIGH_VALUE = 443, IDENTIFICATION = 444,

IF = 445, IGNORE = 446, IGNORING = 447, IN = 448,

INDEX = 449, INDEXED = 450, INDICATE = 451, INITIALIZE = 452,

INITIALIZED = 453, INITIATE = 454, INPUT = 455, INPUT_OUTPUT = 456,

INSPECT = 457, INTO = 458, INTRINSIC = 459, INVALID = 460,

INVALID_KEY = 461, IS = 462, I_O = 463, I_O_CONTROL = 464,

JUSTIFIED = 465, KEY = 466, LABEL = 467, LAST = 468,

LAST_DETAIL = 469, LE = 470, LEADING = 471, LEFT = 472,

LENGTH = 473, LESS = 474, LIMIT = 475, LIMITS = 476,

LINAGE = 477, LINAGE_COUNTER = 478, LINE = 479, LINES = 480,

LINKAGE = 481, LITERAL = 482, LOCALE = 483, LOCALE_DT_FUNC = 484,

LOCAL_STORAGE = 485, LOCK = 486, LOWER_CASE_FUNC = 487, LOW-LIGHT = 488,

LOW_VALUE = 489, MANUAL = 490, MEMORY = 491, MERGE = 492,

MINUS = 493, MNEMONIC_NAME = 494, MODE = 495, MOVE = 496,

MULTIPLE = 497, MULTIPLY = 498, NATIONAL = 499, NATIONAL_EDITED = 500,

NATIVE = 501, NE = 502, NEGATIVE = 503, NEXT = 504,

NEXT_SENTENCE = 505, NO = 506, NOT = 507, NOT_END = 508,

NOT_EOP = 509, NOT_EXCEPTION = 510, NOT_INVALID_KEY = 511, NOT_-OVERFLOW = 512,

NOT_SIZE_ERROR = 513, NO_ADVANCING = 514, NUMBER = 515, NUM-BERS = 516,

NUMERIC = 517, NUMERIC_EDITED = 518, NUMVALC_FUNC = 519, OBJECT_-COMPUTER = 520,

OCCURS = 521, OF = 522, OFF = 523, OMITTED = 524,

ON = 525, ONLY = 526, OPEN = 527, OPTIONAL = 528,

OR = 529, ORDER = 530, ORGANIZATION = 531, OTHER = 532,

OUTPUT = 533, OVERFLOW = 534, OVERLINE = 535, PACKED_DECIMAL = 536,

PADDING = 537, PAGE = 538, PAGE_FOOTING = 539, PAGE_HEADING = 540,

PARAGRAPH = 541, PERFORM = 542, PICTURE = 543, PLUS = 544,

POINTER = 545, POSITION = 546, POSITIVE = 547, PRESENT = 548,

PREVIOUS = 549, PRINTER = 550, PRINTING = 551, PROCEDURE = 552,

PROCEDURES = 553, PROCEED = 554, PROGRAM = 555, PROGRAM_ID = 556,

PROGRAM_NAME = 557, PROGRAM_POINTER = 558, PROMPT = 559, QUOTE = 560,

RANDOM = 561, RD = 562, READ = 563, RECORD = 564,

RECORDING = 565, RECORDS = 566, RECURSIVE = 567, REDEFINES = 568,

REEL = 569, REFERENCE = 570, RELATIVE = 571, RELEASE = 572,

REMAINDER = 573, REMOVAL = 574, RENAMES = 575, REPLACING = 576,

REPORT = 577, REPORTING = 578, REPORTS = 579, REPORT_FOOTING = 580,

REPORT_HEADING = 581, REPOSITORY = 582, REQUIRED = 583, RESERVE = 584,

RETURN = 585, RETURNING = 586, REVERSE_FUNC = 587, REVERSE_-VIDEO = 588,

REWIND = 589, REWRITE = 590, RIGHT = 591, ROLLBACK = 592,

ROUNDED = 593, RUN = 594, SAME = 595, SCREEN = 596,

SCREEN_CONTROL = 597, SCROLL = 598, SD = 599, SEARCH = 600,

SECTION = 601, SECURE = 602, SEGMENT_LIMIT = 603, SELECT = 604,

SEMI_COLON = 605, SENTENCE = 606, SEPARATE = 607, SEQUENCE = 608,

SEQUENTIAL = 609, SET = 610, SHARING = 611, SIGN = 612,

SIGNED = 613, SIGNED_INT = 614, SIGNED_LONG = 615, SIGNED_SHORT = 616,

SIZE = 617, SIZE_ERROR = 618, SORT = 619, SORT_MERGE = 620,

SOURCE = 621, SOURCE_COMPUTER = 622, SPACE = 623, SPECIAL_NAMES = 624,

STANDARD = 625, STANDARD_1 = 626, STANDARD_2 = 627, START = 628,

STATUS = 629, STOP = 630, STRING = 631, SUBSTITUTE_FUNC = 632,

SUBSTITUTE_CASE_FUNC = 633, SUBTRACT = 634, SUM = 635, SUPPRESS = 636,

SYMBOLIC = 637, SYNCHRONIZED = 638, TALLYING = 639, TAPE = 640,

TERMINATE = 641, TEST = 642, THAN = 643, THEN = 644,

THRU = 645, TIME = 646, TIMES = 647, TO = 648,

TOK_FALSE = 649, TOK_FILE = 650, TOK_INITIAL = 651, TOK_NULL = 652,

TOK_TRUE = 653, TOP = 654, TRAILING = 655, TRANSFORM = 656,

TRIM_FUNCTION = 657, TYPE = 658, UNDERLINE = 659, UNIT = 660,

UNLOCK = 661, UNSIGNED = 662, UNSIGNED_INT = 663, UNSIGNED_LONG = 664,

UNSIGNED_SHORT = 665, UNSTRING = 666, UNTIL = 667, UP = 668,

UPDATE = 669, UPON = 670, UPON_ARGUMENT_NUMBER = 671, UPON_-COMMAND_LINE = 672,

UPON_ENVIRONMENT_NAME = 673, UPON_ENVIRONMENT_VALUE = 674, UPPER_CASE_FUNC = 675, USAGE = 676,

USE = 677, USING = 678, VALUE = 679, VARYING = 680,

WAIT = 681, WHEN = 682, WHEN_COMPILED_FUNC = 683, WITH = 684,

WORD = 685, WORDS = 686, WORKING_STORAGE = 687, WRITE = 688,

YYYYDDD = 689, YYYYMMDD = 690, ZERO = 691, UNARY_SIGN = 692,

TOKEN_EOF = 0, ACCEPT = 258, ACCESS = 259, ADD = 260,

ADDRESS = 261, ADVANCING = 262, AFTER = 263, ALL = 264,

ALLOCATE = 265, ALPHABET = 266, ALPHABETIC = 267, ALPHABETIC_-LOWER = 268,

ALPHABETIC_UPPER = 269, ALPHANUMERIC = 270, ALPHANUMERIC_EDITED = 271, ALSO = 272,

ALTER = 273, ALTERNATE = 274, AND = 275, ANY = 276,

ARE = 277, AREA = 278, ARGUMENT_NUMBER = 279, ARGUMENT_VALUE = 280,

AS = 281, ASCENDING = 282, ASSIGN = 283, AT = 284,

AUTO = 285, AUTOMATIC = 286, BACKGROUND_COLOR = 287, BASED = 288,

BEFORE = 289, BELL = 290, BINARY = 291, BINARY_C_LONG = 292,

BINARY_CHAR = 293, BINARY_DOUBLE = 294, BINARY_LONG = 295, BINARY_-SHORT = 296,

BLANK = 297, BLANK_LINE = 298, BLANK_SCREEN = 299, BLINK = 300,

BLOCK = 301, BOTTOM = 302, BY = 303, BYTE_LENGTH = 304,

CALL = 305, CANCEL = 306, CH = 307, CHAINING = 308,

CHARACTER = 309, CHARACTERS = 310, CLASS = 311, CLOSE = 312,

CODE = 313, CODE_SET = 314, COLLATING = 315, COL = 316,

COLS = 317, COLUMN = 318, COLUMNS = 319, COMMA = 320,

COMMAND_LINE = 321, COMMA_DELIM = 322, COMMIT = 323, COMMON = 324,

COMP = 325, COMPUTE = 326, COMP_1 = 327, COMP_2 = 328,

COMP_3 = 329, COMP_4 = 330, COMP_5 = 331, COMP_X = 332,

CONCATENATE_FUNC = 333, CONFIGURATION = 334, CONSTANT = 335, CONTAINS = 336,

CONTENT = 337, CONTINUE = 338, CONTROL = 339, CONTROLS = 340,

CONTROL_FOOTING = 341, CONTROL_HEADING = 342, CONVERTING = 343, CORRESPONDING = 344,

COUNT = 345, CRT = 346, CURRENCY = 347, CURRENT_DATE_FUNC = 348,

CURSOR = 349, CYCLE = 350, DATA = 351, DATE = 352,

DAY = 353, DAY_OF_WEEK = 354, DE = 355, DEBUGGING = 356,

DECIMAL_POINT = 357, DECLARATIVES = 358, DEFAULT = 359, DELETE = 360,

DELIMITED = 361, DELIMITER = 362, DEPENDING = 363, DESCENDING = 364,

DETAIL = 365, DISK = 366, DISPLAY = 367, DIVIDE = 368,

DIVISION = 369, DOWN = 370, DUPLICATES = 371, DYNAMIC = 372,

EBCDIC = 373, ELSE = 374, END = 375, END_ACCEPT = 376,

END_ADD = 377, END_CALL = 378, END_COMPUTE = 379, END_DELETE = 380,

END_DISPLAY = 381, END_DIVIDE = 382, END_EVALUATE = 383, END_-
FUNCTION = 384,

END_IF = 385, END_MULTIPLY = 386, END_PERFORM = 387, END_PROGRAM
= 388,

END_READ = 389, END_RETURN = 390, END_REWRITE = 391, END_SEARCH
= 392,

END_START = 393, END_STRING = 394, END_SUBTRACT = 395, END_UNSTRING
= 396,

END_WRITE = 397, ENTRY = 398, ENVIRONMENT = 399, ENVIRONMENT_-
NAME = 400,

ENVIRONMENT_VALUE = 401, EOL = 402, EOP = 403, EOS = 404,

EQUAL = 405, EQUALS = 406, ERASE = 407, ERROR = 408,

ESCAPE = 409, EVALUATE = 410, EVENT_STATUS = 411, EXCEPTION = 412,

EXCLUSIVE = 413, EXIT = 414, EXTEND = 415, EXTERNAL = 416,

FD = 417, FILE_CONTROL = 418, FILE_ID = 419, FILLER = 420,

FINAL = 421, FIRST = 422, FOOTING = 423, FOR = 424,

FOREGROUND_COLOR = 425, FOREVER = 426, FREE = 427, FROM = 428,

FULL = 429, FUNCTION = 430, FUNCTION_ID = 431, FUNCTION_NAME = 432,

GE = 433, GENERATE = 434, GIVING = 435, GLOBAL = 436,

GO = 437, GOBACK = 438, GREATER = 439, GROUP = 440,

HEADING = 441, HIGHLIGHT = 442, HIGH_VALUE = 443, IDENTIFICATION =
444,

IF = 445, IGNORE = 446, IGNORING = 447, IN = 448,

INDEX = 449, INDEXED = 450, INDICATE = 451, INITIALIZE = 452,

INITIALIZED = 453, INITIATE = 454, INPUT = 455, INPUT_OUTPUT = 456,

INSPECT = 457, INTO = 458, INTRINSIC = 459, INVALID = 460,

INVALID_KEY = 461, IS = 462, I_O = 463, I_O_CONTROL = 464,

JUSTIFIED = 465, KEY = 466, LABEL = 467, LAST = 468,

LAST_DETAIL = 469, LE = 470, LEADING = 471, LEFT = 472,

LENGTH = 473, LESS = 474, LIMIT = 475, LIMITS = 476,

LINAGE = 477, LINAGE_COUNTER = 478, LINE = 479, LINES = 480,

LINKAGE = 481, LITERAL = 482, LOCALE = 483, LOCALE_DT_FUNC = 484,

LOCAL_STORAGE = 485, LOCK = 486, LOWER_CASE_FUNC = 487, LOW-
LIGHT = 488,

LOW_VALUE = 489, MANUAL = 490, MEMORY = 491, MERGE = 492,

MINUS = 493, MNEMONIC_NAME = 494, MODE = 495, MOVE = 496,

MULTIPLE = 497, MULTIPLY = 498, NATIONAL = 499, NATIONAL_EDITED =
500,

NATIVE = 501, NE = 502, NEGATIVE = 503, NEXT = 504,

NEXT_SENTENCE = 505, NO = 506, NOT = 507, NOT_END = 508,

NOT_EOP = 509, NOT_EXCEPTION = 510, NOT_INVALID_KEY = 511, NOT_-OVERFLOW = 512,

NOT_SIZE_ERROR = 513, NO_ADVANCING = 514, NUMBER = 515, NUM-BERS = 516,

NUMERIC = 517, NUMERIC_EDITED = 518, NUMVALC_FUNC = 519, OBJECT_-COMPUTER = 520,

OCCURS = 521, OF = 522, OFF = 523, OMITTED = 524,

ON = 525, ONLY = 526, OPEN = 527, OPTIONAL = 528,

OR = 529, ORDER = 530, ORGANIZATION = 531, OTHER = 532,

OUTPUT = 533, OVERFLOW = 534, OVERLINE = 535, PACKED_DECIMAL = 536,

PADDING = 537, PAGE = 538, PAGE_FOOTING = 539, PAGE_HEADING = 540,

PARAGRAPH = 541, PERFORM = 542, PICTURE = 543, PLUS = 544,

POINTER = 545, POSITION = 546, POSITIVE = 547, PRESENT = 548,

PREVIOUS = 549, PRINTER = 550, PRINTING = 551, PROCEDURE = 552,

PROCEDURES = 553, PROCEED = 554, PROGRAM = 555, PROGRAM_ID = 556,

PROGRAM_NAME = 557, PROGRAM_POINTER = 558, PROMPT = 559, QUOTE = 560,

RANDOM = 561, RD = 562, READ = 563, RECORD = 564,

RECORDING = 565, RECORDS = 566, RECURSIVE = 567, REDEFINES = 568,

REEL = 569, REFERENCE = 570, RELATIVE = 571, RELEASE = 572,

REMAINDER = 573, REMOVAL = 574, RENAMES = 575, REPLACING = 576,

REPORT = 577, REPORTING = 578, REPORTS = 579, REPORT_FOOTING = 580,

REPORT_HEADING = 581, REPOSITORY = 582, REQUIRED = 583, RESERVE = 584,

RETURN = 585, RETURNING = 586, REVERSE_FUNC = 587, REVERSE_-VIDEO = 588,

REWIND = 589, REWRITE = 590, RIGHT = 591, ROLLBACK = 592,

ROUNDED = 593, RUN = 594, SAME = 595, SCREEN = 596,

SCREEN_CONTROL = 597, SCROLL = 598, SD = 599, SEARCH = 600,

SECTION = 601, SECURE = 602, SEGMENT_LIMIT = 603, SELECT = 604,

SEMI_COLON = 605, SENTENCE = 606, SEPARATE = 607, SEQUENCE = 608,

SEQUENTIAL = 609, SET = 610, SHARING = 611, SIGN = 612,

SIGNED = 613, SIGNED_INT = 614, SIGNED_LONG = 615, SIGNED_SHORT = 616,

SIZE = 617, SIZE_ERROR = 618, SORT = 619, SORT_MERGE = 620,

SOURCE = 621, SOURCE_COMPUTER = 622, SPACE = 623, SPECIAL_NAMES = 624,

STANDARD = 625, STANDARD_1 = 626, STANDARD_2 = 627, START = 628,

STATUS = 629, STOP = 630, STRING = 631, SUBSTITUTE_FUNC = 632,

SUBSTITUTE_CASE_FUNC = 633, SUBTRACT = 634, SUM = 635, SUPPRESS = 636,

SYMBOLIC = 637, SYNCHRONIZED = 638, TALLYING = 639, TAPE = 640,

TERMINATE = 641, TEST = 642, THAN = 643, THEN = 644,

THRU = 645, TIME = 646, TIMES = 647, TO = 648,

TOK_FALSE = 649, TOK_FILE = 650, TOK_INITIAL = 651, TOK_NULL = 652,

TOK_TRUE = 653, TOP = 654, TRAILING = 655, TRANSFORM = 656,

TRIM_FUNCTION = 657, TYPE = 658, UNDERLINE = 659, UNIT = 660,

UNLOCK = 661, UNSIGNED = 662, UNSIGNED_INT = 663, UNSIGNED_LONG = 664,

UNSIGNED_SHORT = 665, UNSTRING = 666, UNTIL = 667, UP = 668,

UPDATE = 669, UPON = 670, UPON_ARGUMENT_NUMBER = 671, UPON_-COMMAND_LINE = 672,

UPON_ENVIRONMENT_NAME = 673, UPON_ENVIRONMENT_VALUE = 674, UPPER_CASE_FUNC = 675, USAGE = 676,

USE = 677, USING = 678, VALUE = 679, VARYING = 680,

WAIT = 681, WHEN = 682, WHEN_COMPILED_FUNC = 683, WITH = 684,

WORD = 685, WORDS = 686, WORKING_STORAGE = 687, WRITE = 688,

YYYYDDD = 689, YYYYMMDD = 690, ZERO = 691, UNARY_SIGN = 692,

TOKEN_EOF = 0, COPY = 258, REPLACE = 259, SUPPRESS = 260,

PRINTING = 261, REPLACING = 262, OFF = 263, IN = 264,

OF = 265, BY = 266, EQEQ = 267, TOKEN = 268,

TOKEN_EOF = 0, COPY = 258, REPLACE = 259, SUPPRESS = 260,

PRINTING = 261, REPLACING = 262, OFF = 263, IN = 264,

OF = 265, BY = 266, EQEQ = 267, TOKEN = 268 }

Functions

- int yyparse ()

Variables

- struct cb_program * current_program = NULL
- struct cb_statement * current_statement = NULL
- struct cb_label * current_section = NULL
- struct cb_label * current_paragraph = NULL
- size_t functions_are_all = 0
- int non_const_word = 0

- int yydebug
- int yychar
- YYSTYPE yylval
- int yynerrs

7.8.1 Define Documentation

7.8.1.1 #define ACCEPT 258

Definition at line 495 of file parser.c.

7.8.1.2 #define ACCESS 259

Definition at line 496 of file parser.c.

7.8.1.3 #define ADD 260

Definition at line 497 of file parser.c.

7.8.1.4 #define ADDRESS 261

Definition at line 498 of file parser.c.

7.8.1.5 #define ADVANCING 262

Definition at line 499 of file parser.c.

7.8.1.6 #define AFTER 263

Definition at line 500 of file parser.c.

7.8.1.7 #define ALL 264

Definition at line 501 of file parser.c.

7.8.1.8 #define ALLOCATE 265

Definition at line 502 of file parser.c.

7.8.1.9 #define ALPHABET 266

Definition at line 503 of file parser.c.

7.8.1.10 #define ALPHABETIC 267

Definition at line 504 of file parser.c.

7.8.1.11 #define ALPHABETIC_LOWER 268

Definition at line 505 of file parser.c.

7.8.1.12 #define ALPHABETIC_UPPER 269

Definition at line 506 of file parser.c.

7.8.1.13 #define ALPHANUMERIC 270

Definition at line 507 of file parser.c.

7.8.1.14 #define ALPHANUMERIC_EDITED 271

Definition at line 508 of file parser.c.

7.8.1.15 #define ALSO 272

Definition at line 509 of file parser.c.

7.8.1.16 #define ALTER 273

Definition at line 510 of file parser.c.

7.8.1.17 #define ALTERNATE 274

Definition at line 511 of file parser.c.

7.8.1.18 #define AND 275

Definition at line 512 of file parser.c.

7.8.1.19 #define ANY 276

Definition at line 513 of file parser.c.

7.8.1.20 #define ARE 277

Definition at line 514 of file parser.c.

7.8.1.21 #define AREA 278

Definition at line 515 of file parser.c.

7.8.1.22 #define ARGUMENT_NUMBER 279

Definition at line 516 of file parser.c.

7.8.1.23 #define ARGUMENT_VALUE 280

Definition at line 517 of file parser.c.

7.8.1.24 #define AS 281

Definition at line 518 of file parser.c.

7.8.1.25 #define ASCENDING 282

Definition at line 519 of file parser.c.

7.8.1.26 #define ASSIGN 283

Definition at line 520 of file parser.c.

7.8.1.27 #define AT 284

Definition at line 521 of file parser.c.

7.8.1.28 #define AUTO 285

Definition at line 522 of file parser.c.

7.8.1.29 #define AUTOMATIC 286

Definition at line 523 of file parser.c.

7.8.1.30 #define BACKGROUND_COLOR 287

Definition at line 524 of file parser.c.

7.8.1.31 #define BASED 288

Definition at line 525 of file parser.c.

7.8.1.32 #define BEFORE 289

Definition at line 526 of file parser.c.

7.8.1.33 #define BELL 290

Definition at line 527 of file parser.c.

7.8.1.34 #define BINARY 291

Definition at line 528 of file parser.c.

7.8.1.35 #define BINARY_C_LONG 292

Definition at line 529 of file parser.c.

7.8.1.36 #define BINARY_CHAR 293

Definition at line 530 of file parser.c.

7.8.1.37 #define BINARY_DOUBLE 294

Definition at line 531 of file parser.c.

7.8.1.38 #define BINARY_LONG 295

Definition at line 532 of file parser.c.

7.8.1.39 #define BINARY_SHORT 296

Definition at line 533 of file parser.c.

7.8.1.40 #define BLANK 297

Definition at line 534 of file parser.c.

7.8.1.41 #define BLANK_LINE 298

Definition at line 535 of file parser.c.

7.8.1.42 #define BLANK_SCREEN 299

Definition at line 536 of file parser.c.

7.8.1.43 #define BLINK 300

Definition at line 537 of file parser.c.

7.8.1.44 #define BLOCK 301

Definition at line 538 of file parser.c.

7.8.1.45 #define BOTTOM 302

Definition at line 539 of file parser.c.

7.8.1.46 #define BY 303

Definition at line 540 of file parser.c.

7.8.1.47 #define BYTE_LENGTH 304

Definition at line 541 of file parser.c.

7.8.1.48 #define CALL 305

Definition at line 542 of file parser.c.

7.8.1.49 #define CANCEL 306

Definition at line 543 of file parser.c.

7.8.1.50 #define CH 307

Definition at line 544 of file parser.c.

7.8.1.51 #define CHAINING 308

Definition at line 545 of file parser.c.

7.8.1.52 #define CHARACTER 309

Definition at line 546 of file parser.c.

7.8.1.53 #define CHARACTERS 310

Definition at line 547 of file parser.c.

7.8.1.54 #define CLASS 311

Definition at line 548 of file parser.c.

7.8.1.55 #define CLOSE 312

Definition at line 549 of file parser.c.

7.8.1.56 #define CODE 313

Definition at line 550 of file parser.c.

7.8.1.57 #define CODE_SET 314

Definition at line 551 of file parser.c.

7.8.1.58 #define COL 316

Definition at line 553 of file parser.c.

7.8.1.59 #define COLLATING 315

Definition at line 552 of file parser.c.

7.8.1.60 #define COLS 317

Definition at line 554 of file parser.c.

7.8.1.61 #define COLUMN 318

Definition at line 555 of file parser.c.

7.8.1.62 #define COLUMNS 319

Definition at line 556 of file parser.c.

7.8.1.63 #define COMMA 320

Definition at line 557 of file parser.c.

7.8.1.64 #define COMMA_DELIM 322

Definition at line 559 of file parser.c.

7.8.1.65 #define COMMAND_LINE 321

Definition at line 558 of file parser.c.

7.8.1.66 #define COMMIT 323

Definition at line 560 of file parser.c.

7.8.1.67 #define COMMON 324

Definition at line 561 of file parser.c.

7.8.1.68 #define COMP 325

Definition at line 562 of file parser.c.

7.8.1.69 #define COMP_1 327

Definition at line 564 of file parser.c.

7.8.1.70 #define COMP_2 328

Definition at line 565 of file parser.c.

7.8.1.71 #define COMP_3 329

Definition at line 566 of file parser.c.

7.8.1.72 #define COMP_4 330

Definition at line 567 of file parser.c.

7.8.1.73 #define COMP_5 331

Definition at line 568 of file parser.c.

7.8.1.74 #define COMP_X 332

Definition at line 569 of file parser.c.

7.8.1.75 #define COMPUTE 326

Definition at line 563 of file parser.c.

7.8.1.76 #define CONCATENATE_FUNC 333

Definition at line 570 of file parser.c.

7.8.1.77 #define CONFIGURATION 334

Definition at line 571 of file parser.c.

7.8.1.78 #define CONSTANT 335

Definition at line 572 of file parser.c.

7.8.1.79 #define CONTAINS 336

Definition at line 573 of file parser.c.

7.8.1.80 #define CONTENT 337

Definition at line 574 of file parser.c.

7.8.1.81 #define CONTINUE 338

Definition at line 575 of file parser.c.

7.8.1.82 #define CONTROL 339

Definition at line 576 of file parser.c.

7.8.1.83 #define CONTROL_FOOTING 341

Definition at line 578 of file parser.c.

7.8.1.84 #define CONTROL_HEADING 342

Definition at line 579 of file parser.c.

7.8.1.85 #define CONTROLS 340

Definition at line 577 of file parser.c.

7.8.1.86 #define CONVERTING 343

Definition at line 580 of file parser.c.

7.8.1.87 #define CORRESPONDING 344

Definition at line 581 of file parser.c.

7.8.1.88 #define COUNT 345

Definition at line 582 of file parser.c.

7.8.1.89 #define CRT 346

Definition at line 583 of file parser.c.

7.8.1.90 #define CURRENCY 347

Definition at line 584 of file parser.c.

7.8.1.91 #define CURRENT_DATE_FUNC 348

Definition at line 585 of file parser.c.

7.8.1.92 #define CURSOR 349

Definition at line 586 of file parser.c.

7.8.1.93 #define CYCLE 350

Definition at line 587 of file parser.c.

7.8.1.94 #define DATA 351

Definition at line 588 of file parser.c.

7.8.1.95 #define DATE 352

Definition at line 589 of file parser.c.

7.8.1.96 #define DAY 353

Definition at line 590 of file parser.c.

7.8.1.97 #define DAY_OF_WEEK 354

Definition at line 591 of file parser.c.

7.8.1.98 #define DE 355

Definition at line 592 of file parser.c.

7.8.1.99 #define DEBUGGING 356

Definition at line 593 of file parser.c.

7.8.1.100 #define DECIMAL_POINT 357

Definition at line 594 of file parser.c.

7.8.1.101 #define DECLARATIVES 358

Definition at line 595 of file parser.c.

7.8.1.102 #define DEFAULT 359

Definition at line 596 of file parser.c.

7.8.1.103 #define DELETE 360

Definition at line 597 of file parser.c.

7.8.1.104 #define DELIMITED 361

Definition at line 598 of file parser.c.

7.8.1.105 #define DELIMITER 362

Definition at line 599 of file parser.c.

7.8.1.106 #define DEPENDING 363

Definition at line 600 of file parser.c.

7.8.1.107 #define DESCENDING 364

Definition at line 601 of file parser.c.

7.8.1.108 #define DETAIL 365

Definition at line 602 of file parser.c.

7.8.1.109 #define DISK 366

Definition at line 603 of file parser.c.

7.8.1.110 #define DISPLAY 367

Definition at line 604 of file parser.c.

7.8.1.111 #define DIVIDE 368

Definition at line 605 of file parser.c.

7.8.1.112 #define DIVISION 369

Definition at line 606 of file parser.c.

7.8.1.113 #define DOWN 370

Definition at line 607 of file parser.c.

7.8.1.114 #define DUPLICATES 371

Definition at line 608 of file parser.c.

7.8.1.115 #define DYNAMIC 372

Definition at line 609 of file parser.c.

7.8.1.116 #define EBCDIC 373

Definition at line 610 of file parser.c.

7.8.1.117 #define ELSE 374

Definition at line 611 of file parser.c.

7.8.1.118 #define emit_statement(*x*) current_program->exec_list = cb_cons (x, current_program->exec_list)

Definition at line 951 of file parser.c.

7.8.1.119 #define END 375

Definition at line 612 of file parser.c.

7.8.1.120 #define END_ACCEPT 376

Definition at line 613 of file parser.c.

7.8.1.121 #define END_ADD 377

Definition at line 614 of file parser.c.

7.8.1.122 #define END_CALL 378

Definition at line 615 of file parser.c.

7.8.1.123 #define END_COMPUTE 379

Definition at line 616 of file parser.c.

7.8.1.124 #define END_DELETE 380

Definition at line 617 of file parser.c.

7.8.1.125 #define END_DISPLAY 381

Definition at line 618 of file parser.c.

7.8.1.126 #define END_DIVIDE 382

Definition at line 619 of file parser.c.

7.8.1.127 #define END_EVALUATE 383

Definition at line 620 of file parser.c.

7.8.1.128 #define END_FUNCTION 384

Definition at line 621 of file parser.c.

7.8.1.129 #define END_IF 385

Definition at line 622 of file parser.c.

7.8.1.130 #define END_MULTIPLY 386

Definition at line 623 of file parser.c.

7.8.1.131 #define END_PERFORM 387

Definition at line 624 of file parser.c.

7.8.1.132 #define END_PROGRAM 388

Definition at line 625 of file parser.c.

7.8.1.133 #define END_READ 389

Definition at line 626 of file parser.c.

7.8.1.134 #define END_RETURN 390

Definition at line 627 of file parser.c.

7.8.1.135 #define END_REWRITE 391

Definition at line 628 of file parser.c.

7.8.1.136 #define END_SEARCH 392

Definition at line 629 of file parser.c.

7.8.1.137 #define END_START 393

Definition at line 630 of file parser.c.

7.8.1.138 #define END_STRING 394

Definition at line 631 of file parser.c.

7.8.1.139 #define END_SUBTRACT 395

Definition at line 632 of file parser.c.

7.8.1.140 #define END_UNSTRING 396

Definition at line 633 of file parser.c.

7.8.1.141 #define END_WRITE 397

Definition at line 634 of file parser.c.

7.8.1.142 #define ENTRY 398

Definition at line 635 of file parser.c.

7.8.1.143 #define ENVIRONMENT 399

Definition at line 636 of file parser.c.

7.8.1.144 #define ENVIRONMENT_NAME 400

Definition at line 637 of file parser.c.

7.8.1.145 #define ENVIRONMENT_VALUE 401

Definition at line 638 of file parser.c.

7.8.1.146 #define EOL 402

Definition at line 639 of file parser.c.

7.8.1.147 #define EOP 403

Definition at line 640 of file parser.c.

7.8.1.148 #define EOS 404

Definition at line 641 of file parser.c.

7.8.1.149 #define EQUAL 405

Definition at line 642 of file parser.c.

7.8.1.150 #define EQUALS 406

Definition at line 643 of file parser.c.

7.8.1.151 #define ERASE 407

Definition at line 644 of file parser.c.

7.8.1.152 #define ERROR 408

Definition at line 645 of file parser.c.

7.8.1.153 #define ESCAPE 409

Definition at line 646 of file parser.c.

7.8.1.154 #define EVALUATE 410

Definition at line 647 of file parser.c.

7.8.1.155 #define EVENT_STATUS 411

Definition at line 648 of file parser.c.

7.8.1.156 #define EXCEPTION 412

Definition at line 649 of file parser.c.

7.8.1.157 #define EXCLUSIVE 413

Definition at line 650 of file parser.c.

7.8.1.158 #define EXIT 414

Definition at line 651 of file parser.c.

7.8.1.159 #define EXTEND 415

Definition at line 652 of file parser.c.

7.8.1.160 #define EXTERNAL 416

Definition at line 653 of file parser.c.

7.8.1.161 #define FD 417

Definition at line 654 of file parser.c.

7.8.1.162 #define FILE_CONTROL 418

Definition at line 655 of file parser.c.

7.8.1.163 #define FILE_ID 419

Definition at line 656 of file parser.c.

7.8.1.164 #define FILLER 420

Definition at line 657 of file parser.c.

7.8.1.165 #define FINAL 421

Definition at line 658 of file parser.c.

7.8.1.166 #define FIRST 422

Definition at line 659 of file parser.c.

7.8.1.167 #define FOOTING 423

Definition at line 660 of file parser.c.

7.8.1.168 #define FOR 424

Definition at line 661 of file parser.c.

7.8.1.169 #define FOREGROUND_COLOR 425

Definition at line 662 of file parser.c.

7.8.1.170 #define FOREVER 426

Definition at line 663 of file parser.c.

7.8.1.171 #define FREE 427

Definition at line 664 of file parser.c.

7.8.1.172 #define FROM 428

Definition at line 665 of file parser.c.

7.8.1.173 #define FULL 429

Definition at line 666 of file parser.c.

7.8.1.174 #define FUNCTION 430

Definition at line 667 of file parser.c.

7.8.1.175 #define FUNCTION_ID 431

Definition at line 668 of file parser.c.

7.8.1.176 #define FUNCTION_NAME 432

Definition at line 669 of file parser.c.

7.8.1.177 #define GE 433

Definition at line 670 of file parser.c.

7.8.1.178 #define GENERATE 434

Definition at line 671 of file parser.c.

7.8.1.179 #define GIVING 435

Definition at line 672 of file parser.c.

7.8.1.180 #define GLOBAL 436

Definition at line 673 of file parser.c.

7.8.1.181 #define GO 437

Definition at line 674 of file parser.c.

7.8.1.182 #define GOBACK 438

Definition at line 675 of file parser.c.

7.8.1.183 #define GREATER 439

Definition at line 676 of file parser.c.

7.8.1.184 #define GROUP 440

Definition at line 677 of file parser.c.

7.8.1.185 #define HEADING 441

Definition at line 678 of file parser.c.

7.8.1.186 #define HIGH_VALUE 443

Definition at line 680 of file parser.c.

7.8.1.187 #define HIGHLIGHT 442

Definition at line 679 of file parser.c.

7.8.1.188 #define I_O 463

Definition at line 700 of file parser.c.

7.8.1.189 #define I_O_CONTROL 464

Definition at line 701 of file parser.c.

7.8.1.190 #define IDENTIFICATION 444

Definition at line 681 of file parser.c.

7.8.1.191 #define IF 445

Definition at line 682 of file parser.c.

7.8.1.192 #define IGNORE 446

Definition at line 683 of file parser.c.

7.8.1.193 #define IGNORING 447

Definition at line 684 of file parser.c.

7.8.1.194 #define IN 448

Definition at line 685 of file parser.c.

7.8.1.195 #define INDEX 449

Definition at line 686 of file parser.c.

7.8.1.196 #define INDEXED 450

Definition at line 687 of file parser.c.

7.8.1.197 #define INDICATE 451

Definition at line 688 of file parser.c.

7.8.1.198 #define INITIALIZE 452

Definition at line 689 of file parser.c.

7.8.1.199 #define INITIALIZED 453

Definition at line 690 of file parser.c.

7.8.1.200 #define INITIATE 454

Definition at line 691 of file parser.c.

7.8.1.201 #define INPUT 455

Definition at line 692 of file parser.c.

7.8.1.202 #define INPUT_OUTPUT 456

Definition at line 693 of file parser.c.

7.8.1.203 #define INSPECT 457

Definition at line 694 of file parser.c.

7.8.1.204 #define INTO 458

Definition at line 695 of file parser.c.

7.8.1.205 #define INTRINSIC 459

Definition at line 696 of file parser.c.

7.8.1.206 #define INVALID 460

Definition at line 697 of file parser.c.

7.8.1.207 #define INVALID_KEY 461

Definition at line 698 of file parser.c.

7.8.1.208 #define IS 462

Definition at line 699 of file parser.c.

7.8.1.209 #define JUSTIFIED 465

Definition at line 702 of file parser.c.

7.8.1.210 #define KEY 466

Definition at line 703 of file parser.c.

7.8.1.211 #define LABEL 467

Definition at line 704 of file parser.c.

7.8.1.212 #define LAST 468

Definition at line 705 of file parser.c.

7.8.1.213 #define LAST_DETAIL 469

Definition at line 706 of file parser.c.

7.8.1.214 #define LE 470

Definition at line 707 of file parser.c.

7.8.1.215 #define LEADING 471

Definition at line 708 of file parser.c.

7.8.1.216 #define LEFT 472

Definition at line 709 of file parser.c.

7.8.1.217 #define LENGTH 473

Definition at line 710 of file parser.c.

7.8.1.218 #define LESS 474

Definition at line 711 of file parser.c.

7.8.1.219 #define LIMIT 475

Definition at line 712 of file parser.c.

7.8.1.220 #define LIMITS 476

Definition at line 713 of file parser.c.

7.8.1.221 #define LINAGE 477

Definition at line 714 of file parser.c.

7.8.1.222 #define LINAGE_COUNTER 478

Definition at line 715 of file parser.c.

7.8.1.223 #define LINE 479

Definition at line 716 of file parser.c.

7.8.1.224 #define LINES 480

Definition at line 717 of file parser.c.

7.8.1.225 #define LINKAGE 481

Definition at line 718 of file parser.c.

7.8.1.226 #define LITERAL 482

Definition at line 719 of file parser.c.

7.8.1.227 #define LOCAL_STORAGE 485

Definition at line 722 of file parser.c.

7.8.1.228 #define LOCALE 483

Definition at line 720 of file parser.c.

7.8.1.229 #define LOCALE_DT_FUNC 484

Definition at line 721 of file parser.c.

7.8.1.230 #define LOCK 486

Definition at line 723 of file parser.c.

7.8.1.231 #define LOW_VALUE 489

Definition at line 726 of file parser.c.

7.8.1.232 #define LOWER_CASE_FUNC 487

Definition at line 724 of file parser.c.

7.8.1.233 #define LOWLIGHT 488

Definition at line 725 of file parser.c.

7.8.1.234 #define MANUAL 490

Definition at line 727 of file parser.c.

7.8.1.235 #define MEMORY 491

Definition at line 728 of file parser.c.

7.8.1.236 #define MERGE 492

Definition at line 729 of file parser.c.

7.8.1.237 #define MINUS 493

Definition at line 730 of file parser.c.

7.8.1.238 #define MNEMONIC_NAME 494

Definition at line 731 of file parser.c.

7.8.1.239 #define MODE 495

Definition at line 732 of file parser.c.

7.8.1.240 #define MOVE 496

Definition at line 733 of file parser.c.

7.8.1.241 #define MULTIPLE 497

Definition at line 734 of file parser.c.

7.8.1.242 #define MULTIPLY 498

Definition at line 735 of file parser.c.

7.8.1.243 #define NATIONAL 499

Definition at line 736 of file parser.c.

7.8.1.244 #define NATIONAL_EDITED 500

Definition at line 737 of file parser.c.

7.8.1.245 #define NATIVE 501

Definition at line 738 of file parser.c.

7.8.1.246 #define NE 502

Definition at line 739 of file parser.c.

7.8.1.247 #define NEGATIVE 503

Definition at line 740 of file parser.c.

7.8.1.248 #define NEXT 504

Definition at line 741 of file parser.c.

7.8.1.249 #define NEXT_SENTENCE 505

Definition at line 742 of file parser.c.

7.8.1.250 #define NO 506

Definition at line 743 of file parser.c.

7.8.1.251 #define NO_ADVANCING 514

Definition at line 751 of file parser.c.

7.8.1.252 #define NOT 507

Definition at line 744 of file parser.c.

7.8.1.253 #define NOT_END 508

Definition at line 745 of file parser.c.

7.8.1.254 #define NOT_EOP 509

Definition at line 746 of file parser.c.

7.8.1.255 #define NOT_EXCEPTION 510

Definition at line 747 of file parser.c.

7.8.1.256 #define NOT_INVALID_KEY 511

Definition at line 748 of file parser.c.

7.8.1.257 #define NOT_OVERFLOW 512

Definition at line 749 of file parser.c.

7.8.1.258 #define NOT_SIZE_ERROR 513

Definition at line 750 of file parser.c.

7.8.1.259 #define NUMBER 515

Definition at line 752 of file parser.c.

7.8.1.260 #define NUMBERS 516

Definition at line 753 of file parser.c.

7.8.1.261 #define NUMERIC 517

Definition at line 754 of file parser.c.

7.8.1.262 #define NUMERIC_EDITED 518

Definition at line 755 of file parser.c.

7.8.1.263 #define NUMVALC_FUNC 519

Definition at line 756 of file parser.c.

7.8.1.264 #define OBJECT_COMPUTER 520

Definition at line 757 of file parser.c.

7.8.1.265 #define OCCURS 521

Definition at line 758 of file parser.c.

7.8.1.266 #define OF 522

Definition at line 759 of file parser.c.

7.8.1.267 #define OFF 523

Definition at line 760 of file parser.c.

7.8.1.268 #define OMITTED 524

Definition at line 761 of file parser.c.

7.8.1.269 #define ON 525

Definition at line 762 of file parser.c.

7.8.1.270 #define ONLY 526

Definition at line 763 of file parser.c.

7.8.1.271 #define OPEN 527

Definition at line 764 of file parser.c.

7.8.1.272 #define OPTIONAL 528

Definition at line 765 of file parser.c.

7.8.1.273 #define OR 529

Definition at line 766 of file parser.c.

7.8.1.274 #define ORDER 530

Definition at line 767 of file parser.c.

7.8.1.275 #define ORGANIZATION 531

Definition at line 768 of file parser.c.

7.8.1.276 #define OTHER 532

Definition at line 769 of file parser.c.

7.8.1.277 #define OUTPUT 533

Definition at line 770 of file parser.c.

7.8.1.278 #define OVERFLOW 534

Definition at line 771 of file parser.c.

7.8.1.279 #define OVERLINE 535

Definition at line 772 of file parser.c.

7.8.1.280 #define PACKED_DECIMAL 536

Definition at line 773 of file parser.c.

7.8.1.281 #define PADDING 537

Definition at line 774 of file parser.c.

7.8.1.282 #define PAGE 538

Definition at line 775 of file parser.c.

7.8.1.283 #define PAGE_FOOTING 539

Definition at line 776 of file parser.c.

7.8.1.284 #define PAGE_HEADING 540

Definition at line 777 of file parser.c.

7.8.1.285 #define PARAGRAPH 541

Definition at line 778 of file parser.c.

7.8.1.286 #define PENDING(*x*) cb_warning (_("'%s' not implemented"), x)

Definition at line 949 of file parser.c.

7.8.1.287 #define PERFORM 542

Definition at line 779 of file parser.c.

7.8.1.288 #define PICTURE 543

Definition at line 780 of file parser.c.

7.8.1.289 #define PLUS 544

Definition at line 781 of file parser.c.

7.8.1.290 #define POINTER 545

Definition at line 782 of file parser.c.

7.8.1.291 #define POSITION 546

Definition at line 783 of file parser.c.

7.8.1.292 #define POSITIVE 547

Definition at line 784 of file parser.c.

7.8.1.293 #define PRESENT 548

Definition at line 785 of file parser.c.

7.8.1.294 #define PREVIOUS 549

Definition at line 786 of file parser.c.

7.8.1.295 #define PRINTER 550

Definition at line 787 of file parser.c.

7.8.1.296 #define PRINTING 551

Definition at line 788 of file parser.c.

7.8.1.297 #define PROCEDURE 552

Definition at line 789 of file parser.c.

7.8.1.298 #define PROCEDURES 553

Definition at line 790 of file parser.c.

7.8.1.299 #define PROCEED 554

Definition at line 791 of file parser.c.

7.8.1.300 #define PROGRAM 555

Definition at line 792 of file parser.c.

7.8.1.301 #define PROGRAM_ID 556

Definition at line 793 of file parser.c.

7.8.1.302 #define PROGRAM_NAME 557

Definition at line 794 of file parser.c.

7.8.1.303 #define PROGRAM_POINTER 558

Definition at line 795 of file parser.c.

7.8.1.304 #define PROMPT 559

Definition at line 796 of file parser.c.

7.8.1.305 #define push_expr(type, node) current_expr = cb_build_list (cb_int (type), node, current_expr)

Definition at line 954 of file parser.c.

7.8.1.306 #define QUOTE 560

Definition at line 797 of file parser.c.

7.8.1.307 #define RANDOM 561

Definition at line 798 of file parser.c.

7.8.1.308 #define RD 562

Definition at line 799 of file parser.c.

7.8.1.309 #define READ 563

Definition at line 800 of file parser.c.

7.8.1.310 #define RECORD 564

Definition at line 801 of file parser.c.

7.8.1.311 #define RECORDING 565

Definition at line 802 of file parser.c.

7.8.1.312 #define RECORDS 566

Definition at line 803 of file parser.c.

7.8.1.313 #define RECURSIVE 567

Definition at line 804 of file parser.c.

7.8.1.314 #define REDEFINES 568

Definition at line 805 of file parser.c.

7.8.1.315 #define REEL 569

Definition at line 806 of file parser.c.

7.8.1.316 #define REFERENCE 570

Definition at line 807 of file parser.c.

7.8.1.317 #define RELATIVE 571

Definition at line 808 of file parser.c.

7.8.1.318 #define RELEASE 572

Definition at line 809 of file parser.c.

7.8.1.319 #define REMAINDER 573

Definition at line 810 of file parser.c.

7.8.1.320 #define REMOVAL 574

Definition at line 811 of file parser.c.

7.8.1.321 #define RENAMES 575

Definition at line 812 of file parser.c.

7.8.1.322 #define REPLACING 576

Definition at line 813 of file parser.c.

7.8.1.323 #define REPORT 577

Definition at line 814 of file parser.c.

7.8.1.324 #define REPORT_FOOTING 580

Definition at line 817 of file parser.c.

7.8.1.325 #define REPORT_HEADING 581

Definition at line 818 of file parser.c.

7.8.1.326 #define REPORTING 578

Definition at line 815 of file parser.c.

7.8.1.327 #define REPORTS 579

Definition at line 816 of file parser.c.

7.8.1.328 #define REPOSITORY 582

Definition at line 819 of file parser.c.

7.8.1.329 #define REQUIRED 583

Definition at line 820 of file parser.c.

7.8.1.330 #define RESERVE 584

Definition at line 821 of file parser.c.

7.8.1.331 #define RETURN 585

Definition at line 822 of file parser.c.

7.8.1.332 #define RETURNING 586

Definition at line 823 of file parser.c.

7.8.1.333 #define REVERSE_FUNC 587

Definition at line 824 of file parser.c.

7.8.1.334 #define REVERSE_VIDEO 588

Definition at line 825 of file parser.c.

7.8.1.335 #define REWIND 589

Definition at line 826 of file parser.c.

7.8.1.336 #define REWRITE 590

Definition at line 827 of file parser.c.

7.8.1.337 #define RIGHT 591

Definition at line 828 of file parser.c.

7.8.1.338 #define ROLLBACK 592

Definition at line 829 of file parser.c.

7.8.1.339 #define ROUNDED 593

Definition at line 830 of file parser.c.

7.8.1.340 #define RUN 594

Definition at line 831 of file parser.c.

7.8.1.341 #define SAME 595

Definition at line 832 of file parser.c.

7.8.1.342 #define SCREEN 596

Definition at line 833 of file parser.c.

7.8.1.343 #define SCREEN_CONTROL 597

Definition at line 834 of file parser.c.

7.8.1.344 #define SCROLL 598

Definition at line 835 of file parser.c.

7.8.1.345 #define SD 599

Definition at line 836 of file parser.c.

7.8.1.346 #define SEARCH 600

Definition at line 837 of file parser.c.

7.8.1.347 #define SECTION 601

Definition at line 838 of file parser.c.

7.8.1.348 #define SECURE 602

Definition at line 839 of file parser.c.

7.8.1.349 #define SEGMENT_LIMIT 603

Definition at line 840 of file parser.c.

7.8.1.350 #define SELECT 604

Definition at line 841 of file parser.c.

7.8.1.351 #define SEMI_COLON 605

Definition at line 842 of file parser.c.

7.8.1.352 #define SENTENCE 606

Definition at line 843 of file parser.c.

7.8.1.353 #define SEPARATE 607

Definition at line 844 of file parser.c.

7.8.1.354 #define SEQUENCE 608

Definition at line 845 of file parser.c.

7.8.1.355 #define SEQUENTIAL 609

Definition at line 846 of file parser.c.

7.8.1.356 #define SET 610

Definition at line 847 of file parser.c.

7.8.1.357 #define SHARING 611

Definition at line 848 of file parser.c.

7.8.1.358 #define SIGN 612

Definition at line 849 of file parser.c.

7.8.1.359 #define SIGNED 613

Definition at line 850 of file parser.c.

7.8.1.360 #define SIGNED_INT 614

Definition at line 851 of file parser.c.

7.8.1.361 #define SIGNED_LONG 615

Definition at line 852 of file parser.c.

7.8.1.362 #define SIGNED_SHORT 616

Definition at line 853 of file parser.c.

7.8.1.363 #define SIZE 617

Definition at line 854 of file parser.c.

7.8.1.364 #define SIZE_ERROR 618

Definition at line 855 of file parser.c.

7.8.1.365 #define SORT 619

Definition at line 856 of file parser.c.

7.8.1.366 #define SORT_MERGE 620

Definition at line 857 of file parser.c.

7.8.1.367 #define SOURCE 621

Definition at line 858 of file parser.c.

7.8.1.368 #define SOURCE_COMPUTER 622

Definition at line 859 of file parser.c.

7.8.1.369 #define SPACE 623

Definition at line 860 of file parser.c.

7.8.1.370 #define SPECIAL_NAMES 624

Definition at line 861 of file parser.c.

7.8.1.371 #define STANDARD 625

Definition at line 862 of file parser.c.

7.8.1.372 #define STANDARD_1 626

Definition at line 863 of file parser.c.

7.8.1.373 #define STANDARD_2 627

Definition at line 864 of file parser.c.

7.8.1.374 #define START 628

Definition at line 865 of file parser.c.

7.8.1.375 #define STATUS 629

Definition at line 866 of file parser.c.

7.8.1.376 #define STOP 630

Definition at line 867 of file parser.c.

7.8.1.377 #define STRING 631

Definition at line 868 of file parser.c.

7.8.1.378 #define SUBSTITUTE_CASE_FUNC 633

Definition at line 870 of file parser.c.

7.8.1.379 #define SUBSTITUTE_FUNC 632

Definition at line 869 of file parser.c.

7.8.1.380 #define SUBTRACT 634

Definition at line 871 of file parser.c.

7.8.1.381 #define SUM 635

Definition at line 872 of file parser.c.

7.8.1.382 #define SUPPRESS 636

Definition at line 873 of file parser.c.

7.8.1.383 #define SYMBOLIC 637

Definition at line 874 of file parser.c.

7.8.1.384 #define SYNCHRONIZED 638

Definition at line 875 of file parser.c.

7.8.1.385 #define TALLYING 639

Definition at line 876 of file parser.c.

7.8.1.386 #define TAPE 640

Definition at line 877 of file parser.c.

7.8.1.387 #define TERM_ACCEPT 1

Definition at line 958 of file parser.c.

7.8.1.388 #define TERM_ADD 2

Definition at line 959 of file parser.c.

7.8.1.389 #define TERM_CALL 3

Definition at line 960 of file parser.c.

7.8.1.390 #define TERM_COMPUTE 4

Definition at line 961 of file parser.c.

7.8.1.391 #define TERM_DELETE 5

Definition at line 962 of file parser.c.

7.8.1.392 #define TERM_DISPLAY 6

Definition at line 963 of file parser.c.

7.8.1.393 #define TERM_DIVIDE 7

Definition at line 964 of file parser.c.

7.8.1.394 #define TERM_EVALUATE 8

Definition at line 965 of file parser.c.

7.8.1.395 #define TERM_IF 9

Definition at line 966 of file parser.c.

7.8.1.396 #define TERM_MAX 22

Definition at line 979 of file parser.c.

7.8.1.397 #define TERM_MULTIPLY 10

Definition at line 967 of file parser.c.

7.8.1.398 #define TERM_NONE 0

Definition at line 957 of file parser.c.

7.8.1.399 #define TERM_PERFORM 11

Definition at line 968 of file parser.c.

7.8.1.400 #define TERM_READ 12

Definition at line 969 of file parser.c.

7.8.1.401 #define TERM_RECEIVE 13

Definition at line 970 of file parser.c.

7.8.1.402 #define TERM_RETURN 14

Definition at line 971 of file parser.c.

7.8.1.403 #define TERM_REWRITE 15

Definition at line 972 of file parser.c.

7.8.1.404 #define TERM_SEARCH 16

Definition at line 973 of file parser.c.

7.8.1.405 #define TERM_START 17

Definition at line 974 of file parser.c.

7.8.1.406 #define TERM_STRING 18

Definition at line 975 of file parser.c.

7.8.1.407 #define TERM_SUBTRACT 19

Definition at line 976 of file parser.c.

7.8.1.408 #define TERM_UNSTRING 20

Definition at line 977 of file parser.c.

7.8.1.409 #define TERM_WRITE 21

Definition at line 978 of file parser.c.

7.8.1.410 #define TERMINATE 641

Definition at line 878 of file parser.c.

7.8.1.411 #define TEST 642

Definition at line 879 of file parser.c.

7.8.1.412 #define THAN 643

Definition at line 880 of file parser.c.

7.8.1.413 #define THEN 644

Definition at line 881 of file parser.c.

7.8.1.414 #define THRU 645

Definition at line 882 of file parser.c.

7.8.1.415 #define TIME 646

Definition at line 883 of file parser.c.

7.8.1.416 #define TIMES 647

Definition at line 884 of file parser.c.

7.8.1.417 #define TO 648

Definition at line 885 of file parser.c.

7.8.1.418 #define TOK_FALSE 649

Definition at line 886 of file parser.c.

7.8.1.419 #define TOK_FILE 650

Definition at line 887 of file parser.c.

7.8.1.420 #define TOK_INITIAL 651

Definition at line 888 of file parser.c.

7.8.1.421 #define TOK_NULL 652

Definition at line 889 of file parser.c.

7.8.1.422 #define TOK_TRUE 653

Definition at line 890 of file parser.c.

7.8.1.423 #define TOKEN_EOF 0

Definition at line 494 of file parser.c.

7.8.1.424 #define TOP 654

Definition at line 891 of file parser.c.

7.8.1.425 #define TRAILING 655

Definition at line 892 of file parser.c.

7.8.1.426 #define TRANSFORM 656

Definition at line 893 of file parser.c.

7.8.1.427 #define TRIM_FUNCTION 657

Definition at line 894 of file parser.c.

7.8.1.428 #define TYPE 658

Definition at line 895 of file parser.c.

7.8.1.429 #define UNARY_SIGN 692

Definition at line 929 of file parser.c.

7.8.1.430 #define UNDERLINE 659

Definition at line 896 of file parser.c.

7.8.1.431 #define UNIT 660

Definition at line 897 of file parser.c.

7.8.1.432 #define UNLOCK 661

Definition at line 898 of file parser.c.

7.8.1.433 #define UNSIGNED 662

Definition at line 899 of file parser.c.

7.8.1.434 #define UNSIGNED_INT 663

Definition at line 900 of file parser.c.

7.8.1.435 #define UNSIGNED_LONG 664

Definition at line 901 of file parser.c.

7.8.1.436 #define UNSIGNED_SHORT 665

Definition at line 902 of file parser.c.

7.8.1.437 #define UNSTRING 666

Definition at line 903 of file parser.c.

7.8.1.438 #define UNTIL 667

Definition at line 904 of file parser.c.

7.8.1.439 #define UP 668

Definition at line 905 of file parser.c.

7.8.1.440 #define UPDATE 669

Definition at line 906 of file parser.c.

7.8.1.441 #define UPON 670

Definition at line 907 of file parser.c.

7.8.1.442 #define UPON_ARGUMENT_NUMBER 671

Definition at line 908 of file parser.c.

7.8.1.443 #define UPON_COMMAND_LINE 672

Definition at line 909 of file parser.c.

7.8.1.444 #define UPON_ENVIRONMENT_NAME 673

Definition at line 910 of file parser.c.

7.8.1.445 #define UPON_ENVIRONMENT_VALUE 674

Definition at line 911 of file parser.c.

7.8.1.446 #define UPPER_CASE_FUNC 675

Definition at line 912 of file parser.c.

7.8.1.447 #define USAGE 676

Definition at line 913 of file parser.c.

7.8.1.448 #define USE 677

Definition at line 914 of file parser.c.

7.8.1.449 #define USING 678

Definition at line 915 of file parser.c.

7.8.1.450 #define VALUE 679

Definition at line 916 of file parser.c.

7.8.1.451 #define VARYING 680

Definition at line 917 of file parser.c.

7.8.1.452 #define WAIT 681

Definition at line 918 of file parser.c.

7.8.1.453 #define WHEN 682

Definition at line 919 of file parser.c.

7.8.1.454 #define WHEN_COMPILED_FUNC 683

Definition at line 920 of file parser.c.

7.8.1.455 #define WITH 684

Definition at line 921 of file parser.c.

7.8.1.456 #define WORD 685

Definition at line 922 of file parser.c.

7.8.1.457 #define WORDS 686

Definition at line 923 of file parser.c.

7.8.1.458 #define WORKING_STORAGE 687

Definition at line 924 of file parser.c.

7.8.1.459 #define WRITE 688

Definition at line 925 of file parser.c.

7.8.1.460 #define YY_REDUCE_PRINT(*Rule*)

Value:

```
do {                                    \
  if (yydebug)                          \
    yy_reduce_print (Rule);             \
} while (0)
```

Definition at line 4900 of file parser.c.

7.8.1.461 #define YY_STACK_PRINT(*Bottom, Top*)

Value:

```
do {                                                            \
  if (yydebug)                                                  \
    yy_stack_print ((Bottom), (Top));                           \
} while (0)
```

Definition at line 4870 of file parser.c.

7.8.1.462 #define YYABORT goto yyabortlab

Definition at line 4768 of file parser.c.

7.8.1.463 #define YYACCEPT goto yyacceptlab

Definition at line 4767 of file parser.c.

7.8.1.464 #define YYBACKUP(*Token, Value*)

Value:

```
do                                                              \
  if (yychar == YYEMPTY && yylen == 1)                          \
    {                                                           \
      yychar = (Token);                                         \
      yylval = (Value);                                         \
      yytoken = YYTRANSLATE (yychar);                           \
      YYPOPSTACK;                                               \
      goto yybackup;                                            \
    }                                                           \
  else                                                          \
    {                                                           \
      yyerror ("syntax error: cannot back up");\
      YYERROR;                                                  \
    }                                                           \
while (0)
```

Definition at line 4779 of file parser.c.

7.8.1.465 #define YYBISON 1

Definition at line 37 of file parser.c.

7.8.1.466 #define yyclearin (yychar = YYEMPTY)

Definition at line 4763 of file parser.c.

7.8.1.467 #define YYCOPY(*To, From, Count*)

Value:

```
do                                      \
        {                                       \
          register YYSIZE_T yyi;                \
          for (yyi = 0; yyi < (Count); yyi++)   \
            (To)[yyi] = (From)[yyi];            \
        }                                       \
      while (0)
```

Definition at line 1305 of file parser.c.

7.8.1.468 #define YYDEBUG 1

Definition at line 946 of file parser.c.

7.8.1.469 #define YYDPRINTF(*Args*)

Value:

```
do {                                            \
  if (yydebug)                                  \
    YYFPRINTF Args;                             \
} while (0)
```

Definition at line 4826 of file parser.c.

7.8.1.470 #define YYDSYMPRINT(*Args*)

Value:

```
do {                                            \
  if (yydebug)                                  \
    yysymprint Args;                            \
} while (0)
```

Definition at line 4832 of file parser.c.

7.8.1.471 #define YYDSYMPRINTF(*Title, Token, Value, Location*)

Value:

```
do {                                                                    \
  if (yydebug)                                                          \
    {                                                                   \
      YYFPRINTF (stderr, "%s ", Title);                                 \
      yysymprint (stderr,                                               \
                  Token, Value);        \
      YYFPRINTF (stderr, "\n");                                         \
    }                                                                   \
} while (0)
```

Definition at line 4838 of file parser.c.

7.8.1.472 #define YYEMPTY (-2)

Definition at line 4764 of file parser.c.

7.8.1.473 #define YYEOF 0

Definition at line 4765 of file parser.c.

7.8.1.474 #define YYERRCODE 256

Definition at line 4797 of file parser.c.

7.8.1.475 #define yyerrok (yyerrstatus = 0)

Definition at line 4762 of file parser.c.

7.8.1.476 #define yyerror cb_error

Definition at line 945 of file parser.c.

7.8.1.477 #define YYERROR goto yyerrlab1

Definition at line 4769 of file parser.c.

7.8.1.478 #define YYERROR_VERBOSE 1

Definition at line 1226 of file parser.c.

7.8.1.479 #define YYERROR_VERBOSE 1

Definition at line 1226 of file parser.c.

7.8.1.480 #define YYFAIL goto yyerrlab

Definition at line 4775 of file parser.c.

7.8.1.481 #define YYFINAL 3

Definition at line 1341 of file parser.c.

7.8.1.482 #define YYFPRINTF fprintf

Definition at line 4823 of file parser.c.

7.8.1.483 #define YYINITDEPTH 200

Definition at line 4920 of file parser.c.

7.8.1.484 #define YYLAST 5462

Definition at line 1343 of file parser.c.

7.8.1.485 #define YYLEX yylex ()

Definition at line 4815 of file parser.c.

7.8.1.486 #define YYLLOC_DEFAULT(*Current, Rhs, N*)

Value:

```
Current.first_line   = Rhs[1].first_line;      \
  Current.first_column = Rhs[1].first_column;    \
  Current.last_line    = Rhs[N].last_line;       \
  Current.last_column  = Rhs[N].last_column;
```

Definition at line 4803 of file parser.c.

7.8.1.487 #define YYLSP_NEEDED 0

Definition at line 46 of file parser.c.

7.8.1.488 #define YYMAXDEPTH 10000

Definition at line 4935 of file parser.c.

7.8.1.489 #define YYMAXUTOK 692

Definition at line 1356 of file parser.c.

7.8.1.490 #define YYNNTS 690

Definition at line 1348 of file parser.c.

7.8.1.491 #define YYNRULES 1518

Definition at line 1350 of file parser.c.

7.8.1.492 #define YYNSTATES 2240

Definition at line 1352 of file parser.c.

7.8.1.493 #define YYNTOKENS 451

Definition at line 1346 of file parser.c.

7.8.1.494 #define YYPACT_NINF -1922

Definition at line 3106 of file parser.c.

7.8.1.495 #define YYPOPSTACK (yyvsp--, yyssp--)

7.8.1.496 #define YYPURE 0

Definition at line 43 of file parser.c.

7.8.1.497 #define YYRECOVERING( ) (!!yyerrstatus)

Definition at line 4777 of file parser.c.

7.8.1.498 #define YYSIZE_T unsigned int

Definition at line 4759 of file parser.c.

7.8.1.499 #define YYSKELETON_NAME "yacc.c"

Definition at line 40 of file parser.c.

7.8.1.500 #define YYSTACK_ALLOC malloc

Definition at line 1272 of file parser.c.

7.8.1.501 #define YYSTACK_BYTES(*N*)

Value:

```
((N) * (sizeof (short) + sizeof (YYSTYPE))                              \
      + YYSTACK_GAP_MAXIMUM)
```

Definition at line 1294 of file parser.c.

7.8.1.502 #define YYSTACK_FREE free

Definition at line 1273 of file parser.c.

7.8.1.503 #define YYSTACK_GAP_MAXIMUM (sizeof (union yyalloc) - 1)

Definition at line 1290 of file parser.c.

7.8.1.504 #define YYSTACK_RELOCATE(*Stack*)

Value:

```
do                                                                      \
      {                                                                 \
        YYSIZE_T yynewbytes;                                            \
        YYCOPY (&yyptr->Stack, Stack, yysize);                          \
        Stack = &yyptr->Stack;                                          \
        yynewbytes = yystacksize * sizeof (*Stack) + YYSTACK_GAP_MAXIMUM; \
        yyptr += yynewbytes / sizeof (*yyptr);                          \
      }                                                                 \
    while (0)
```

Definition at line 1321 of file parser.c.

7.8.1.505 #define YYTABLE_NINF -1518

Definition at line 3413 of file parser.c.

7.8.1.506 #define YYTERROR 1

Definition at line 4796 of file parser.c.

7.8.1.507 #define YYTOKENTYPE

Definition at line 52 of file parser.c.

7.8.1.508 #define YYTRANSLATE(*YYX*) ((unsigned int) (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)

Definition at line 1358 of file parser.c.

7.8.1.509 #define YYUNDEFTOK 2

Definition at line 1355 of file parser.c.

7.8.1.510 #define YYYYDDD 689

Definition at line 926 of file parser.c.

7.8.1.511 #define YYYYMMDD 690

Definition at line 927 of file parser.c.

7.8.1.512 #define ZERO 691

Definition at line 928 of file parser.c.

7.8.2 Typedef Documentation

7.8.2.1 typedef short yysigned_char

Definition at line 1337 of file parser.c.

7.8.3 Enumeration Type Documentation

7.8.3.1 enum yytokentype

Enumerator:

TOKEN_EOF

ACCEPT

ACCESS

ADD

ADDRESS

ADVANCING

AFTER

ALL

ALLOCATE

ALPHABET

ALPHABETIC

ALPHABETIC_LOWER

ALPHABETIC_UPPER

ALPHANUMERIC

ALPHANUMERIC_EDITED

ALSO

ALTER

ALTERNATE

AND

ANY

ARE

AREA

ARGUMENT_NUMBER

ARGUMENT_VALUE

AS

ASCENDING

ASSIGN

AT

AUTO

AUTOMATIC

BACKGROUND_COLOR

BASED

BEFORE

BELL

BINARY

BINARY_C_LONG

BINARY_CHAR

BINARY_DOUBLE

BINARY_LONG

BINARY_SHORT

BLANK

BLANK_LINE

BLANK_SCREEN

BLINK

BLOCK

BOTTOM

BY

BYTE_LENGTH

CALL

CANCEL

CH

CHAINING

CHARACTER

CHARACTERS

CLASS

CLOSE

CODE

CODE_SET

COLLATING

COL

COLS

COLUMN

COLUMNS

COMMA

COMMAND_LINE

COMMA_DELIM

COMMIT

COMMON

COMP

COMPUTE

COMP_1

COMP_2

COMP_3

COMP_4

COMP_5

COMP_X

CONCATENATE_FUNC

CONFIGURATION

CONSTANT

CONTAINS

CONTENT

CONTINUE

CONTROL
CONTROLS
CONTROL_FOOTING
CONTROL_HEADING
CONVERTING
CORRESPONDING
COUNT
CRT
CURRENCY
CURRENT_DATE_FUNC
CURSOR
CYCLE
DATA
DATE
DAY
DAY_OF_WEEK
DE
DEBUGGING
DECIMAL_POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EBCDIC
ELSE
END
END_ACCEPT

END_ADD
END_CALL
END_COMPUTE
END_DELETE
END_DISPLAY
END_DIVIDE
END_EVALUATE
END_FUNCTION
END_IF
END_MULTIPLY
END_PERFORM
END_PROGRAM
END_READ
END_RETURN
END_REWRITE
END_SEARCH
END_START
END_STRING
END_SUBTRACT
END_UNSTRING
END_WRITE
ENTRY
ENVIRONMENT
ENVIRONMENT_NAME
ENVIRONMENT_VALUE
EOL
EOP
EOS
EQUAL
EQUALS
ERASE
ERROR
ESCAPE
EVALUATE
EVENT_STATUS
EXCEPTION
EXCLUSIVE
EXIT

EXTEND
EXTERNAL
FD
FILE_CONTROL
FILE_ID
FILLER
FINAL
FIRST
FOOTING
FOR
FOREGROUND_COLOR
FOREVER
FREE
FROM
FULL
FUNCTION
FUNCTION_ID
FUNCTION_NAME
GE
GENERATE
GIVING
GLOBAL
GO
GOBACK
GREATER
GROUP
HEADING
HIGHLIGHT
HIGH_VALUE
IDENTIFICATION
IF
IGNORE
IGNORING
IN
INDEX
INDEXED
INDICATE
INITIALIZE

INITIALIZED
INITIATE
INPUT
INPUT_OUTPUT
INSPECT
INTO
INTRINSIC
INVALID
INVALID_KEY
IS
I_O
I_O_CONTROL
JUSTIFIED
KEY
LABEL
LAST
LAST_DETAIL
LE
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE_COUNTER
LINE
LINES
LINKAGE
LITERAL
LOCALE
LOCALE_DT_FUNC
LOCAL_STORAGE
LOCK
LOWER_CASE_FUNC
LOWLIGHT
LOW_VALUE
MANUAL

MEMORY
MERGE
MINUS
MNEMONIC_NAME
MODE
MOVE
MULTIPLE
MULTIPLY
NATIONAL
NATIONAL_EDITED
NATIVE
NE
NEGATIVE
NEXT
NEXT_SENTENCE
NO
NOT
NOT_END
NOT_EOP
NOT_EXCEPTION
NOT_INVALID_KEY
NOT_OVERFLOW
NOT_SIZE_ERROR
NO_ADVANCING
NUMBER
NUMBERS
NUMERIC
NUMERIC_EDITED
NUMVALC_FUNC
OBJECT_COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL

OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
OVERLINE
PACKED_DECIMAL
PADDING
PAGE
PAGE_FOOTING
PAGE_HEADING
PARAGRAPH
PERFORM
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRESENT
PREVIOUS
PRINTER
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM_ID
PROGRAM_NAME
PROGRAM_POINTER
PROMPT
QUOTE
RANDOM
RD
READ
RECORD
RECORDING
RECORDS

RECURSIVE
REDEFINES
REEL
REFERENCE
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
REPORT_FOOTING
REPORT_HEADING
REPOSITORY
REQUIRED
RESERVE
RETURN
RETURNING
REVERSE_FUNC
REVERSE_VIDEO
REWIND
REWRITE
RIGHT
ROLLBACK
ROUNDED
RUN
SAME
SCREEN
SCREEN_CONTROL
SCROLL
SD
SEARCH
SECTION
SECURE
SEGMENT_LIMIT
SELECT

SEMI_COLON
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHARING
SIGN
SIGNED
SIGNED_INT
SIGNED_LONG
SIGNED_SHORT
SIZE
SIZE_ERROR
SORT
SORT_MERGE
SOURCE
SOURCE_COMPUTER
SPACE
SPECIAL_NAMES
STANDARD
STANDARD_1
STANDARD_2
START
STATUS
STOP
STRING
SUBSTITUTE_FUNC
SUBSTITUTE_CASE_FUNC
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNCHRONIZED
TALLYING
TAPE
TERMINATE
TEST

THAN

THEN

THRU

TIME

TIMES

TO

TOK_FALSE

TOK_FILE

TOK_INITIAL

TOK_NULL

TOK_TRUE

TOP

TRAILING

TRANSFORM

TRIM_FUNCTION

TYPE

UNDERLINE

UNIT

UNLOCK

UNSIGNED

UNSIGNED_INT

UNSIGNED_LONG

UNSIGNED_SHORT

UNSTRING

UNTIL

UP

UPDATE

UPON

UPON_ARGUMENT_NUMBER

UPON_COMMAND_LINE

UPON_ENVIRONMENT_NAME

UPON_ENVIRONMENT_VALUE

UPPER_CASE_FUNC

USAGE

USE

USING

VALUE

VARYING

WAIT
WHEN
WHEN_COMPILED_FUNC
WITH
WORD
WORDS
WORKING_STORAGE
WRITE
YYYYDDD
YYYYMMDD
ZERO
UNARY_SIGN
TOKEN_EOF
ACCEPT
ACCESS
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALLOCATE
ALPHABET
ALPHABETIC
ALPHABETIC_LOWER
ALPHABETIC_UPPER
ALPHANUMERIC
ALPHANUMERIC_EDITED
ALSO
ALTER
ALTERNATE
AND
ANY
ARE
AREA
ARGUMENT_NUMBER
ARGUMENT_VALUE
AS
ASCENDING

ASSIGN

AT

AUTO

AUTOMATIC

BACKGROUND_COLOR

BASED

BEFORE

BELL

BINARY

BINARY_C_LONG

BINARY_CHAR

BINARY_DOUBLE

BINARY_LONG

BINARY_SHORT

BLANK

BLANK_LINE

BLANK_SCREEN

BLINK

BLOCK

BOTTOM

BY

BYTE_LENGTH

CALL

CANCEL

CH

CHAINING

CHARACTER

CHARACTERS

CLASS

CLOSE

CODE

CODE_SET

COLLATING

COL

COLS

COLUMN

COLUMNS

COMMA

COMMAND_LINE
COMMA_DELIM
COMMIT
COMMON
COMP
COMPUTE
COMP_1
COMP_2
COMP_3
COMP_4
COMP_5
COMP_X
CONCATENATE_FUNC
CONFIGURATION
CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONTROL_FOOTING
CONTROL_HEADING
CONVERTING
CORRESPONDING
COUNT
CRT
CURRENCY
CURRENT_DATE_FUNC
CURSOR
CYCLE
DATA
DATE
DAY
DAY_OF_WEEK
DE
DEBUGGING
DECIMAL_POINT
DECLARATIVES

DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EBCDIC
ELSE
END
END_ACCEPT
END_ADD
END_CALL
END_COMPUTE
END_DELETE
END_DISPLAY
END_DIVIDE
END_EVALUATE
END_FUNCTION
END_IF
END_MULTIPLY
END_PERFORM
END_PROGRAM
END_READ
END_RETURN
END_REWRITE
END_SEARCH
END_START
END_STRING
END_SUBTRACT
END_UNSTRING

END_WRITE

ENTRY

ENVIRONMENT

ENVIRONMENT_NAME

ENVIRONMENT_VALUE

EOL

EOP

EOS

EQUAL

EQUALS

ERASE

ERROR

ESCAPE

EVALUATE

EVENT_STATUS

EXCEPTION

EXCLUSIVE

EXIT

EXTEND

EXTERNAL

FD

FILE_CONTROL

FILE_ID

FILLER

FINAL

FIRST

FOOTING

FOR

FOREGROUND_COLOR

FOREVER

FREE

FROM

FULL

FUNCTION

FUNCTION_ID

FUNCTION_NAME

GE

GENERATE

GIVING
GLOBAL
GO
GOBACK
GREATER
GROUP
HEADING
HIGHLIGHT
HIGH_VALUE
IDENTIFICATION
IF
IGNORE
IGNORING
IN
INDEX
INDEXED
INDICATE
INITIALIZE
INITIALIZED
INITIATE
INPUT
INPUT_OUTPUT
INSPECT
INTO
INTRINSIC
INVALID
INVALID_KEY
IS
I_O
I_O_CONTROL
JUSTIFIED
KEY
LABEL
LAST
LAST_DETAIL
LE
LEADING
LEFT

LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE_COUNTER
LINE
LINES
LINKAGE
LITERAL
LOCALE
LOCALE_DT_FUNC
LOCAL_STORAGE
LOCK
LOWER_CASE_FUNC
LOWLIGHT
LOW_VALUE
MANUAL
MEMORY
MERGE
MINUS
MNEMONIC_NAME
MODE
MOVE
MULTIPLE
MULTIPLY
NATIONAL
NATIONAL_EDITED
NATIVE
NE
NEGATIVE
NEXT
NEXT_SENTENCE
NO
NOT
NOT_END
NOT_EOP
NOT_EXCEPTION

NOT_INVALID_KEY
NOT_OVERFLOW
NOT_SIZE_ERROR
NO_ADVANCING
NUMBER
NUMBERS
NUMERIC
NUMERIC_EDITED
NUMVALC_FUNC
OBJECT_COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
OVERLINE
PACKED_DECIMAL
PADDING
PAGE
PAGE_FOOTING
PAGE_HEADING
PARAGRAPH
PERFORM
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRESENT

PREVIOUS
PRINTER
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM_ID
PROGRAM_NAME
PROGRAM_POINTER
PROMPT
QUOTE
RANDOM
RD
READ
RECORD
RECORDING
RECORDS
RECURSIVE
REDEFINES
REEL
REFERENCE
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
REPORT_FOOTING
REPORT_HEADING
REPOSITORY
REQUIRED
RESERVE
RETURN
RETURNING

REVERSE_FUNC
REVERSE_VIDEO
REWIND
REWRITE
RIGHT
ROLLBACK
ROUNDED
RUN
SAME
SCREEN
SCREEN_CONTROL
SCROLL
SD
SEARCH
SECTION
SECURE
SEGMENT_LIMIT
SELECT
SEMI_COLON
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHARING
SIGN
SIGNED
SIGNED_INT
SIGNED_LONG
SIGNED_SHORT
SIZE
SIZE_ERROR
SORT
SORT_MERGE
SOURCE
SOURCE_COMPUTER
SPACE
SPECIAL_NAMES

STANDARD

STANDARD_1

STANDARD_2

START

STATUS

STOP

STRING

SUBSTITUTE_FUNC

SUBSTITUTE_CASE_FUNC

SUBTRACT

SUM

SUPPRESS

SYMBOLIC

SYNCHRONIZED

TALLYING

TAPE

TERMINATE

TEST

THAN

THEN

THRU

TIME

TIMES

TO

TOK_FALSE

TOK_FILE

TOK_INITIAL

TOK_NULL

TOK_TRUE

TOP

TRAILING

TRANSFORM

TRIM_FUNCTION

TYPE

UNDERLINE

UNIT

UNLOCK

UNSIGNED

UNSIGNED_INT
UNSIGNED_LONG
UNSIGNED_SHORT
UNSTRING
UNTIL
UP
UPDATE
UPON
UPON_ARGUMENT_NUMBER
UPON_COMMAND_LINE
UPON_ENVIRONMENT_NAME
UPON_ENVIRONMENT_VALUE
UPPER_CASE_FUNC
USAGE
USE
USING
VALUE
VARYING
WAIT
WHEN
WHEN_COMPILED_FUNC
WITH
WORD
WORDS
WORKING_STORAGE
WRITE
YYYYDDD
YYYYMMDD
ZERO
UNARY_SIGN
TOKEN_EOF
COPY
REPLACE
SUPPRESS
PRINTING
REPLACING
OFF
IN

OF

BY

EQEQ

TOKEN

TOKEN_EOF

COPY

REPLACE

SUPPRESS

PRINTING

REPLACING

OFF

IN

OF

BY

EQEQ

TOKEN

Definition at line 55 of file parser.c.

```
                {
TOKEN_EOF = 0,
ACCEPT = 258,
ACCESS = 259,
ADD = 260,
ADDRESS = 261,
ADVANCING = 262,
AFTER = 263,
ALL = 264,
ALLOCATE = 265,
ALPHABET = 266,
ALPHABETIC = 267,
ALPHABETIC_LOWER = 268,
ALPHABETIC_UPPER = 269,
ALPHANUMERIC = 270,
ALPHANUMERIC_EDITED = 271,
ALSO = 272,
ALTER = 273,
ALTERNATE = 274,
AND = 275,
ANY = 276,
ARE = 277,
AREA = 278,
ARGUMENT_NUMBER = 279,
ARGUMENT_VALUE = 280,
AS = 281,
ASCENDING = 282,
ASSIGN = 283,
AT = 284,
AUTO = 285,
AUTOMATIC = 286,
BACKGROUND_COLOR = 287,
```

```
  BASED = 288,
  BEFORE = 289,
  BELL = 290,
  BINARY = 291,
  BINARY_C_LONG = 292,
  BINARY_CHAR = 293,
  BINARY_DOUBLE = 294,
  BINARY_LONG = 295,
  BINARY_SHORT = 296,
  BLANK = 297,
  BLANK_LINE = 298,
  BLANK_SCREEN = 299,
  BLINK = 300,
  BLOCK = 301,
  BOTTOM = 302,
  BY = 303,
  BYTE_LENGTH = 304,
  CALL = 305,
  CANCEL = 306,
  CH = 307,
  CHAINING = 308,
  CHARACTER = 309,
  CHARACTERS = 310,
  CLASS = 311,
  CLOSE = 312,
  CODE = 313,
  CODE_SET = 314,
  COLLATING = 315,
  COL = 316,
  COLS = 317,
  COLUMN = 318,
  COLUMNS = 319,
  COMMA = 320,
  COMMAND_LINE = 321,
  COMMA_DELIM = 322,
  COMMIT = 323,
  COMMON = 324,
  COMP = 325,
  COMPUTE = 326,
  COMP_1 = 327,
  COMP_2 = 328,
  COMP_3 = 329,
  COMP_4 = 330,
  COMP_5 = 331,
  COMP_X = 332,
  CONCATENATE_FUNC = 333,
  CONFIGURATION = 334,
  CONSTANT = 335,
  CONTAINS = 336,
  CONTENT = 337,
  CONTINUE = 338,
  CONTROL = 339,
  CONTROLS = 340,
  CONTROL_FOOTING = 341,
  CONTROL_HEADING = 342,
  CONVERTING = 343,
  CORRESPONDING = 344,
  COUNT = 345,
  CRT = 346,
  CURRENCY = 347,
  CURRENT_DATE_FUNC = 348,
  CURSOR = 349,
```

```
CYCLE = 350,
DATA = 351,
DATE = 352,
DAY = 353,
DAY_OF_WEEK = 354,
DE = 355,
DEBUGGING = 356,
DECIMAL_POINT = 357,
DECLARATIVES = 358,
DEFAULT = 359,
DELETE = 360,
DELIMITED = 361,
DELIMITER = 362,
DEPENDING = 363,
DESCENDING = 364,
DETAIL = 365,
DISK = 366,
DISPLAY = 367,
DIVIDE = 368,
DIVISION = 369,
DOWN = 370,
DUPLICATES = 371,
DYNAMIC = 372,
EBCDIC = 373,
ELSE = 374,
END = 375,
END_ACCEPT = 376,
END_ADD = 377,
END_CALL = 378,
END_COMPUTE = 379,
END_DELETE = 380,
END_DISPLAY = 381,
END_DIVIDE = 382,
END_EVALUATE = 383,
END_FUNCTION = 384,
END_IF = 385,
END_MULTIPLY = 386,
END_PERFORM = 387,
END_PROGRAM = 388,
END_READ = 389,
END_RETURN = 390,
END_REWRITE = 391,
END_SEARCH = 392,
END_START = 393,
END_STRING = 394,
END_SUBTRACT = 395,
END_UNSTRING = 396,
END_WRITE = 397,
ENTRY = 398,
ENVIRONMENT = 399,
ENVIRONMENT_NAME = 400,
ENVIRONMENT_VALUE = 401,
EOL = 402,
EOP = 403,
EOS = 404,
EQUAL = 405,
EQUALS = 406,
ERASE = 407,
ERROR = 408,
ESCAPE = 409,
EVALUATE = 410,
EVENT_STATUS = 411,
```

```
  EXCEPTION = 412,
  EXCLUSIVE = 413,
  EXIT = 414,
  EXTEND = 415,
  EXTERNAL = 416,
  FD = 417,
  FILE_CONTROL = 418,
  FILE_ID = 419,
  FILLER = 420,
  FINAL = 421,
  FIRST = 422,
  FOOTING = 423,
  FOR = 424,
  FOREGROUND_COLOR = 425,
  FOREVER = 426,
  FREE = 427,
  FROM = 428,
  FULL = 429,
  FUNCTION = 430,
  FUNCTION_ID = 431,
  FUNCTION_NAME = 432,
  GE = 433,
  GENERATE = 434,
  GIVING = 435,
  GLOBAL = 436,
  GO = 437,
  GOBACK = 438,
  GREATER = 439,
  GROUP = 440,
  HEADING = 441,
  HIGHLIGHT = 442,
  HIGH_VALUE = 443,
  IDENTIFICATION = 444,
  IF = 445,
  IGNORE = 446,
  IGNORING = 447,
  IN = 448,
  INDEX = 449,
  INDEXED = 450,
  INDICATE = 451,
  INITIALIZE = 452,
  INITIALIZED = 453,
  INITIATE = 454,
  INPUT = 455,
  INPUT_OUTPUT = 456,
  INSPECT = 457,
  INTO = 458,
  INTRINSIC = 459,
  INVALID = 460,
  INVALID_KEY = 461,
  IS = 462,
  I_O = 463,
  I_O_CONTROL = 464,
  JUSTIFIED = 465,
  KEY = 466,
  LABEL = 467,
  LAST = 468,
  LAST_DETAIL = 469,
  LE = 470,
  LEADING = 471,
  LEFT = 472,
  LENGTH = 473,
```

```
LESS = 474,
LIMIT = 475,
LIMITS = 476,
LINAGE = 477,
LINAGE_COUNTER = 478,
LINE = 479,
LINES = 480,
LINKAGE = 481,
LITERAL = 482,
LOCALE = 483,
LOCALE_DT_FUNC = 484,
LOCAL_STORAGE = 485,
LOCK = 486,
LOWER_CASE_FUNC = 487,
LOWLIGHT = 488,
LOW_VALUE = 489,
MANUAL = 490,
MEMORY = 491,
MERGE = 492,
MINUS = 493,
MNEMONIC_NAME = 494,
MODE = 495,
MOVE = 496,
MULTIPLE = 497,
MULTIPLY = 498,
NATIONAL = 499,
NATIONAL_EDITED = 500,
NATIVE = 501,
NE = 502,
NEGATIVE = 503,
NEXT = 504,
NEXT_SENTENCE = 505,
NO = 506,
NOT = 507,
NOT_END = 508,
NOT_EOP = 509,
NOT_EXCEPTION = 510,
NOT_INVALID_KEY = 511,
NOT_OVERFLOW = 512,
NOT_SIZE_ERROR = 513,
NO_ADVANCING = 514,
NUMBER = 515,
NUMBERS = 516,
NUMERIC = 517,
NUMERIC_EDITED = 518,
NUMVALC_FUNC = 519,
OBJECT_COMPUTER = 520,
OCCURS = 521,
OF = 522,
OFF = 523,
OMITTED = 524,
ON = 525,
ONLY = 526,
OPEN = 527,
OPTIONAL = 528,
OR = 529,
ORDER = 530,
ORGANIZATION = 531,
OTHER = 532,
OUTPUT = 533,
OVERFLOW = 534,
OVERLINE = 535,
```

```
  PACKED_DECIMAL = 536,
  PADDING = 537,
  PAGE = 538,
  PAGE_FOOTING = 539,
  PAGE_HEADING = 540,
  PARAGRAPH = 541,
  PERFORM = 542,
  PICTURE = 543,
  PLUS = 544,
  POINTER = 545,
  POSITION = 546,
  POSITIVE = 547,
  PRESENT = 548,
  PREVIOUS = 549,
  PRINTER = 550,
  PRINTING = 551,
  PROCEDURE = 552,
  PROCEDURES = 553,
  PROCEED = 554,
  PROGRAM = 555,
  PROGRAM_ID = 556,
  PROGRAM_NAME = 557,
  PROGRAM_POINTER = 558,
  PROMPT = 559,
  QUOTE = 560,
  RANDOM = 561,
  RD = 562,
  READ = 563,
  RECORD = 564,
  RECORDING = 565,
  RECORDS = 566,
  RECURSIVE = 567,
  REDEFINES = 568,
  REEL = 569,
  REFERENCE = 570,
  RELATIVE = 571,
  RELEASE = 572,
  REMAINDER = 573,
  REMOVAL = 574,
  RENAMES = 575,
  REPLACING = 576,
  REPORT = 577,
  REPORTING = 578,
  REPORTS = 579,
  REPORT_FOOTING = 580,
  REPORT_HEADING = 581,
  REPOSITORY = 582,
  REQUIRED = 583,
  RESERVE = 584,
  RETURN = 585,
  RETURNING = 586,
  REVERSE_FUNC = 587,
  REVERSE_VIDEO = 588,
  REWIND = 589,
  REWRITE = 590,
  RIGHT = 591,
  ROLLBACK = 592,
  ROUNDED = 593,
  RUN = 594,
  SAME = 595,
  SCREEN = 596,
  SCREEN_CONTROL = 597,
```

```
SCROLL = 598,
SD = 599,
SEARCH = 600,
SECTION = 601,
SECURE = 602,
SEGMENT_LIMIT = 603,
SELECT = 604,
SEMI_COLON = 605,
SENTENCE = 606,
SEPARATE = 607,
SEQUENCE = 608,
SEQUENTIAL = 609,
SET = 610,
SHARING = 611,
SIGN = 612,
SIGNED = 613,
SIGNED_INT = 614,
SIGNED_LONG = 615,
SIGNED_SHORT = 616,
SIZE = 617,
SIZE_ERROR = 618,
SORT = 619,
SORT_MERGE = 620,
SOURCE = 621,
SOURCE_COMPUTER = 622,
SPACE = 623,
SPECIAL_NAMES = 624,
STANDARD = 625,
STANDARD_1 = 626,
STANDARD_2 = 627,
START = 628,
STATUS = 629,
STOP = 630,
STRING = 631,
SUBSTITUTE_FUNC = 632,
SUBSTITUTE_CASE_FUNC = 633,
SUBTRACT = 634,
SUM = 635,
SUPPRESS = 636,
SYMBOLIC = 637,
SYNCHRONIZED = 638,
TALLYING = 639,
TAPE = 640,
TERMINATE = 641,
TEST = 642,
THAN = 643,
THEN = 644,
THRU = 645,
TIME = 646,
TIMES = 647,
TO = 648,
TOK_FALSE = 649,
TOK_FILE = 650,
TOK_INITIAL = 651,
TOK_NULL = 652,
TOK_TRUE = 653,
TOP = 654,
TRAILING = 655,
TRANSFORM = 656,
TRIM_FUNCTION = 657,
TYPE = 658,
UNDERLINE = 659,
```

```
  UNIT = 660,
  UNLOCK = 661,
  UNSIGNED = 662,
  UNSIGNED_INT = 663,
  UNSIGNED_LONG = 664,
  UNSIGNED_SHORT = 665,
  UNSTRING = 666,
  UNTIL = 667,
  UP = 668,
  UPDATE = 669,
  UPON = 670,
  UPON_ARGUMENT_NUMBER = 671,
  UPON_COMMAND_LINE = 672,
  UPON_ENVIRONMENT_NAME = 673,
  UPON_ENVIRONMENT_VALUE = 674,
  UPPER_CASE_FUNC = 675,
  USAGE = 676,
  USE = 677,
  USING = 678,
  VALUE = 679,
  VARYING = 680,
  WAIT = 681,
  WHEN = 682,
  WHEN_COMPILED_FUNC = 683,
  WITH = 684,
  WORD = 685,
  WORDS = 686,
  WORKING_STORAGE = 687,
  WRITE = 688,
  YYYYDDD = 689,
  YYYYMMDD = 690,
  ZERO = 691,
  UNARY_SIGN = 692
};
```

7.8.4 Function Documentation

7.8.4.1 int yyparse (void)

7.8.5 Variable Documentation

7.8.5.1 struct cb_label* current_paragraph = NULL

Definition at line 986 of file parser.c.

7.8.5.2 struct cb_program* current_program = NULL

Definition at line 983 of file parser.c.

7.8.5.3 struct cb_label* current_section = NULL

Definition at line 985 of file parser.c.

7.8.5.4 struct cb_statement* current_statement = NULL

Definition at line 984 of file parser.c.

7.8.5.5 size_t functions_are_all = 0

Definition at line 987 of file parser.c.

7.8.5.6 int non_const_word = 0

Definition at line 988 of file parser.c.

7.8.5.7 int yychar

Definition at line 5078 of file parser.c.

7.8.5.8 int yydebug

Definition at line 4908 of file parser.c.

7.8.5.9 YYSTYPE yylval

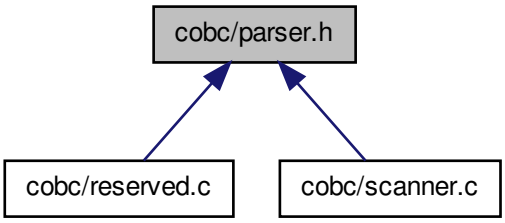
Definition at line 5081 of file parser.c.

7.8.5.10 int yynerrs

Definition at line 5084 of file parser.c.

7.9 cobc/parser.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define TOKEN_EOF 0
- #define ACCEPT 258
- #define ACCESS 259
- #define ADD 260
- #define ADDRESS 261
- #define ADVANCING 262
- #define AFTER 263
- #define ALL 264
- #define ALLOCATE 265
- #define ALPHABET 266
- #define ALPHABETIC 267
- #define ALPHABETIC_LOWER 268
- #define ALPHABETIC_UPPER 269
- #define ALPHANUMERIC 270
- #define ALPHANUMERIC_EDITED 271
- #define ALSO 272
- #define ALTER 273
- #define ALTERNATE 274
- #define AND 275
- #define ANY 276
- #define ARE 277
- #define AREA 278
- #define ARGUMENT_NUMBER 279
- #define ARGUMENT_VALUE 280
- #define AS 281

- #define COMP_2 328
- #define COMP_3 329
- #define COMP_4 330
- #define COMP_5 331
- #define COMP_X 332
- #define CONCATENATE_FUNC 333
- #define CONFIGURATION 334
- #define CONSTANT 335
- #define CONTAINS 336
- #define CONTENT 337
- #define CONTINUE 338
- #define CONTROL 339
- #define CONTROLS 340
- #define CONTROL_FOOTING 341
- #define CONTROL_HEADING 342
- #define CONVERTING 343
- #define CORRESPONDING 344
- #define COUNT 345
- #define CRT 346
- #define CURRENCY 347
- #define CURRENT_DATE_FUNC 348
- #define CURSOR 349
- #define CYCLE 350
- #define DATA 351
- #define DATE 352
- #define DAY 353
- #define DAY_OF_WEEK 354
- #define DE 355
- #define DEBUGGING 356
- #define DECIMAL_POINT 357
- #define DECLARATIVES 358
- #define DEFAULT 359
- #define DELETE 360
- #define DELIMITED 361
- #define DELIMITER 362
- #define DEPENDING 363
- #define DESCENDING 364
- #define DETAIL 365
- #define DISK 366
- #define DISPLAY 367
- #define DIVIDE 368
- #define DIVISION 369
- #define DOWN 370
- #define DUPLICATES 371
- #define DYNAMIC 372
- #define EBCDIC 373

- #define ELSE 374
- #define END 375
- #define END_ACCEPT 376
- #define END_ADD 377
- #define END_CALL 378
- #define END_COMPUTE 379
- #define END_DELETE 380
- #define END_DISPLAY 381
- #define END_DIVIDE 382
- #define END_EVALUATE 383
- #define END_FUNCTION 384
- #define END_IF 385
- #define END_MULTIPLY 386
- #define END_PERFORM 387
- #define END_PROGRAM 388
- #define END_READ 389
- #define END_RETURN 390
- #define END_REWRITE 391
- #define END_SEARCH 392
- #define END_START 393
- #define END_STRING 394
- #define END_SUBTRACT 395
- #define END_UNSTRING 396
- #define END_WRITE 397
- #define ENTRY 398
- #define ENVIRONMENT 399
- #define ENVIRONMENT_NAME 400
- #define ENVIRONMENT_VALUE 401
- #define EOL 402
- #define EOP 403
- #define EOS 404
- #define EQUAL 405
- #define EQUALS 406
- #define ERASE 407
- #define ERROR 408
- #define ESCAPE 409
- #define EVALUATE 410
- #define EVENT_STATUS 411
- #define EXCEPTION 412
- #define EXCLUSIVE 413
- #define EXIT 414
- #define EXTEND 415
- #define EXTERNAL 416
- #define FD 417
- #define FILE_CONTROL 418
- #define FILE_ID 419

- #define FILLER 420
- #define FINAL 421
- #define FIRST 422
- #define FOOTING 423
- #define FOR 424
- #define FOREGROUND_COLOR 425
- #define FOREVER 426
- #define FREE 427
- #define FROM 428
- #define FULL 429
- #define FUNCTION 430
- #define FUNCTION_ID 431
- #define FUNCTION_NAME 432
- #define GE 433
- #define GENERATE 434
- #define GIVING 435
- #define GLOBAL 436
- #define GO 437
- #define GOBACK 438
- #define GREATER 439
- #define GROUP 440
- #define HEADING 441
- #define HIGHLIGHT 442
- #define HIGH_VALUE 443
- #define IDENTIFICATION 444
- #define IF 445
- #define IGNORE 446
- #define IGNORING 447
- #define IN 448
- #define INDEX 449
- #define INDEXED 450
- #define INDICATE 451
- #define INITIALIZE 452
- #define INITIALIZED 453
- #define INITIATE 454
- #define INPUT 455
- #define INPUT_OUTPUT 456
- #define INSPECT 457
- #define INTO 458
- #define INTRINSIC 459
- #define INVALID 460
- #define INVALID_KEY 461
- #define IS 462
- #define I_O 463
- #define I_O_CONTROL 464
- #define JUSTIFIED 465

- #define NOT_OVERFLOW 512
- #define NOT_SIZE_ERROR 513
- #define NO_ADVANCING 514
- #define NUMBER 515
- #define NUMBERS 516
- #define NUMERIC 517
- #define NUMERIC_EDITED 518
- #define NUMVALC_FUNC 519
- #define OBJECT_COMPUTER 520
- #define OCCURS 521
- #define OF 522
- #define OFF 523
- #define OMITTED 524
- #define ON 525
- #define ONLY 526
- #define OPEN 527
- #define OPTIONAL 528
- #define OR 529
- #define ORDER 530
- #define ORGANIZATION 531
- #define OTHER 532
- #define OUTPUT 533
- #define OVERFLOW 534
- #define OVERLINE 535
- #define PACKED_DECIMAL 536
- #define PADDING 537
- #define PAGE 538
- #define PAGE_FOOTING 539
- #define PAGE_HEADING 540
- #define PARAGRAPH 541
- #define PERFORM 542
- #define PICTURE 543
- #define PLUS 544
- #define POINTER 545
- #define POSITION 546
- #define POSITIVE 547
- #define PRESENT 548
- #define PREVIOUS 549
- #define PRINTER 550
- #define PRINTING 551
- #define PROCEDURE 552
- #define PROCEDURES 553
- #define PROCEED 554
- #define PROGRAM 555
- #define PROGRAM_ID 556
- #define PROGRAM_NAME 557

- #define PROGRAM_POINTER 558
- #define PROMPT 559
- #define QUOTE 560
- #define RANDOM 561
- #define RD 562
- #define READ 563
- #define RECORD 564
- #define RECORDING 565
- #define RECORDS 566
- #define RECURSIVE 567
- #define REDEFINES 568
- #define REEL 569
- #define REFERENCE 570
- #define RELATIVE 571
- #define RELEASE 572
- #define REMAINDER 573
- #define REMOVAL 574
- #define RENAMES 575
- #define REPLACING 576
- #define REPORT 577
- #define REPORTING 578
- #define REPORTS 579
- #define REPORT_FOOTING 580
- #define REPORT_HEADING 581
- #define REPOSITORY 582
- #define REQUIRED 583
- #define RESERVE 584
- #define RETURN 585
- #define RETURNING 586
- #define REVERSE_FUNC 587
- #define REVERSE_VIDEO 588
- #define REWIND 589
- #define REWRITE 590
- #define RIGHT 591
- #define ROLLBACK 592
- #define ROUNDED 593
- #define RUN 594
- #define SAME 595
- #define SCREEN 596
- #define SCREEN_CONTROL 597
- #define SCROLL 598
- #define SD 599
- #define SEARCH 600
- #define SECTION 601
- #define SECURE 602
- #define SEGMENT_LIMIT 603

- #define SELECT 604
- #define SEMI_COLON 605
- #define SENTENCE 606
- #define SEPARATE 607
- #define SEQUENCE 608
- #define SEQUENTIAL 609
- #define SET 610
- #define SHARING 611
- #define SIGN 612
- #define SIGNED 613
- #define SIGNED_INT 614
- #define SIGNED_LONG 615
- #define SIGNED_SHORT 616
- #define SIZE 617
- #define SIZE_ERROR 618
- #define SORT 619
- #define SORT_MERGE 620
- #define SOURCE 621
- #define SOURCE_COMPUTER 622
- #define SPACE 623
- #define SPECIAL_NAMES 624
- #define STANDARD 625
- #define STANDARD_1 626
- #define STANDARD_2 627
- #define START 628
- #define STATUS 629
- #define STOP 630
- #define STRING 631
- #define SUBSTITUTE_FUNC 632
- #define SUBSTITUTE_CASE_FUNC 633
- #define SUBTRACT 634
- #define SUM 635
- #define SUPPRESS 636
- #define SYMBOLIC 637
- #define SYNCHRONIZED 638
- #define TALLYING 639
- #define TAPE 640
- #define TERMINATE 641
- #define TEST 642
- #define THAN 643
- #define THEN 644
- #define THRU 645
- #define TIME 646
- #define TIMES 647
- #define TO 648
- #define TOK_FALSE 649

- #define TOK_FILE 650
- #define TOK_INITIAL 651
- #define TOK_NULL 652
- #define TOK_TRUE 653
- #define TOP 654
- #define TRAILING 655
- #define TRANSFORM 656
- #define TRIM_FUNCTION 657
- #define TYPE 658
- #define UNDERLINE 659
- #define UNIT 660
- #define UNLOCK 661
- #define UNSIGNED 662
- #define UNSIGNED_INT 663
- #define UNSIGNED_LONG 664
- #define UNSIGNED_SHORT 665
- #define UNSTRING 666
- #define UNTIL 667
- #define UP 668
- #define UPDATE 669
- #define UPON 670
- #define UPON_ARGUMENT_NUMBER 671
- #define UPON_COMMAND_LINE 672
- #define UPON_ENVIRONMENT_NAME 673
- #define UPON_ENVIRONMENT_VALUE 674
- #define UPPER_CASE_FUNC 675
- #define USAGE 676
- #define USE 677
- #define USING 678
- #define VALUE 679
- #define VARYING 680
- #define WAIT 681
- #define WHEN 682
- #define WHEN_COMPILED_FUNC 683
- #define WITH 684
- #define WORD 685
- #define WORDS 686
- #define WORKING_STORAGE 687
- #define WRITE 688
- #define YYYYDDD 689
- #define YYYYMMDD 690
- #define ZERO 691
- #define UNARY_SIGN 692
- #define yystype YYSTYPE
- #define YYSTYPE_IS_DECLARED 1
- #define YYSTYPE_IS_TRIVIAL 1

Typedefs

- typedef int YYSTYPE

Enumerations

- enum yytokentype {

  TOKEN_EOF = 0, ACCEPT = 258, ACCESS = 259, ADD = 260,

  ADDRESS = 261, ADVANCING = 262, AFTER = 263, ALL = 264,

  ALLOCATE = 265, ALPHABET = 266, ALPHABETIC = 267, ALPHABETIC_-LOWER = 268,

  ALPHABETIC_UPPER = 269, ALPHANUMERIC = 270, ALPHANUMERIC_EDITED = 271, ALSO = 272,

  ALTER = 273, ALTERNATE = 274, AND = 275, ANY = 276,

  ARE = 277, AREA = 278, ARGUMENT_NUMBER = 279, ARGUMENT_VALUE = 280,

  AS = 281, ASCENDING = 282, ASSIGN = 283, AT = 284,

  AUTO = 285, AUTOMATIC = 286, BACKGROUND_COLOR = 287, BASED = 288,

  BEFORE = 289, BELL = 290, BINARY = 291, BINARY_C_LONG = 292,

  BINARY_CHAR = 293, BINARY_DOUBLE = 294, BINARY_LONG = 295, BINARY_-SHORT = 296,

  BLANK = 297, BLANK_LINE = 298, BLANK_SCREEN = 299, BLINK = 300,

  BLOCK = 301, BOTTOM = 302, BY = 303, BYTE_LENGTH = 304,

  CALL = 305, CANCEL = 306, CH = 307, CHAINING = 308,

  CHARACTER = 309, CHARACTERS = 310, CLASS = 311, CLOSE = 312,

  CODE = 313, CODE_SET = 314, COLLATING = 315, COL = 316,

  COLS = 317, COLUMN = 318, COLUMNS = 319, COMMA = 320,

  COMMAND_LINE = 321, COMMA_DELIM = 322, COMMIT = 323, COMMON = 324,

  COMP = 325, COMPUTE = 326, COMP_1 = 327, COMP_2 = 328,

  COMP_3 = 329, COMP_4 = 330, COMP_5 = 331, COMP_X = 332,

  CONCATENATE_FUNC = 333, CONFIGURATION = 334, CONSTANT = 335, CONTAINS = 336,

  CONTENT = 337, CONTINUE = 338, CONTROL = 339, CONTROLS = 340,

  CONTROL_FOOTING = 341, CONTROL_HEADING = 342, CONVERTING = 343, CORRESPONDING = 344,

  COUNT = 345, CRT = 346, CURRENCY = 347, CURRENT_DATE_FUNC = 348,

  CURSOR = 349, CYCLE = 350, DATA = 351, DATE = 352,

  DAY = 353, DAY_OF_WEEK = 354, DE = 355, DEBUGGING = 356,

  DECIMAL_POINT = 357, DECLARATIVES = 358, DEFAULT = 359, DELETE = 360,

DELIMITED = 361, DELIMITER = 362, DEPENDING = 363, DESCENDING = 364,

DETAIL = 365, DISK = 366, DISPLAY = 367, DIVIDE = 368,

DIVISION = 369, DOWN = 370, DUPLICATES = 371, DYNAMIC = 372,

EBCDIC = 373, ELSE = 374, END = 375, END_ACCEPT = 376,

END_ADD = 377, END_CALL = 378, END_COMPUTE = 379, END_DELETE = 380,

END_DISPLAY = 381, END_DIVIDE = 382, END_EVALUATE = 383, END_-FUNCTION = 384,

END_IF = 385, END_MULTIPLY = 386, END_PERFORM = 387, END_PROGRAM = 388,

END_READ = 389, END_RETURN = 390, END_REWRITE = 391, END_SEARCH = 392,

END_START = 393, END_STRING = 394, END_SUBTRACT = 395, END_UNSTRING = 396,

END_WRITE = 397, ENTRY = 398, ENVIRONMENT = 399, ENVIRONMENT_-NAME = 400,

ENVIRONMENT_VALUE = 401, EOL = 402, EOP = 403, EOS = 404,

EQUAL = 405, EQUALS = 406, ERASE = 407, ERROR = 408,

ESCAPE = 409, EVALUATE = 410, EVENT_STATUS = 411, EXCEPTION = 412,

EXCLUSIVE = 413, EXIT = 414, EXTEND = 415, EXTERNAL = 416,

FD = 417, FILE_CONTROL = 418, FILE_ID = 419, FILLER = 420,

FINAL = 421, FIRST = 422, FOOTING = 423, FOR = 424,

FOREGROUND_COLOR = 425, FOREVER = 426, FREE = 427, FROM = 428,

FULL = 429, FUNCTION = 430, FUNCTION_ID = 431, FUNCTION_NAME = 432,

GE = 433, GENERATE = 434, GIVING = 435, GLOBAL = 436,

GO = 437, GOBACK = 438, GREATER = 439, GROUP = 440,

HEADING = 441, HIGHLIGHT = 442, HIGH_VALUE = 443, IDENTIFICATION = 444,

IF = 445, IGNORE = 446, IGNORING = 447, IN = 448,

INDEX = 449, INDEXED = 450, INDICATE = 451, INITIALIZE = 452,

INITIALIZED = 453, INITIATE = 454, INPUT = 455, INPUT_OUTPUT = 456,

INSPECT = 457, INTO = 458, INTRINSIC = 459, INVALID = 460,

INVALID_KEY = 461, IS = 462, I_O = 463, I_O_CONTROL = 464,

JUSTIFIED = 465, KEY = 466, LABEL = 467, LAST = 468,

LAST_DETAIL = 469, LE = 470, LEADING = 471, LEFT = 472,

LENGTH = 473, LESS = 474, LIMIT = 475, LIMITS = 476,

LINAGE = 477, LINAGE_COUNTER = 478, LINE = 479, LINES = 480,

LINKAGE = 481, LITERAL = 482, LOCALE = 483, LOCALE_DT_FUNC = 484,

LOCAL_STORAGE = 485, LOCK = 486, LOWER_CASE_FUNC = 487, LOW-LIGHT = 488,

LOW_VALUE = 489, MANUAL = 490, MEMORY = 491, MERGE = 492,

MINUS = 493, MNEMONIC_NAME = 494, MODE = 495, MOVE = 496,

MULTIPLE = 497, MULTIPLY = 498, NATIONAL = 499, NATIONAL_EDITED = 500,

NATIVE = 501, NE = 502, NEGATIVE = 503, NEXT = 504,

NEXT_SENTENCE = 505, NO = 506, NOT = 507, NOT_END = 508,

NOT_EOP = 509, NOT_EXCEPTION = 510, NOT_INVALID_KEY = 511, NOT_-OVERFLOW = 512,

NOT_SIZE_ERROR = 513, NO_ADVANCING = 514, NUMBER = 515, NUM-BERS = 516,

NUMERIC = 517, NUMERIC_EDITED = 518, NUMVALC_FUNC = 519, OBJECT_-COMPUTER = 520,

OCCURS = 521, OF = 522, OFF = 523, OMITTED = 524,

ON = 525, ONLY = 526, OPEN = 527, OPTIONAL = 528,

OR = 529, ORDER = 530, ORGANIZATION = 531, OTHER = 532,

OUTPUT = 533, OVERFLOW = 534, OVERLINE = 535, PACKED_DECIMAL = 536,

PADDING = 537, PAGE = 538, PAGE_FOOTING = 539, PAGE_HEADING = 540,

PARAGRAPH = 541, PERFORM = 542, PICTURE = 543, PLUS = 544,

POINTER = 545, POSITION = 546, POSITIVE = 547, PRESENT = 548,

PREVIOUS = 549, PRINTER = 550, PRINTING = 551, PROCEDURE = 552,

PROCEDURES = 553, PROCEED = 554, PROGRAM = 555, PROGRAM_ID = 556,

PROGRAM_NAME = 557, PROGRAM_POINTER = 558, PROMPT = 559, QUOTE = 560,

RANDOM = 561, RD = 562, READ = 563, RECORD = 564,

RECORDING = 565, RECORDS = 566, RECURSIVE = 567, REDEFINES = 568,

REEL = 569, REFERENCE = 570, RELATIVE = 571, RELEASE = 572,

REMAINDER = 573, REMOVAL = 574, RENAMES = 575, REPLACING = 576,

REPORT = 577, REPORTING = 578, REPORTS = 579, REPORT_FOOTING = 580,

REPORT_HEADING = 581, REPOSITORY = 582, REQUIRED = 583, RESERVE = 584,

RETURN = 585, RETURNING = 586, REVERSE_FUNC = 587, REVERSE_-VIDEO = 588,

REWIND = 589, REWRITE = 590, RIGHT = 591, ROLLBACK = 592,

ROUNDED = 593, RUN = 594, SAME = 595, SCREEN = 596,

SCREEN_CONTROL = 597, SCROLL = 598, SD = 599, SEARCH = 600,

SECTION = 601, SECURE = 602, SEGMENT_LIMIT = 603, SELECT = 604,

SEMI_COLON = 605, SENTENCE = 606, SEPARATE = 607, SEQUENCE = 608,

SEQUENTIAL = 609, SET = 610, SHARING = 611, SIGN = 612,

SIGNED = 613, SIGNED_INT = 614, SIGNED_LONG = 615, SIGNED_SHORT = 616,

SIZE = 617, SIZE_ERROR = 618, SORT = 619, SORT_MERGE = 620,

SOURCE = 621, SOURCE_COMPUTER = 622, SPACE = 623, SPECIAL_NAMES = 624,

STANDARD = 625, STANDARD_1 = 626, STANDARD_2 = 627, START = 628,

STATUS = 629, STOP = 630, STRING = 631, SUBSTITUTE_FUNC = 632,

SUBSTITUTE_CASE_FUNC = 633, SUBTRACT = 634, SUM = 635, SUPPRESS = 636,

SYMBOLIC = 637, SYNCHRONIZED = 638, TALLYING = 639, TAPE = 640,

TERMINATE = 641, TEST = 642, THAN = 643, THEN = 644,

THRU = 645, TIME = 646, TIMES = 647, TO = 648,

TOK_FALSE = 649, TOK_FILE = 650, TOK_INITIAL = 651, TOK_NULL = 652,

TOK_TRUE = 653, TOP = 654, TRAILING = 655, TRANSFORM = 656,

TRIM_FUNCTION = 657, TYPE = 658, UNDERLINE = 659, UNIT = 660,

UNLOCK = 661, UNSIGNED = 662, UNSIGNED_INT = 663, UNSIGNED_LONG = 664,

UNSIGNED_SHORT = 665, UNSTRING = 666, UNTIL = 667, UP = 668,

UPDATE = 669, UPON = 670, UPON_ARGUMENT_NUMBER = 671, UPON_-COMMAND_LINE = 672,

UPON_ENVIRONMENT_NAME = 673, UPON_ENVIRONMENT_VALUE = 674, UPPER_CASE_FUNC = 675, USAGE = 676,

USE = 677, USING = 678, VALUE = 679, VARYING = 680,

WAIT = 681, WHEN = 682, WHEN_COMPILED_FUNC = 683, WITH = 684,

WORD = 685, WORDS = 686, WORKING_STORAGE = 687, WRITE = 688,

YYYYDDD = 689, YYYYMMDD = 690, ZERO = 691, UNARY_SIGN = 692,

TOKEN_EOF = 0, ACCEPT = 258, ACCESS = 259, ADD = 260,

ADDRESS = 261, ADVANCING = 262, AFTER = 263, ALL = 264,

ALLOCATE = 265, ALPHABET = 266, ALPHABETIC = 267, ALPHABETIC_-LOWER = 268,

ALPHABETIC_UPPER = 269, ALPHANUMERIC = 270, ALPHANUMERIC_EDITED = 271, ALSO = 272,

ALTER = 273, ALTERNATE = 274, AND = 275, ANY = 276,

ARE = 277, AREA = 278, ARGUMENT_NUMBER = 279, ARGUMENT_VALUE = 280,

AS = 281, ASCENDING = 282, ASSIGN = 283, AT = 284,

AUTO = 285, AUTOMATIC = 286, BACKGROUND_COLOR = 287, BASED = 288,

BEFORE = 289, BELL = 290, BINARY = 291, BINARY_C_LONG = 292,

BINARY_CHAR = 293, BINARY_DOUBLE = 294, BINARY_LONG = 295, BINARY_-
SHORT = 296,

BLANK = 297, BLANK_LINE = 298, BLANK_SCREEN = 299, BLINK = 300,

BLOCK = 301, BOTTOM = 302, BY = 303, BYTE_LENGTH = 304,

CALL = 305, CANCEL = 306, CH = 307, CHAINING = 308,

CHARACTER = 309, CHARACTERS = 310, CLASS = 311, CLOSE = 312,

CODE = 313, CODE_SET = 314, COLLATING = 315, COL = 316,

COLS = 317, COLUMN = 318, COLUMNS = 319, COMMA = 320,

COMMAND_LINE = 321, COMMA_DELIM = 322, COMMIT = 323, COMMON =
324,

COMP = 325, COMPUTE = 326, COMP_1 = 327, COMP_2 = 328,

COMP_3 = 329, COMP_4 = 330, COMP_5 = 331, COMP_X = 332,

CONCATENATE_FUNC = 333, CONFIGURATION = 334, CONSTANT = 335,
CONTAINS = 336,

CONTENT = 337, CONTINUE = 338, CONTROL = 339, CONTROLS = 340,

CONTROL_FOOTING = 341, CONTROL_HEADING = 342, CONVERTING =
343, CORRESPONDING = 344,

COUNT = 345, CRT = 346, CURRENCY = 347, CURRENT_DATE_FUNC = 348,

CURSOR = 349, CYCLE = 350, DATA = 351, DATE = 352,

DAY = 353, DAY_OF_WEEK = 354, DE = 355, DEBUGGING = 356,

DECIMAL_POINT = 357, DECLARATIVES = 358, DEFAULT = 359, DELETE =
360,

DELIMITED = 361, DELIMITER = 362, DEPENDING = 363, DESCENDING =
364,

DETAIL = 365, DISK = 366, DISPLAY = 367, DIVIDE = 368,

DIVISION = 369, DOWN = 370, DUPLICATES = 371, DYNAMIC = 372,

EBCDIC = 373, ELSE = 374, END = 375, END_ACCEPT = 376,

END_ADD = 377, END_CALL = 378, END_COMPUTE = 379, END_DELETE =
380,

END_DISPLAY = 381, END_DIVIDE = 382, END_EVALUATE = 383, END_-
FUNCTION = 384,

END_IF = 385, END_MULTIPLY = 386, END_PERFORM = 387, END_PROGRAM
= 388,

END_READ = 389, END_RETURN = 390, END_REWRITE = 391, END_SEARCH
= 392,

END_START = 393, END_STRING = 394, END_SUBTRACT = 395, END_UNSTRING
= 396,

END_WRITE = 397, ENTRY = 398, ENVIRONMENT = 399, ENVIRONMENT_-
NAME = 400,

ENVIRONMENT_VALUE = 401, EOL = 402, EOP = 403, EOS = 404,

EQUAL = 405, EQUALS = 406, ERASE = 407, ERROR = 408,

ESCAPE = 409, EVALUATE = 410, EVENT_STATUS = 411, EXCEPTION = 412,

EXCLUSIVE = 413, EXIT = 414, EXTEND = 415, EXTERNAL = 416,

FD = 417, FILE_CONTROL = 418, FILE_ID = 419, FILLER = 420,

FINAL = 421, FIRST = 422, FOOTING = 423, FOR = 424,

FOREGROUND_COLOR = 425, FOREVER = 426, FREE = 427, FROM = 428,

FULL = 429, FUNCTION = 430, FUNCTION_ID = 431, FUNCTION_NAME = 432,

GE = 433, GENERATE = 434, GIVING = 435, GLOBAL = 436,

GO = 437, GOBACK = 438, GREATER = 439, GROUP = 440,

HEADING = 441, HIGHLIGHT = 442, HIGH_VALUE = 443, IDENTIFICATION = 444,

IF = 445, IGNORE = 446, IGNORING = 447, IN = 448,

INDEX = 449, INDEXED = 450, INDICATE = 451, INITIALIZE = 452,

INITIALIZED = 453, INITIATE = 454, INPUT = 455, INPUT_OUTPUT = 456,

INSPECT = 457, INTO = 458, INTRINSIC = 459, INVALID = 460,

INVALID_KEY = 461, IS = 462, I_O = 463, I_O_CONTROL = 464,

JUSTIFIED = 465, KEY = 466, LABEL = 467, LAST = 468,

LAST_DETAIL = 469, LE = 470, LEADING = 471, LEFT = 472,

LENGTH = 473, LESS = 474, LIMIT = 475, LIMITS = 476,

LINAGE = 477, LINAGE_COUNTER = 478, LINE = 479, LINES = 480,

LINKAGE = 481, LITERAL = 482, LOCALE = 483, LOCALE_DT_FUNC = 484,

LOCAL_STORAGE = 485, LOCK = 486, LOWER_CASE_FUNC = 487, LOW-LIGHT = 488,

LOW_VALUE = 489, MANUAL = 490, MEMORY = 491, MERGE = 492,

MINUS = 493, MNEMONIC_NAME = 494, MODE = 495, MOVE = 496,

MULTIPLE = 497, MULTIPLY = 498, NATIONAL = 499, NATIONAL_EDITED = 500,

NATIVE = 501, NE = 502, NEGATIVE = 503, NEXT = 504,

NEXT_SENTENCE = 505, NO = 506, NOT = 507, NOT_END = 508,

NOT_EOP = 509, NOT_EXCEPTION = 510, NOT_INVALID_KEY = 511, NOT_-OVERFLOW = 512,

NOT_SIZE_ERROR = 513, NO_ADVANCING = 514, NUMBER = 515, NUM-BERS = 516,

NUMERIC = 517, NUMERIC_EDITED = 518, NUMVALC_FUNC = 519, OBJECT_-COMPUTER = 520,

OCCURS = 521, OF = 522, OFF = 523, OMITTED = 524,

ON = 525, ONLY = 526, OPEN = 527, OPTIONAL = 528,

OR = 529, ORDER = 530, ORGANIZATION = 531, OTHER = 532,

OUTPUT = 533, OVERFLOW = 534, OVERLINE = 535, PACKED_DECIMAL = 536,

PADDING = 537, PAGE = 538, PAGE_FOOTING = 539, PAGE_HEADING = 540,

PARAGRAPH = 541, PERFORM = 542, PICTURE = 543, PLUS = 544,

POINTER = 545, POSITION = 546, POSITIVE = 547, PRESENT = 548,

PREVIOUS = 549, PRINTER = 550, PRINTING = 551, PROCEDURE = 552,

PROCEDURES = 553, PROCEED = 554, PROGRAM = 555, PROGRAM_ID = 556,

PROGRAM_NAME = 557, PROGRAM_POINTER = 558, PROMPT = 559, QUOTE = 560,

RANDOM = 561, RD = 562, READ = 563, RECORD = 564,

RECORDING = 565, RECORDS = 566, RECURSIVE = 567, REDEFINES = 568,

REEL = 569, REFERENCE = 570, RELATIVE = 571, RELEASE = 572,

REMAINDER = 573, REMOVAL = 574, RENAMES = 575, REPLACING = 576,

REPORT = 577, REPORTING = 578, REPORTS = 579, REPORT_FOOTING = 580,

REPORT_HEADING = 581, REPOSITORY = 582, REQUIRED = 583, RESERVE = 584,

RETURN = 585, RETURNING = 586, REVERSE_FUNC = 587, REVERSE_-VIDEO = 588,

REWIND = 589, REWRITE = 590, RIGHT = 591, ROLLBACK = 592,

ROUNDED = 593, RUN = 594, SAME = 595, SCREEN = 596,

SCREEN_CONTROL = 597, SCROLL = 598, SD = 599, SEARCH = 600,

SECTION = 601, SECURE = 602, SEGMENT_LIMIT = 603, SELECT = 604,

SEMI_COLON = 605, SENTENCE = 606, SEPARATE = 607, SEQUENCE = 608,

SEQUENTIAL = 609, SET = 610, SHARING = 611, SIGN = 612,

SIGNED = 613, SIGNED_INT = 614, SIGNED_LONG = 615, SIGNED_SHORT = 616,

SIZE = 617, SIZE_ERROR = 618, SORT = 619, SORT_MERGE = 620,

SOURCE = 621, SOURCE_COMPUTER = 622, SPACE = 623, SPECIAL_NAMES = 624,

STANDARD = 625, STANDARD_1 = 626, STANDARD_2 = 627, START = 628,

STATUS = 629, STOP = 630, STRING = 631, SUBSTITUTE_FUNC = 632,

SUBSTITUTE_CASE_FUNC = 633, SUBTRACT = 634, SUM = 635, SUPPRESS = 636,

SYMBOLIC = 637, SYNCHRONIZED = 638, TALLYING = 639, TAPE = 640,

TERMINATE = 641, TEST = 642, THAN = 643, THEN = 644,

THRU = 645, TIME = 646, TIMES = 647, TO = 648,

TOK_FALSE = 649, TOK_FILE = 650, TOK_INITIAL = 651, TOK_NULL = 652,

TOK_TRUE = 653, TOP = 654, TRAILING = 655, TRANSFORM = 656,

TRIM_FUNCTION = 657, TYPE = 658, UNDERLINE = 659, UNIT = 660,

UNLOCK = 661, UNSIGNED = 662, UNSIGNED_INT = 663, UNSIGNED_LONG = 664,

UNSIGNED_SHORT = 665, UNSTRING = 666, UNTIL = 667, UP = 668,

UPDATE = 669, UPON = 670, UPON_ARGUMENT_NUMBER = 671, UPON_COMMAND_LINE = 672,

UPON_ENVIRONMENT_NAME = 673, UPON_ENVIRONMENT_VALUE = 674, UPPER_CASE_FUNC = 675, USAGE = 676,

USE = 677, USING = 678, VALUE = 679, VARYING = 680,

WAIT = 681, WHEN = 682, WHEN_COMPILED_FUNC = 683, WITH = 684,

WORD = 685, WORDS = 686, WORKING_STORAGE = 687, WRITE = 688,

YYYYDDD = 689, YYYYMMDD = 690, ZERO = 691, UNARY_SIGN = 692,

TOKEN_EOF = 0, COPY = 258, REPLACE = 259, SUPPRESS = 260,

PRINTING = 261, REPLACING = 262, OFF = 263, IN = 264,

OF = 265, BY = 266, EQEQ = 267, TOKEN = 268,

TOKEN_EOF = 0, COPY = 258, REPLACE = 259, SUPPRESS = 260,

PRINTING = 261, REPLACING = 262, OFF = 263, IN = 264,

OF = 265, BY = 266, EQEQ = 267, TOKEN = 268 }

Variables

- YYSTYPE yylval

7.9.1 Define Documentation

7.9.1.1 #define ACCEPT 258

Definition at line 471 of file parser.h.

7.9.1.2 #define ACCESS 259

Definition at line 472 of file parser.h.

7.9.1.3 #define ADD 260

Definition at line 473 of file parser.h.

7.9.1.4 #define ADDRESS 261

Definition at line 474 of file parser.h.

7.9.1.5 #define ADVANCING 262

Definition at line 475 of file parser.h.

7.9.1.6 #define AFTER 263

Definition at line 476 of file parser.h.

7.9.1.7 #define ALL 264

Definition at line 477 of file parser.h.

7.9.1.8 #define ALLOCATE 265

Definition at line 478 of file parser.h.

7.9.1.9 #define ALPHABET 266

Definition at line 479 of file parser.h.

7.9.1.10 #define ALPHABETIC 267

Definition at line 480 of file parser.h.

7.9.1.11 #define ALPHABETIC_LOWER 268

Definition at line 481 of file parser.h.

7.9.1.12 #define ALPHABETIC_UPPER 269

Definition at line 482 of file parser.h.

7.9.1.13 #define ALPHANUMERIC 270

Definition at line 483 of file parser.h.

7.9.1.14 #define ALPHANUMERIC_EDITED 271

Definition at line 484 of file parser.h.

7.9.1.15 #define ALSO 272

Definition at line 485 of file parser.h.

7.9.1.16 #define ALTER 273

Definition at line 486 of file parser.h.

7.9.1.17 #define ALTERNATE 274

Definition at line 487 of file parser.h.

7.9.1.18 #define AND 275

Definition at line 488 of file parser.h.

7.9.1.19 #define ANY 276

Definition at line 489 of file parser.h.

7.9.1.20 #define ARE 277

Definition at line 490 of file parser.h.

7.9.1.21 #define AREA 278

Definition at line 491 of file parser.h.

7.9.1.22 #define ARGUMENT_NUMBER 279

Definition at line 492 of file parser.h.

7.9.1.23 #define ARGUMENT_VALUE 280

Definition at line 493 of file parser.h.

7.9.1.24 #define AS 281

Definition at line 494 of file parser.h.

7.9.1.25 #define ASCENDING 282

Definition at line 495 of file parser.h.

7.9.1.26 #define ASSIGN 283

Definition at line 496 of file parser.h.

7.9.1.27 #define AT 284

Definition at line 497 of file parser.h.

7.9.1.28 #define AUTO 285

Definition at line 498 of file parser.h.

7.9.1.29 #define AUTOMATIC 286

Definition at line 499 of file parser.h.

7.9.1.30 #define BACKGROUND_COLOR 287

Definition at line 500 of file parser.h.

7.9.1.31 #define BASED 288

Definition at line 501 of file parser.h.

7.9.1.32 #define BEFORE 289

Definition at line 502 of file parser.h.

7.9.1.33 #define BELL 290

Definition at line 503 of file parser.h.

7.9.1.34 #define BINARY 291

Definition at line 504 of file parser.h.

7.9.1.35 #define BINARY_C_LONG 292

Definition at line 505 of file parser.h.

7.9.1.36 #define BINARY_CHAR 293

Definition at line 506 of file parser.h.

7.9.1.37 #define BINARY_DOUBLE 294

Definition at line 507 of file parser.h.

7.9.1.38 #define BINARY_LONG 295

Definition at line 508 of file parser.h.

7.9.1.39 #define BINARY_SHORT 296

Definition at line 509 of file parser.h.

7.9.1.40 #define BLANK 297

Definition at line 510 of file parser.h.

7.9.1.41 #define BLANK_LINE 298

Definition at line 511 of file parser.h.

7.9.1.42 #define BLANK_SCREEN 299

Definition at line 512 of file parser.h.

7.9.1.43 #define BLINK 300

Definition at line 513 of file parser.h.

7.9.1.44 #define BLOCK 301

Definition at line 514 of file parser.h.

7.9.1.45 #define BOTTOM 302

Definition at line 515 of file parser.h.

7.9.1.46 #define BY 303

Definition at line 516 of file parser.h.

7.9.1.47 #define BYTE_LENGTH 304

Definition at line 517 of file parser.h.

7.9.1.48 #define CALL 305

Definition at line 518 of file parser.h.

7.9.1.49 #define CANCEL 306

Definition at line 519 of file parser.h.

7.9.1.50 #define CH 307

Definition at line 520 of file parser.h.

7.9.1.51 #define CHAINING 308

Definition at line 521 of file parser.h.

7.9.1.52 #define CHARACTER 309

Definition at line 522 of file parser.h.

7.9.1.53 #define CHARACTERS 310

Definition at line 523 of file parser.h.

7.9.1.54 #define CLASS 311

Definition at line 524 of file parser.h.

7.9.1.55 #define CLOSE 312

Definition at line 525 of file parser.h.

7.9.1.56 #define CODE 313

Definition at line 526 of file parser.h.

7.9.1.57 #define CODE_SET 314

Definition at line 527 of file parser.h.

7.9.1.58 #define COL 316

Definition at line 529 of file parser.h.

7.9.1.59 #define COLLATING 315

Definition at line 528 of file parser.h.

7.9.1.60 #define COLS 317

Definition at line 530 of file parser.h.

7.9.1.61 #define COLUMN 318

Definition at line 531 of file parser.h.

7.9.1.62 #define COLUMNS 319

Definition at line 532 of file parser.h.

7.9.1.63 #define COMMA 320

Definition at line 533 of file parser.h.

7.9.1.64 #define COMMA_DELIM 322

Definition at line 535 of file parser.h.

7.9.1.65 #define COMMAND_LINE 321

Definition at line 534 of file parser.h.

7.9.1.66 #define COMMIT 323

Definition at line 536 of file parser.h.

7.9.1.67 #define COMMON 324

Definition at line 537 of file parser.h.

7.9.1.68 #define COMP 325

Definition at line 538 of file parser.h.

7.9.1.69 #define COMP_1 327

Definition at line 540 of file parser.h.

7.9.1.70 #define COMP_2 328

Definition at line 541 of file parser.h.

7.9.1.71 #define COMP_3 329

Definition at line 542 of file parser.h.

7.9.1.72 #define COMP_4 330

Definition at line 543 of file parser.h.

7.9.1.73 #define COMP_5 331

Definition at line 544 of file parser.h.

7.9.1.74 #define COMP_X 332

Definition at line 545 of file parser.h.

7.9.1.75 #define COMPUTE 326

Definition at line 539 of file parser.h.

7.9.1.76 #define CONCATENATE_FUNC 333

Definition at line 546 of file parser.h.

7.9.1.77 #define CONFIGURATION 334

Definition at line 547 of file parser.h.

7.9.1.78 #define CONSTANT 335

Definition at line 548 of file parser.h.

7.9.1.79 #define CONTAINS 336

Definition at line 549 of file parser.h.

7.9.1.80 #define CONTENT 337

Definition at line 550 of file parser.h.

7.9.1.81 #define CONTINUE 338

Definition at line 551 of file parser.h.

7.9.1.82 #define CONTROL 339

Definition at line 552 of file parser.h.

7.9.1.83 #define CONTROL_FOOTING 341

Definition at line 554 of file parser.h.

7.9.1.84 #define CONTROL_HEADING 342

Definition at line 555 of file parser.h.

7.9.1.85 #define CONTROLS 340

Definition at line 553 of file parser.h.

7.9.1.86 #define CONVERTING 343

Definition at line 556 of file parser.h.

7.9.1.87 #define CORRESPONDING 344

Definition at line 557 of file parser.h.

7.9.1.88 #define COUNT 345

Definition at line 558 of file parser.h.

7.9.1.89 #define CRT 346

Definition at line 559 of file parser.h.

7.9.1.90 #define CURRENCY 347

Definition at line 560 of file parser.h.

7.9.1.91 #define CURRENT_DATE_FUNC 348

Definition at line 561 of file parser.h.

7.9.1.92 #define CURSOR 349

Definition at line 562 of file parser.h.

7.9.1.93 #define CYCLE 350

Definition at line 563 of file parser.h.

7.9.1.94 #define DATA 351

Definition at line 564 of file parser.h.

7.9.1.95 #define DATE 352

Definition at line 565 of file parser.h.

7.9.1.96 #define DAY 353

Definition at line 566 of file parser.h.

7.9.1.97 #define DAY_OF_WEEK 354

Definition at line 567 of file parser.h.

7.9.1.98 #define DE 355

Definition at line 568 of file parser.h.

7.9.1.99 #define DEBUGGING 356

Definition at line 569 of file parser.h.

7.9.1.100 #define DECIMAL_POINT 357

Definition at line 570 of file parser.h.

7.9.1.101 #define DECLARATIVES 358

Definition at line 571 of file parser.h.

7.9.1.102 #define DEFAULT 359

Definition at line 572 of file parser.h.

7.9.1.103 #define DELETE 360

Definition at line 573 of file parser.h.

7.9.1.104 #define DELIMITED 361

Definition at line 574 of file parser.h.

7.9.1.105 #define DELIMITER 362

Definition at line 575 of file parser.h.

7.9.1.106 #define DEPENDING 363

Definition at line 576 of file parser.h.

7.9.1.107 #define DESCENDING 364

Definition at line 577 of file parser.h.

7.9.1.108 #define DETAIL 365

Definition at line 578 of file parser.h.

7.9.1.109 #define DISK 366

Definition at line 579 of file parser.h.

7.9.1.110 #define DISPLAY 367

Definition at line 580 of file parser.h.

7.9.1.111 #define DIVIDE 368

Definition at line 581 of file parser.h.

7.9.1.112 #define DIVISION 369

Definition at line 582 of file parser.h.

7.9.1.113 #define DOWN 370

Definition at line 583 of file parser.h.

7.9.1.114 #define DUPLICATES 371

Definition at line 584 of file parser.h.

7.9.1.115 #define DYNAMIC 372

Definition at line 585 of file parser.h.

7.9.1.116 #define EBCDIC 373

Definition at line 586 of file parser.h.

7.9.1.117 #define ELSE 374

Definition at line 587 of file parser.h.

7.9.1.118 #define END 375

Definition at line 588 of file parser.h.

7.9.1.119 #define END_ACCEPT 376

Definition at line 589 of file parser.h.

7.9.1.120 #define END_ADD 377

Definition at line 590 of file parser.h.

7.9.1.121 #define END_CALL 378

Definition at line 591 of file parser.h.

7.9.1.122 #define END_COMPUTE 379

Definition at line 592 of file parser.h.

7.9.1.123 #define END_DELETE 380

Definition at line 593 of file parser.h.

7.9.1.124 #define END_DISPLAY 381

Definition at line 594 of file parser.h.

7.9.1.125 #define END_DIVIDE 382

Definition at line 595 of file parser.h.

7.9.1.126 #define END_EVALUATE 383

Definition at line 596 of file parser.h.

7.9.1.127 #define END_FUNCTION 384

Definition at line 597 of file parser.h.

7.9.1.128 #define END_IF 385

Definition at line 598 of file parser.h.

7.9.1.129 #define END_MULTIPLY 386

Definition at line 599 of file parser.h.

7.9.1.130 #define END_PERFORM 387

Definition at line 600 of file parser.h.

7.9.1.131 #define END_PROGRAM 388

Definition at line 601 of file parser.h.

7.9.1.132 #define END_READ 389

Definition at line 602 of file parser.h.

7.9.1.133 #define END_RETURN 390

Definition at line 603 of file parser.h.

7.9.1.134 #define END_REWRITE 391

Definition at line 604 of file parser.h.

7.9.1.135 #define END_SEARCH 392

Definition at line 605 of file parser.h.

7.9.1.136 #define END_START 393

Definition at line 606 of file parser.h.

7.9.1.137 #define END_STRING 394

Definition at line 607 of file parser.h.

7.9.1.138 #define END_SUBTRACT 395

Definition at line 608 of file parser.h.

7.9.1.139 #define END_UNSTRING 396

Definition at line 609 of file parser.h.

7.9.1.140 #define END_WRITE 397

Definition at line 610 of file parser.h.

7.9.1.141 #define ENTRY 398

Definition at line 611 of file parser.h.

7.9.1.142 #define ENVIRONMENT 399

Definition at line 612 of file parser.h.

7.9.1.143 #define ENVIRONMENT_NAME 400

Definition at line 613 of file parser.h.

7.9.1.144 #define ENVIRONMENT_VALUE 401

Definition at line 614 of file parser.h.

7.9.1.145 #define EOL 402

Definition at line 615 of file parser.h.

7.9.1.146 #define EOP 403

Definition at line 616 of file parser.h.

7.9.1.147 #define EOS 404

Definition at line 617 of file parser.h.

7.9.1.148 #define EQUAL 405

Definition at line 618 of file parser.h.

7.9.1.149 #define EQUALS 406

Definition at line 619 of file parser.h.

7.9.1.150 #define ERASE 407

Definition at line 620 of file parser.h.

7.9.1.151 #define ERROR 408

Definition at line 621 of file parser.h.

7.9.1.152 #define ESCAPE 409

Definition at line 622 of file parser.h.

7.9.1.153 #define EVALUATE 410

Definition at line 623 of file parser.h.

7.9.1.154 #define EVENT_STATUS 411

Definition at line 624 of file parser.h.

7.9.1.155 #define EXCEPTION 412

Definition at line 625 of file parser.h.

7.9.1.156 #define EXCLUSIVE 413

Definition at line 626 of file parser.h.

7.9.1.157 #define EXIT 414

Definition at line 627 of file parser.h.

7.9.1.158 #define EXTEND 415

Definition at line 628 of file parser.h.

7.9.1.159 #define EXTERNAL 416

Definition at line 629 of file parser.h.

7.9.1.160 #define FD 417

Definition at line 630 of file parser.h.

7.9.1.161 #define FILE_CONTROL 418

Definition at line 631 of file parser.h.

7.9.1.162 #define FILE_ID 419

Definition at line 632 of file parser.h.

7.9.1.163 #define FILLER 420

Definition at line 633 of file parser.h.

7.9.1.164 #define FINAL 421

Definition at line 634 of file parser.h.

7.9.1.165 #define FIRST 422

Definition at line 635 of file parser.h.

7.9.1.166 #define FOOTING 423

Definition at line 636 of file parser.h.

7.9.1.167 #define FOR 424

Definition at line 637 of file parser.h.

7.9.1.168 #define FOREGROUND_COLOR 425

Definition at line 638 of file parser.h.

7.9.1.169 #define FOREVER 426

Definition at line 639 of file parser.h.

7.9.1.170 #define FREE 427

Definition at line 640 of file parser.h.

7.9.1.171 #define FROM 428

Definition at line 641 of file parser.h.

7.9.1.172 #define FULL 429

Definition at line 642 of file parser.h.

7.9.1.173 #define FUNCTION 430

Definition at line 643 of file parser.h.

7.9.1.174 #define FUNCTION_ID 431

Definition at line 644 of file parser.h.

7.9.1.175 #define FUNCTION_NAME 432

Definition at line 645 of file parser.h.

7.9.1.176 #define GE 433

Definition at line 646 of file parser.h.

7.9.1.177 #define GENERATE 434

Definition at line 647 of file parser.h.

7.9.1.178 #define GIVING 435

Definition at line 648 of file parser.h.

7.9.1.179 #define GLOBAL 436

Definition at line 649 of file parser.h.

7.9.1.180 #define GO 437

Definition at line 650 of file parser.h.

7.9.1.181 #define GOBACK 438

Definition at line 651 of file parser.h.

7.9.1.182 #define GREATER 439

Definition at line 652 of file parser.h.

7.9.1.183 #define GROUP 440

Definition at line 653 of file parser.h.

7.9.1.184 #define HEADING 441

Definition at line 654 of file parser.h.

7.9.1.185 #define HIGH_VALUE 443

Definition at line 656 of file parser.h.

7.9.1.186 #define HIGHLIGHT 442

Definition at line 655 of file parser.h.

7.9.1.187 #define I_O 463

Definition at line 676 of file parser.h.

7.9.1.188 #define I_O_CONTROL 464

Definition at line 677 of file parser.h.

7.9.1.189 #define IDENTIFICATION 444

Definition at line 657 of file parser.h.

7.9.1.190 #define IF 445

Definition at line 658 of file parser.h.

7.9.1.191 #define IGNORE 446

Definition at line 659 of file parser.h.

7.9.1.192 #define IGNORING 447

Definition at line 660 of file parser.h.

7.9.1.193 #define IN 448

Definition at line 661 of file parser.h.

7.9.1.194 #define INDEX 449

Definition at line 662 of file parser.h.

7.9.1.195 #define INDEXED 450

Definition at line 663 of file parser.h.

7.9.1.196 #define INDICATE 451

Definition at line 664 of file parser.h.

7.9.1.197 #define INITIALIZE 452

Definition at line 665 of file parser.h.

7.9.1.198 #define INITIALIZED 453

Definition at line 666 of file parser.h.

7.9.1.199 #define INITIATE 454

Definition at line 667 of file parser.h.

7.9.1.200 #define INPUT 455

Definition at line 668 of file parser.h.

7.9.1.201 #define INPUT_OUTPUT 456

Definition at line 669 of file parser.h.

7.9.1.202 #define INSPECT 457

Definition at line 670 of file parser.h.

7.9.1.203 #define INTO 458

Definition at line 671 of file parser.h.

7.9.1.204 #define INTRINSIC 459

Definition at line 672 of file parser.h.

7.9.1.205 #define INVALID 460

Definition at line 673 of file parser.h.

7.9.1.206 #define INVALID_KEY 461

Definition at line 674 of file parser.h.

7.9.1.207 #define IS 462

Definition at line 675 of file parser.h.

7.9.1.208 #define JUSTIFIED 465

Definition at line 678 of file parser.h.

7.9.1.209 #define KEY 466

Definition at line 679 of file parser.h.

7.9.1.210 #define LABEL 467

Definition at line 680 of file parser.h.

7.9.1.211 #define LAST 468

Definition at line 681 of file parser.h.

7.9.1.212 #define LAST_DETAIL 469

Definition at line 682 of file parser.h.

7.9.1.213 #define LE 470

Definition at line 683 of file parser.h.

7.9.1.214 #define LEADING 471

Definition at line 684 of file parser.h.

7.9.1.215 #define LEFT 472

Definition at line 685 of file parser.h.

7.9.1.216 #define LENGTH 473

Definition at line 686 of file parser.h.

7.9.1.217 #define LESS 474

Definition at line 687 of file parser.h.

7.9.1.218 #define LIMIT 475

Definition at line 688 of file parser.h.

7.9.1.219 #define LIMITS 476

Definition at line 689 of file parser.h.

7.9.1.220 #define LINAGE 477

Definition at line 690 of file parser.h.

7.9.1.221 #define LINAGE_COUNTER 478

Definition at line 691 of file parser.h.

7.9.1.222 #define LINE 479

Definition at line 692 of file parser.h.

7.9.1.223 #define LINES 480

Definition at line 693 of file parser.h.

7.9.1.224 #define LINKAGE 481

Definition at line 694 of file parser.h.

7.9.1.225 #define LITERAL 482

Definition at line 695 of file parser.h.

7.9.1.226 #define LOCAL_STORAGE 485

Definition at line 698 of file parser.h.

7.9.1.227 #define LOCALE 483

Definition at line 696 of file parser.h.

7.9.1.228 #define LOCALE_DT_FUNC 484

Definition at line 697 of file parser.h.

7.9.1.229 #define LOCK 486

Definition at line 699 of file parser.h.

7.9.1.230 #define LOW_VALUE 489

Definition at line 702 of file parser.h.

7.9.1.231 #define LOWER_CASE_FUNC 487

Definition at line 700 of file parser.h.

7.9.1.232 #define LOWLIGHT 488

Definition at line 701 of file parser.h.

7.9.1.233 #define MANUAL 490

Definition at line 703 of file parser.h.

7.9.1.234 #define MEMORY 491

Definition at line 704 of file parser.h.

7.9.1.235 #define MERGE 492

Definition at line 705 of file parser.h.

7.9.1.236 #define MINUS 493

Definition at line 706 of file parser.h.

7.9.1.237 #define MNEMONIC_NAME 494

Definition at line 707 of file parser.h.

7.9.1.238 #define MODE 495

Definition at line 708 of file parser.h.

7.9.1.239 #define MOVE 496

Definition at line 709 of file parser.h.

7.9.1.240 #define MULTIPLE 497

Definition at line 710 of file parser.h.

7.9.1.241 #define MULTIPLY 498

Definition at line 711 of file parser.h.

7.9.1.242 #define NATIONAL 499

Definition at line 712 of file parser.h.

7.9.1.243 #define NATIONAL_EDITED 500

Definition at line 713 of file parser.h.

7.9.1.244 #define NATIVE 501

Definition at line 714 of file parser.h.

7.9.1.245 #define NE 502

Definition at line 715 of file parser.h.

7.9.1.246 #define NEGATIVE 503

Definition at line 716 of file parser.h.

7.9.1.247 #define NEXT 504

Definition at line 717 of file parser.h.

7.9.1.248 #define NEXT_SENTENCE 505

Definition at line 718 of file parser.h.

7.9.1.249 #define NO 506

Definition at line 719 of file parser.h.

7.9.1.250 #define NO_ADVANCING 514

Definition at line 727 of file parser.h.

7.9.1.251 #define NOT 507

Definition at line 720 of file parser.h.

7.9.1.252 #define NOT_END 508

Definition at line 721 of file parser.h.

7.9.1.253 #define NOT_EOP 509

Definition at line 722 of file parser.h.

7.9.1.254 #define NOT_EXCEPTION 510

Definition at line 723 of file parser.h.

7.9.1.255 #define NOT_INVALID_KEY 511

Definition at line 724 of file parser.h.

7.9.1.256 #define NOT_OVERFLOW 512

Definition at line 725 of file parser.h.

7.9.1.257 #define NOT_SIZE_ERROR 513

Definition at line 726 of file parser.h.

7.9.1.258 #define NUMBER 515

Definition at line 728 of file parser.h.

7.9.1.259 #define NUMBERS 516

Definition at line 729 of file parser.h.

7.9.1.260 #define NUMERIC 517

Definition at line 730 of file parser.h.

7.9.1.261 #define NUMERIC_EDITED 518

Definition at line 731 of file parser.h.

7.9.1.262 #define NUMVALC_FUNC 519

Definition at line 732 of file parser.h.

7.9.1.263 #define OBJECT_COMPUTER 520

Definition at line 733 of file parser.h.

7.9.1.264 #define OCCURS 521

Definition at line 734 of file parser.h.

7.9.1.265 #define OF 522

Definition at line 735 of file parser.h.

7.9.1.266 #define OFF 523

Definition at line 736 of file parser.h.

7.9.1.267 #define OMITTED 524

Definition at line 737 of file parser.h.

7.9.1.268 #define ON 525

Definition at line 738 of file parser.h.

7.9.1.269 #define ONLY 526

Definition at line 739 of file parser.h.

7.9.1.270 #define OPEN 527

Definition at line 740 of file parser.h.

7.9.1.271 #define OPTIONAL 528

Definition at line 741 of file parser.h.

7.9.1.272 #define OR 529

Definition at line 742 of file parser.h.

7.9.1.273 #define ORDER 530

Definition at line 743 of file parser.h.

7.9.1.274 #define ORGANIZATION 531

Definition at line 744 of file parser.h.

7.9.1.275 #define OTHER 532

Definition at line 745 of file parser.h.

7.9.1.276 #define OUTPUT 533

Definition at line 746 of file parser.h.

7.9.1.277 #define OVERFLOW 534

Definition at line 747 of file parser.h.

7.9.1.278 #define OVERLINE 535

Definition at line 748 of file parser.h.

7.9.1.279 #define PACKED_DECIMAL 536

Definition at line 749 of file parser.h.

7.9.1.280 #define PADDING 537

Definition at line 750 of file parser.h.

7.9.1.281 #define PAGE 538

Definition at line 751 of file parser.h.

7.9.1.282 #define PAGE_FOOTING 539

Definition at line 752 of file parser.h.

7.9.1.283 #define PAGE_HEADING 540

Definition at line 753 of file parser.h.

7.9.1.284 #define PARAGRAPH 541

Definition at line 754 of file parser.h.

7.9.1.285 #define PERFORM 542

Definition at line 755 of file parser.h.

7.9.1.286 #define PICTURE 543

Definition at line 756 of file parser.h.

7.9.1.287 #define PLUS 544

Definition at line 757 of file parser.h.

7.9.1.288 #define POINTER 545

Definition at line 758 of file parser.h.

7.9.1.289 #define POSITION 546

Definition at line 759 of file parser.h.

7.9.1.290 #define POSITIVE 547

Definition at line 760 of file parser.h.

7.9.1.291 #define PRESENT 548

Definition at line 761 of file parser.h.

7.9.1.292 #define PREVIOUS 549

Definition at line 762 of file parser.h.

7.9.1.293 #define PRINTER 550

Definition at line 763 of file parser.h.

7.9.1.294 #define PRINTING 551

Definition at line 764 of file parser.h.

7.9.1.295 #define PROCEDURE 552

Definition at line 765 of file parser.h.

7.9.1.296 #define PROCEDURES 553

Definition at line 766 of file parser.h.

7.9.1.297 #define PROCEED 554

Definition at line 767 of file parser.h.

7.9.1.298 #define PROGRAM 555

Definition at line 768 of file parser.h.

7.9.1.299 #define PROGRAM_ID 556

Definition at line 769 of file parser.h.

7.9.1.300 #define PROGRAM_NAME 557

Definition at line 770 of file parser.h.

7.9.1.301 #define PROGRAM_POINTER 558

Definition at line 771 of file parser.h.

7.9.1.302 #define PROMPT 559

Definition at line 772 of file parser.h.

7.9.1.303 #define QUOTE 560

Definition at line 773 of file parser.h.

7.9.1.304 #define RANDOM 561

Definition at line 774 of file parser.h.

7.9.1.305 #define RD 562

Definition at line 775 of file parser.h.

7.9.1.306 #define READ 563

Definition at line 776 of file parser.h.

7.9.1.307 #define RECORD 564

Definition at line 777 of file parser.h.

7.9.1.308 #define RECORDING 565

Definition at line 778 of file parser.h.

7.9.1.309 #define RECORDS 566

Definition at line 779 of file parser.h.

7.9.1.310 #define RECURSIVE 567

Definition at line 780 of file parser.h.

7.9.1.311 #define REDEFINES 568

Definition at line 781 of file parser.h.

7.9.1.312 #define REEL 569

Definition at line 782 of file parser.h.

7.9.1.313 #define REFERENCE 570

Definition at line 783 of file parser.h.

7.9.1.314 #define RELATIVE 571

Definition at line 784 of file parser.h.

7.9.1.315 #define RELEASE 572

Definition at line 785 of file parser.h.

7.9.1.316 #define REMAINDER 573

Definition at line 786 of file parser.h.

7.9.1.317 #define REMOVAL 574

Definition at line 787 of file parser.h.

7.9.1.318 #define RENAMES 575

Definition at line 788 of file parser.h.

7.9.1.319 #define REPLACING 576

Definition at line 789 of file parser.h.

7.9.1.320 #define REPORT 577

Definition at line 790 of file parser.h.

7.9.1.321 #define REPORT_FOOTING 580

Definition at line 793 of file parser.h.

7.9.1.322 #define REPORT_HEADING 581

Definition at line 794 of file parser.h.

7.9.1.323 #define REPORTING 578

Definition at line 791 of file parser.h.

7.9.1.324 #define REPORTS 579

Definition at line 792 of file parser.h.

7.9.1.325 #define REPOSITORY 582

Definition at line 795 of file parser.h.

7.9.1.326 #define REQUIRED 583

Definition at line 796 of file parser.h.

7.9.1.327 #define RESERVE 584

Definition at line 797 of file parser.h.

7.9.1.328 #define RETURN 585

Definition at line 798 of file parser.h.

7.9.1.329 #define RETURNING 586

Definition at line 799 of file parser.h.

7.9.1.330 #define REVERSE_FUNC 587

Definition at line 800 of file parser.h.

7.9.1.331 #define REVERSE_VIDEO 588

Definition at line 801 of file parser.h.

7.9.1.332 #define REWIND 589

Definition at line 802 of file parser.h.

7.9.1.333 #define REWRITE 590

Definition at line 803 of file parser.h.

7.9.1.334 #define RIGHT 591

Definition at line 804 of file parser.h.

7.9.1.335 #define ROLLBACK 592

Definition at line 805 of file parser.h.

7.9.1.336 #define ROUNDED 593

Definition at line 806 of file parser.h.

7.9.1.337 #define RUN 594

Definition at line 807 of file parser.h.

7.9.1.338 #define SAME 595

Definition at line 808 of file parser.h.

7.9.1.339 #define SCREEN 596

Definition at line 809 of file parser.h.

7.9.1.340 #define SCREEN_CONTROL 597

Definition at line 810 of file parser.h.

7.9.1.341 #define SCROLL 598

Definition at line 811 of file parser.h.

7.9.1.342 #define SD 599

Definition at line 812 of file parser.h.

7.9.1.343 #define SEARCH 600

Definition at line 813 of file parser.h.

7.9.1.344 #define SECTION 601

Definition at line 814 of file parser.h.

7.9.1.345 #define SECURE 602

Definition at line 815 of file parser.h.

7.9.1.346 #define SEGMENT_LIMIT 603

Definition at line 816 of file parser.h.

7.9.1.347 #define SELECT 604

Definition at line 817 of file parser.h.

7.9.1.348 #define SEMI_COLON 605

Definition at line 818 of file parser.h.

7.9.1.349 #define SENTENCE 606

Definition at line 819 of file parser.h.

7.9.1.350 #define SEPARATE 607

Definition at line 820 of file parser.h.

7.9.1.351 #define SEQUENCE 608

Definition at line 821 of file parser.h.

7.9.1.352 #define SEQUENTIAL 609

Definition at line 822 of file parser.h.

7.9.1.353 #define SET 610

Definition at line 823 of file parser.h.

7.9.1.354 #define SHARING 611

Definition at line 824 of file parser.h.

7.9.1.355 #define SIGN 612

Definition at line 825 of file parser.h.

7.9.1.356 #define SIGNED 613

Definition at line 826 of file parser.h.

7.9.1.357 #define SIGNED_INT 614

Definition at line 827 of file parser.h.

7.9.1.358 #define SIGNED_LONG 615

Definition at line 828 of file parser.h.

7.9.1.359 #define SIGNED_SHORT 616

Definition at line 829 of file parser.h.

7.9.1.360 #define SIZE 617

Definition at line 830 of file parser.h.

7.9.1.361 #define SIZE_ERROR 618

Definition at line 831 of file parser.h.

7.9.1.362 #define SORT 619

Definition at line 832 of file parser.h.

7.9.1.363 #define SORT_MERGE 620

Definition at line 833 of file parser.h.

7.9.1.364 #define SOURCE 621

Definition at line 834 of file parser.h.

7.9.1.365 #define SOURCE_COMPUTER 622

Definition at line 835 of file parser.h.

7.9.1.366 #define SPACE 623

Definition at line 836 of file parser.h.

7.9.1.367 #define SPECIAL_NAMES 624

Definition at line 837 of file parser.h.

7.9.1.368 #define STANDARD 625

Definition at line 838 of file parser.h.

7.9.1.369 #define STANDARD_1 626

Definition at line 839 of file parser.h.

7.9.1.370 #define STANDARD_2 627

Definition at line 840 of file parser.h.

7.9.1.371 #define START 628

Definition at line 841 of file parser.h.

7.9.1.372 #define STATUS 629

Definition at line 842 of file parser.h.

7.9.1.373 #define STOP 630

Definition at line 843 of file parser.h.

7.9.1.374 #define STRING 631

Definition at line 844 of file parser.h.

7.9.1.375 #define SUBSTITUTE_CASE_FUNC 633

Definition at line 846 of file parser.h.

7.9.1.376 #define SUBSTITUTE_FUNC 632

Definition at line 845 of file parser.h.

7.9.1.377 #define SUBTRACT 634

Definition at line 847 of file parser.h.

7.9.1.378 #define SUM 635

Definition at line 848 of file parser.h.

7.9.1.379 #define SUPPRESS 636

Definition at line 849 of file parser.h.

7.9.1.380 #define SYMBOLIC 637

Definition at line 850 of file parser.h.

7.9.1.381 #define SYNCHRONIZED 638

Definition at line 851 of file parser.h.

7.9.1.382 #define TALLYING 639

Definition at line 852 of file parser.h.

7.9.1.383 #define TAPE 640

Definition at line 853 of file parser.h.

7.9.1.384 #define TERMINATE 641

Definition at line 854 of file parser.h.

7.9.1.385 #define TEST 642

Definition at line 855 of file parser.h.

7.9.1.386 #define THAN 643

Definition at line 856 of file parser.h.

7.9.1.387 #define THEN 644

Definition at line 857 of file parser.h.

7.9.1.388 #define THRU 645

Definition at line 858 of file parser.h.

7.9.1.389 #define TIME 646

Definition at line 859 of file parser.h.

7.9.1.390 #define TIMES 647

Definition at line 860 of file parser.h.

7.9.1.391 #define TO 648

Definition at line 861 of file parser.h.

7.9.1.392 #define TOK_FALSE 649

Definition at line 862 of file parser.h.

7.9.1.393 #define TOK_FILE 650

Definition at line 863 of file parser.h.

7.9.1.394 #define TOK_INITIAL 651

Definition at line 864 of file parser.h.

7.9.1.395 #define TOK_NULL 652

Definition at line 865 of file parser.h.

7.9.1.396 #define TOK_TRUE 653

Definition at line 866 of file parser.h.

7.9.1.397 #define TOKEN_EOF 0

Definition at line 470 of file parser.h.

7.9.1.398 #define TOP 654

Definition at line 867 of file parser.h.

7.9.1.399 #define TRAILING 655

Definition at line 868 of file parser.h.

7.9.1.400 #define TRANSFORM 656

Definition at line 869 of file parser.h.

7.9.1.401 #define TRIM_FUNCTION 657

Definition at line 870 of file parser.h.

7.9.1.402 #define TYPE 658

Definition at line 871 of file parser.h.

7.9.1.403 #define UNARY_SIGN 692

Definition at line 905 of file parser.h.

7.9.1.404 #define UNDERLINE 659

Definition at line 872 of file parser.h.

7.9.1.405 #define UNIT 660

Definition at line 873 of file parser.h.

7.9.1.406 #define UNLOCK 661

Definition at line 874 of file parser.h.

7.9.1.407 #define UNSIGNED 662

Definition at line 875 of file parser.h.

7.9.1.408 #define UNSIGNED_INT 663

Definition at line 876 of file parser.h.

7.9.1.409 #define UNSIGNED_LONG 664

Definition at line 877 of file parser.h.

7.9.1.410 #define UNSIGNED_SHORT 665

Definition at line 878 of file parser.h.

7.9.1.411 #define UNSTRING 666

Definition at line 879 of file parser.h.

7.9.1.412 #define UNTIL 667

Definition at line 880 of file parser.h.

7.9.1.413 #define UP 668

Definition at line 881 of file parser.h.

7.9.1.414 #define UPDATE 669

Definition at line 882 of file parser.h.

7.9.1.415 #define UPON 670

Definition at line 883 of file parser.h.

7.9.1.416 #define UPON_ARGUMENT_NUMBER 671

Definition at line 884 of file parser.h.

7.9.1.417 #define UPON_COMMAND_LINE 672

Definition at line 885 of file parser.h.

7.9.1.418 #define UPON_ENVIRONMENT_NAME 673

Definition at line 886 of file parser.h.

7.9.1.419 #define UPON_ENVIRONMENT_VALUE 674

Definition at line 887 of file parser.h.

7.9.1.420 #define UPPER_CASE_FUNC 675

Definition at line 888 of file parser.h.

7.9.1.421 #define USAGE 676

Definition at line 889 of file parser.h.

7.9.1.422 #define USE 677

Definition at line 890 of file parser.h.

7.9.1.423 #define USING 678

Definition at line 891 of file parser.h.

7.9.1.424 #define VALUE 679

Definition at line 892 of file parser.h.

7.9.1.425 #define VARYING 680

Definition at line 893 of file parser.h.

7.9.1.426 #define WAIT 681

Definition at line 894 of file parser.h.

7.9.1.427 #define WHEN 682

Definition at line 895 of file parser.h.

7.9.1.428 #define WHEN_COMPILED_FUNC 683

Definition at line 896 of file parser.h.

7.9.1.429 #define WITH 684

Definition at line 897 of file parser.h.

7.9.1.430 #define WORD 685

Definition at line 898 of file parser.h.

7.9.1.431 #define WORDS 686

Definition at line 899 of file parser.h.

7.9.1.432 #define WORKING_STORAGE 687

Definition at line 900 of file parser.h.

7.9.1.433 #define WRITE 688

Definition at line 901 of file parser.h.

7.9.1.434 #define yystype YYSTYPE

Definition at line 912 of file parser.h.

7.9.1.435 #define YYSTYPE_IS_DECLARED 1

Definition at line 913 of file parser.h.

7.9.1.436 #define YYSTYPE_IS_TRIVIAL 1

Definition at line 914 of file parser.h.

7.9.1.437 #define YYYYDDD 689

Definition at line 902 of file parser.h.

7.9.1.438 #define YYYYMMDD 690

Definition at line 903 of file parser.h.

7.9.1.439 #define ZERO 691

Definition at line 904 of file parser.h.

7.9.2 Typedef Documentation

7.9.2.1 typedef int YYSTYPE

Definition at line 911 of file parser.h.

7.9.3 Enumeration Type Documentation

7.9.3.1 enum yytokentype

Enumerator:

TOKEN_EOF

ACCEPT

ACCESS

ADD

ADDRESS

ADVANCING

AFTER

ALL

ALLOCATE

ALPHABET

ALPHABETIC

ALPHABETIC_LOWER

ALPHABETIC_UPPER

ALPHANUMERIC

ALPHANUMERIC_EDITED

ALSO

ALTER

ALTERNATE

AND

ANY

ARE

AREA

ARGUMENT_NUMBER

ARGUMENT_VALUE

AS

ASCENDING

ASSIGN

AT

AUTO

AUTOMATIC

BACKGROUND_COLOR

BASED

BEFORE

BELL

BINARY

BINARY_C_LONG

BINARY_CHAR

BINARY_DOUBLE

BINARY_LONG

BINARY_SHORT

BLANK

BLANK_LINE

BLANK_SCREEN

BLINK

BLOCK

BOTTOM

BY

BYTE_LENGTH

CALL
CANCEL
CH
CHAINING
CHARACTER
CHARACTERS
CLASS
CLOSE
CODE
CODE_SET
COLLATING
COL
COLS
COLUMN
COLUMNS
COMMA
COMMAND_LINE
COMMA_DELIM
COMMIT
COMMON
COMP
COMPUTE
COMP_1
COMP_2
COMP_3
COMP_4
COMP_5
COMP_X
CONCATENATE_FUNC
CONFIGURATION
CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONTROL_FOOTING
CONTROL_HEADING

CONVERTING
CORRESPONDING
COUNT
CRT
CURRENCY
CURRENT_DATE_FUNC
CURSOR
CYCLE
DATA
DATE
DAY
DAY_OF_WEEK
DE
DEBUGGING
DECIMAL_POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EBCDIC
ELSE
END
END_ACCEPT
END_ADD
END_CALL
END_COMPUTE
END_DELETE

END_DISPLAY
END_DIVIDE
END_EVALUATE
END_FUNCTION
END_IF
END_MULTIPLY
END_PERFORM
END_PROGRAM
END_READ
END_RETURN
END_REWRITE
END_SEARCH
END_START
END_STRING
END_SUBTRACT
END_UNSTRING
END_WRITE
ENTRY
ENVIRONMENT
ENVIRONMENT_NAME
ENVIRONMENT_VALUE
EOL
EOP
EOS
EQUAL
EQUALS
ERASE
ERROR
ESCAPE
EVALUATE
EVENT_STATUS
EXCEPTION
EXCLUSIVE
EXIT
EXTEND
EXTERNAL
FD
FILE_CONTROL

FILE_ID
FILLER
FINAL
FIRST
FOOTING
FOR
FOREGROUND_COLOR
FOREVER
FREE
FROM
FULL
FUNCTION
FUNCTION_ID
FUNCTION_NAME
GE
GENERATE
GIVING
GLOBAL
GO
GOBACK
GREATER
GROUP
HEADING
HIGHLIGHT
HIGH_VALUE
IDENTIFICATION
IF
IGNORE
IGNORING
IN
INDEX
INDEXED
INDICATE
INITIALIZE
INITIALIZED
INITIATE
INPUT
INPUT_OUTPUT

INSPECT
INTO
INTRINSIC
INVALID
INVALID_KEY
IS
I_O
I_O_CONTROL
JUSTIFIED
KEY
LABEL
LAST
LAST_DETAIL
LE
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE_COUNTER
LINE
LINES
LINKAGE
LITERAL
LOCALE
LOCALE_DT_FUNC
LOCAL_STORAGE
LOCK
LOWER_CASE_FUNC
LOWLIGHT
LOW_VALUE
MANUAL
MEMORY
MERGE
MINUS
MNEMONIC_NAME

MODE
MOVE
MULTIPLE
MULTIPLY
NATIONAL
NATIONAL_EDITED
NATIVE
NE
NEGATIVE
NEXT
NEXT_SENTENCE
NO
NOT
NOT_END
NOT_EOP
NOT_EXCEPTION
NOT_INVALID_KEY
NOT_OVERFLOW
NOT_SIZE_ERROR
NO_ADVANCING
NUMBER
NUMBERS
NUMERIC
NUMERIC_EDITED
NUMVALC_FUNC
OBJECT_COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER

OUTPUT
OVERFLOW
OVERLINE
PACKED_DECIMAL
PADDING
PAGE
PAGE_FOOTING
PAGE_HEADING
PARAGRAPH
PERFORM
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRESENT
PREVIOUS
PRINTER
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM_ID
PROGRAM_NAME
PROGRAM_POINTER
PROMPT
QUOTE
RANDOM
RD
READ
RECORD
RECORDING
RECORDS
RECURSIVE
REDEFINES
REEL
REFERENCE

RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
REPORT_FOOTING
REPORT_HEADING
REPOSITORY
REQUIRED
RESERVE
RETURN
RETURNING
REVERSE_FUNC
REVERSE_VIDEO
REWIND
REWRITE
RIGHT
ROLLBACK
ROUNDED
RUN
SAME
SCREEN
SCREEN_CONTROL
SCROLL
SD
SEARCH
SECTION
SECURE
SEGMENT_LIMIT
SELECT
SEMI_COLON
SENTENCE
SEPARATE
SEQUENCE

SEQUENTIAL
SET
SHARING
SIGN
SIGNED
SIGNED_INT
SIGNED_LONG
SIGNED_SHORT
SIZE
SIZE_ERROR
SORT
SORT_MERGE
SOURCE
SOURCE_COMPUTER
SPACE
SPECIAL_NAMES
STANDARD
STANDARD_1
STANDARD_2
START
STATUS
STOP
STRING
SUBSTITUTE_FUNC
SUBSTITUTE_CASE_FUNC
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNCHRONIZED
TALLYING
TAPE
TERMINATE
TEST
THAN
THEN
THRU
TIME

TIMES

TO

TOK_FALSE

TOK_FILE

TOK_INITIAL

TOK_NULL

TOK_TRUE

TOP

TRAILING

TRANSFORM

TRIM_FUNCTION

TYPE

UNDERLINE

UNIT

UNLOCK

UNSIGNED

UNSIGNED_INT

UNSIGNED_LONG

UNSIGNED_SHORT

UNSTRING

UNTIL

UP

UPDATE

UPON

UPON_ARGUMENT_NUMBER

UPON_COMMAND_LINE

UPON_ENVIRONMENT_NAME

UPON_ENVIRONMENT_VALUE

UPPER_CASE_FUNC

USAGE

USE

USING

VALUE

VARYING

WAIT

WHEN

WHEN_COMPILED_FUNC

WITH

WORD
WORDS
WORKING_STORAGE
WRITE
YYYYDDD
YYYYMMDD
ZERO
UNARY_SIGN
TOKEN_EOF
ACCEPT
ACCESS
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALLOCATE
ALPHABET
ALPHABETIC
ALPHABETIC_LOWER
ALPHABETIC_UPPER
ALPHANUMERIC
ALPHANUMERIC_EDITED
ALSO
ALTER
ALTERNATE
AND
ANY
ARE
AREA
ARGUMENT_NUMBER
ARGUMENT_VALUE
AS
ASCENDING
ASSIGN
AT
AUTO
AUTOMATIC

BACKGROUND_COLOR
BASED
BEFORE
BELL
BINARY
BINARY_C_LONG
BINARY_CHAR
BINARY_DOUBLE
BINARY_LONG
BINARY_SHORT
BLANK
BLANK_LINE
BLANK_SCREEN
BLINK
BLOCK
BOTTOM
BY
BYTE_LENGTH
CALL
CANCEL
CH
CHAINING
CHARACTER
CHARACTERS
CLASS
CLOSE
CODE
CODE_SET
COLLATING
COL
COLS
COLUMN
COLUMNS
COMMA
COMMAND_LINE
COMMA_DELIM
COMMIT
COMMON

COMP
COMPUTE
COMP_1
COMP_2
COMP_3
COMP_4
COMP_5
COMP_X
CONCATENATE_FUNC
CONFIGURATION
CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONTROL_FOOTING
CONTROL_HEADING
CONVERTING
CORRESPONDING
COUNT
CRT
CURRENCY
CURRENT_DATE_FUNC
CURSOR
CYCLE
DATA
DATE
DAY
DAY_OF_WEEK
DE
DEBUGGING
DECIMAL_POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER

DEPENDING
DESCENDING
DETAIL
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EBCDIC
ELSE
END
END_ACCEPT
END_ADD
END_CALL
END_COMPUTE
END_DELETE
END_DISPLAY
END_DIVIDE
END_EVALUATE
END_FUNCTION
END_IF
END_MULTIPLY
END_PERFORM
END_PROGRAM
END_READ
END_RETURN
END_REWRITE
END_SEARCH
END_START
END_STRING
END_SUBTRACT
END_UNSTRING
END_WRITE
ENTRY
ENVIRONMENT
ENVIRONMENT_NAME

ENVIRONMENT_VALUE

EOL

EOP

EOS

EQUAL

EQUALS

ERASE

ERROR

ESCAPE

EVALUATE

EVENT_STATUS

EXCEPTION

EXCLUSIVE

EXIT

EXTEND

EXTERNAL

FD

FILE_CONTROL

FILE_ID

FILLER

FINAL

FIRST

FOOTING

FOR

FOREGROUND_COLOR

FOREVER

FREE

FROM

FULL

FUNCTION

FUNCTION_ID

FUNCTION_NAME

GE

GENERATE

GIVING

GLOBAL

GO

GOBACK

GREATER
GROUP
HEADING
HIGHLIGHT
HIGH_VALUE
IDENTIFICATION
IF
IGNORE
IGNORING
IN
INDEX
INDEXED
INDICATE
INITIALIZE
INITIALIZED
INITIATE
INPUT
INPUT_OUTPUT
INSPECT
INTO
INTRINSIC
INVALID
INVALID_KEY
IS
I_O
I_O_CONTROL
JUSTIFIED
KEY
LABEL
LAST
LAST_DETAIL
LE
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS

LINAGE
LINAGE_COUNTER
LINE
LINES
LINKAGE
LITERAL
LOCALE
LOCALE_DT_FUNC
LOCAL_STORAGE
LOCK
LOWER_CASE_FUNC
LOWLIGHT
LOW_VALUE
MANUAL
MEMORY
MERGE
MINUS
MNEMONIC_NAME
MODE
MOVE
MULTIPLE
MULTIPLY
NATIONAL
NATIONAL_EDITED
NATIVE
NE
NEGATIVE
NEXT
NEXT_SENTENCE
NO
NOT
NOT_END
NOT_EOP
NOT_EXCEPTION
NOT_INVALID_KEY
NOT_OVERFLOW
NOT_SIZE_ERROR
NO_ADVANCING

NUMBER
NUMBERS
NUMERIC
NUMERIC_EDITED
NUMVALC_FUNC
OBJECT_COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
OVERLINE
PACKED_DECIMAL
PADDING
PAGE
PAGE_FOOTING
PAGE_HEADING
PARAGRAPH
PERFORM
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRESENT
PREVIOUS
PRINTER
PRINTING
PROCEDURE

PROCEDURES
PROCEED
PROGRAM
PROGRAM_ID
PROGRAM_NAME
PROGRAM_POINTER
PROMPT
QUOTE
RANDOM
RD
READ
RECORD
RECORDING
RECORDS
RECURSIVE
REDEFINES
REEL
REFERENCE
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
REPORT_FOOTING
REPORT_HEADING
REPOSITORY
REQUIRED
RESERVE
RETURN
RETURNING
REVERSE_FUNC
REVERSE_VIDEO
REWIND
REWRITE

RIGHT
ROLLBACK
ROUNDED
RUN
SAME
SCREEN
SCREEN_CONTROL
SCROLL
SD
SEARCH
SECTION
SECURE
SEGMENT_LIMIT
SELECT
SEMI_COLON
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHARING
SIGN
SIGNED
SIGNED_INT
SIGNED_LONG
SIGNED_SHORT
SIZE
SIZE_ERROR
SORT
SORT_MERGE
SOURCE
SOURCE_COMPUTER
SPACE
SPECIAL_NAMES
STANDARD
STANDARD_1
STANDARD_2
START

STATUS
STOP
STRING
SUBSTITUTE_FUNC
SUBSTITUTE_CASE_FUNC
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNCHRONIZED
TALLYING
TAPE
TERMINATE
TEST
THAN
THEN
THRU
TIME
TIMES
TO
TOK_FALSE
TOK_FILE
TOK_INITIAL
TOK_NULL
TOK_TRUE
TOP
TRAILING
TRANSFORM
TRIM_FUNCTION
TYPE
UNDERLINE
UNIT
UNLOCK
UNSIGNED
UNSIGNED_INT
UNSIGNED_LONG
UNSIGNED_SHORT
UNSTRING

UNTIL
UP
UPDATE
UPON
UPON_ARGUMENT_NUMBER
UPON_COMMAND_LINE
UPON_ENVIRONMENT_NAME
UPON_ENVIRONMENT_VALUE
UPPER_CASE_FUNC
USAGE
USE
USING
VALUE
VARYING
WAIT
WHEN
WHEN_COMPILED_FUNC
WITH
WORD
WORDS
WORKING_STORAGE
WRITE
YYYYDDD
YYYYMMDD
ZERO
UNARY_SIGN
TOKEN_EOF
COPY
REPLACE
SUPPRESS
PRINTING
REPLACING
OFF
IN
OF
BY
EQEQ
TOKEN

TOKEN_EOF

COPY

REPLACE

SUPPRESS

PRINTING

REPLACING

OFF

IN

OF

BY

EQEQ

TOKEN

Definition at line 31 of file parser.h.

```
                {
TOKEN_EOF = 0,
ACCEPT = 258,
ACCESS = 259,
ADD = 260,
ADDRESS = 261,
ADVANCING = 262,
AFTER = 263,
ALL = 264,
ALLOCATE = 265,
ALPHABET = 266,
ALPHABETIC = 267,
ALPHABETIC_LOWER = 268,
ALPHABETIC_UPPER = 269,
ALPHANUMERIC = 270,
ALPHANUMERIC_EDITED = 271,
ALSO = 272,
ALTER = 273,
ALTERNATE = 274,
AND = 275,
ANY = 276,
ARE = 277,
AREA = 278,
ARGUMENT_NUMBER = 279,
ARGUMENT_VALUE = 280,
AS = 281,
ASCENDING = 282,
ASSIGN = 283,
AT = 284,
AUTO = 285,
AUTOMATIC = 286,
BACKGROUND_COLOR = 287,
BASED = 288,
BEFORE = 289,
BELL = 290,
BINARY = 291,
BINARY_C_LONG = 292,
BINARY_CHAR = 293,
BINARY_DOUBLE = 294,
```

```
  BINARY_LONG = 295,
  BINARY_SHORT = 296,
  BLANK = 297,
  BLANK_LINE = 298,
  BLANK_SCREEN = 299,
  BLINK = 300,
  BLOCK = 301,
  BOTTOM = 302,
  BY = 303,
  BYTE_LENGTH = 304,
  CALL = 305,
  CANCEL = 306,
  CH = 307,
  CHAINING = 308,
  CHARACTER = 309,
  CHARACTERS = 310,
  CLASS = 311,
  CLOSE = 312,
  CODE = 313,
  CODE_SET = 314,
  COLLATING = 315,
  COL = 316,
  COLS = 317,
  COLUMN = 318,
  COLUMNS = 319,
  COMMA = 320,
  COMMAND_LINE = 321,
  COMMA_DELIM = 322,
  COMMIT = 323,
  COMMON = 324,
  COMP = 325,
  COMPUTE = 326,
  COMP_1 = 327,
  COMP_2 = 328,
  COMP_3 = 329,
  COMP_4 = 330,
  COMP_5 = 331,
  COMP_X = 332,
  CONCATENATE_FUNC = 333,
  CONFIGURATION = 334,
  CONSTANT = 335,
  CONTAINS = 336,
  CONTENT = 337,
  CONTINUE = 338,
  CONTROL = 339,
  CONTROLS = 340,
  CONTROL_FOOTING = 341,
  CONTROL_HEADING = 342,
  CONVERTING = 343,
  CORRESPONDING = 344,
  COUNT = 345,
  CRT = 346,
  CURRENCY = 347,
  CURRENT_DATE_FUNC = 348,
  CURSOR = 349,
  CYCLE = 350,
  DATA = 351,
  DATE = 352,
  DAY = 353,
  DAY_OF_WEEK = 354,
  DE = 355,
  DEBUGGING = 356,
```

```
DECIMAL_POINT = 357,
DECLARATIVES = 358,
DEFAULT = 359,
DELETE = 360,
DELIMITED = 361,
DELIMITER = 362,
DEPENDING = 363,
DESCENDING = 364,
DETAIL = 365,
DISK = 366,
DISPLAY = 367,
DIVIDE = 368,
DIVISION = 369,
DOWN = 370,
DUPLICATES = 371,
DYNAMIC = 372,
EBCDIC = 373,
ELSE = 374,
END = 375,
END_ACCEPT = 376,
END_ADD = 377,
END_CALL = 378,
END_COMPUTE = 379,
END_DELETE = 380,
END_DISPLAY = 381,
END_DIVIDE = 382,
END_EVALUATE = 383,
END_FUNCTION = 384,
END_IF = 385,
END_MULTIPLY = 386,
END_PERFORM = 387,
END_PROGRAM = 388,
END_READ = 389,
END_RETURN = 390,
END_REWRITE = 391,
END_SEARCH = 392,
END_START = 393,
END_STRING = 394,
END_SUBTRACT = 395,
END_UNSTRING = 396,
END_WRITE = 397,
ENTRY = 398,
ENVIRONMENT = 399,
ENVIRONMENT_NAME = 400,
ENVIRONMENT_VALUE = 401,
EOL = 402,
EOP = 403,
EOS = 404,
EQUAL = 405,
EQUALS = 406,
ERASE = 407,
ERROR = 408,
ESCAPE = 409,
EVALUATE = 410,
EVENT_STATUS = 411,
EXCEPTION = 412,
EXCLUSIVE = 413,
EXIT = 414,
EXTEND = 415,
EXTERNAL = 416,
FD = 417,
FILE_CONTROL = 418,
```

```
FILE_ID = 419,
FILLER = 420,
FINAL = 421,
FIRST = 422,
FOOTING = 423,
FOR = 424,
FOREGROUND_COLOR = 425,
FOREVER = 426,
FREE = 427,
FROM = 428,
FULL = 429,
FUNCTION = 430,
FUNCTION_ID = 431,
FUNCTION_NAME = 432,
GE = 433,
GENERATE = 434,
GIVING = 435,
GLOBAL = 436,
GO = 437,
GOBACK = 438,
GREATER = 439,
GROUP = 440,
HEADING = 441,
HIGHLIGHT = 442,
HIGH_VALUE = 443,
IDENTIFICATION = 444,
IF = 445,
IGNORE = 446,
IGNORING = 447,
IN = 448,
INDEX = 449,
INDEXED = 450,
INDICATE = 451,
INITIALIZE = 452,
INITIALIZED = 453,
INITIATE = 454,
INPUT = 455,
INPUT_OUTPUT = 456,
INSPECT = 457,
INTO = 458,
INTRINSIC = 459,
INVALID = 460,
INVALID_KEY = 461,
IS = 462,
I_O = 463,
I_O_CONTROL = 464,
JUSTIFIED = 465,
KEY = 466,
LABEL = 467,
LAST = 468,
LAST_DETAIL = 469,
LE = 470,
LEADING = 471,
LEFT = 472,
LENGTH = 473,
LESS = 474,
LIMIT = 475,
LIMITS = 476,
LINAGE = 477,
LINAGE_COUNTER = 478,
LINE = 479,
LINES = 480,
```

```
LINKAGE = 481,
LITERAL = 482,
LOCALE = 483,
LOCALE_DT_FUNC = 484,
LOCAL_STORAGE = 485,
LOCK = 486,
LOWER_CASE_FUNC = 487,
LOWLIGHT = 488,
LOW_VALUE = 489,
MANUAL = 490,
MEMORY = 491,
MERGE = 492,
MINUS = 493,
MNEMONIC_NAME = 494,
MODE = 495,
MOVE = 496,
MULTIPLE = 497,
MULTIPLY = 498,
NATIONAL = 499,
NATIONAL_EDITED = 500,
NATIVE = 501,
NE = 502,
NEGATIVE = 503,
NEXT = 504,
NEXT_SENTENCE = 505,
NO = 506,
NOT = 507,
NOT_END = 508,
NOT_EOP = 509,
NOT_EXCEPTION = 510,
NOT_INVALID_KEY = 511,
NOT_OVERFLOW = 512,
NOT_SIZE_ERROR = 513,
NO_ADVANCING = 514,
NUMBER = 515,
NUMBERS = 516,
NUMERIC = 517,
NUMERIC_EDITED = 518,
NUMVALC_FUNC = 519,
OBJECT_COMPUTER = 520,
OCCURS = 521,
OF = 522,
OFF = 523,
OMITTED = 524,
ON = 525,
ONLY = 526,
OPEN = 527,
OPTIONAL = 528,
OR = 529,
ORDER = 530,
ORGANIZATION = 531,
OTHER = 532,
OUTPUT = 533,
OVERFLOW = 534,
OVERLINE = 535,
PACKED_DECIMAL = 536,
PADDING = 537,
PAGE = 538,
PAGE_FOOTING = 539,
PAGE_HEADING = 540,
PARAGRAPH = 541,
PERFORM = 542,
```

```
PICTURE = 543,
PLUS = 544,
POINTER = 545,
POSITION = 546,
POSITIVE = 547,
PRESENT = 548,
PREVIOUS = 549,
PRINTER = 550,
PRINTING = 551,
PROCEDURE = 552,
PROCEDURES = 553,
PROCEED = 554,
PROGRAM = 555,
PROGRAM_ID = 556,
PROGRAM_NAME = 557,
PROGRAM_POINTER = 558,
PROMPT = 559,
QUOTE = 560,
RANDOM = 561,
RD = 562,
READ = 563,
RECORD = 564,
RECORDING = 565,
RECORDS = 566,
RECURSIVE = 567,
REDEFINES = 568,
REEL = 569,
REFERENCE = 570,
RELATIVE = 571,
RELEASE = 572,
REMAINDER = 573,
REMOVAL = 574,
RENAMES = 575,
REPLACING = 576,
REPORT = 577,
REPORTING = 578,
REPORTS = 579,
REPORT_FOOTING = 580,
REPORT_HEADING = 581,
REPOSITORY = 582,
REQUIRED = 583,
RESERVE = 584,
RETURN = 585,
RETURNING = 586,
REVERSE_FUNC = 587,
REVERSE_VIDEO = 588,
REWIND = 589,
REWRITE = 590,
RIGHT = 591,
ROLLBACK = 592,
ROUNDED = 593,
RUN = 594,
SAME = 595,
SCREEN = 596,
SCREEN_CONTROL = 597,
SCROLL = 598,
SD = 599,
SEARCH = 600,
SECTION = 601,
SECURE = 602,
SEGMENT_LIMIT = 603,
SELECT = 604,
```

```
SEMI_COLON = 605,
SENTENCE = 606,
SEPARATE = 607,
SEQUENCE = 608,
SEQUENTIAL = 609,
SET = 610,
SHARING = 611,
SIGN = 612,
SIGNED = 613,
SIGNED_INT = 614,
SIGNED_LONG = 615,
SIGNED_SHORT = 616,
SIZE = 617,
SIZE_ERROR = 618,
SORT = 619,
SORT_MERGE = 620,
SOURCE = 621,
SOURCE_COMPUTER = 622,
SPACE = 623,
SPECIAL_NAMES = 624,
STANDARD = 625,
STANDARD_1 = 626,
STANDARD_2 = 627,
START = 628,
STATUS = 629,
STOP = 630,
STRING = 631,
SUBSTITUTE_FUNC = 632,
SUBSTITUTE_CASE_FUNC = 633,
SUBTRACT = 634,
SUM = 635,
SUPPRESS = 636,
SYMBOLIC = 637,
SYNCHRONIZED = 638,
TALLYING = 639,
TAPE = 640,
TERMINATE = 641,
TEST = 642,
THAN = 643,
THEN = 644,
THRU = 645,
TIME = 646,
TIMES = 647,
TO = 648,
TOK_FALSE = 649,
TOK_FILE = 650,
TOK_INITIAL = 651,
TOK_NULL = 652,
TOK_TRUE = 653,
TOP = 654,
TRAILING = 655,
TRANSFORM = 656,
TRIM_FUNCTION = 657,
TYPE = 658,
UNDERLINE = 659,
UNIT = 660,
UNLOCK = 661,
UNSIGNED = 662,
UNSIGNED_INT = 663,
UNSIGNED_LONG = 664,
UNSIGNED_SHORT = 665,
UNSTRING = 666,
```

```
    UNTIL = 667,
    UP = 668,
    UPDATE = 669,
    UPON = 670,
    UPON_ARGUMENT_NUMBER = 671,
    UPON_COMMAND_LINE = 672,
    UPON_ENVIRONMENT_NAME = 673,
    UPON_ENVIRONMENT_VALUE = 674,
    UPPER_CASE_FUNC = 675,
    USAGE = 676,
    USE = 677,
    USING = 678,
    VALUE = 679,
    VARYING = 680,
    WAIT = 681,
    WHEN = 682,
    WHEN_COMPILED_FUNC = 683,
    WITH = 684,
    WORD = 685,
    WORDS = 686,
    WORKING_STORAGE = 687,
    WRITE = 688,
    YYYYDDD = 689,
    YYYYMMDD = 690,
    ZERO = 691,
    UNARY_SIGN = 692
  };
```

7.9.4 Variable Documentation

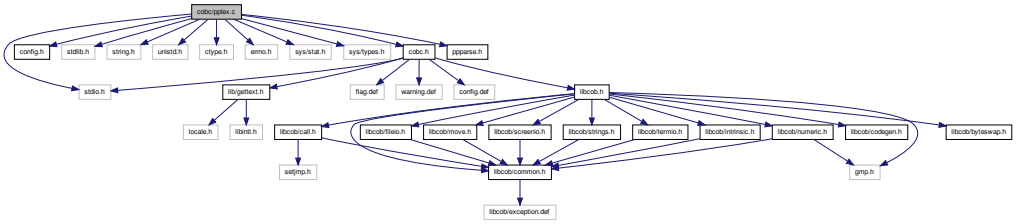
7.9.4.1 YYSTYPE yylval

Definition at line 5081 of file parser.c.

7.10 cobc/pplex.c File Reference

```
#include <stdio.h>
#include "config.h"
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "cobc.h"
#include "ppparse.h"
```

Include dependency graph for pplex.c:



Classes

- struct yy_buffer_state
- struct **copy_info**

Defines

- #define yy_create_buffer pp_create_buffer
- #define yy_delete_buffer pp_delete_buffer
- #define yy_scan_buffer pp_scan_buffer
- #define yy_scan_string pp_scan_string
- #define yy_scan_bytes pp_scan_bytes
- #define yy_flex_debug pp_flex_debug
- #define yy_init_buffer pp_init_buffer
- #define yy_flush_buffer pp_flush_buffer
- #define yy_load_buffer_state pp_load_buffer_state
- #define yy_switch_to_buffer pp_switch_to_buffer
- #define yyin ppin
- #define yyleng ppleng
- #define yylex pplex
- #define yyout ppout
- #define yyrestart pprestart
- #define yytext pptext
- #define FLEX_SCANNER
- #define YY_FLEX_MAJOR_VERSION 2
- #define YY_FLEX_MINOR_VERSION 5
- #define yyconst
- #define YY_PROTO(proto) ()
- #define YY_NULL 0
- #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)
- #define BEGIN yy_start = 1 + 2 ∗
- #define YY_START ((yy_start - 1) / 2)
- #define YYSTATE YY_START

- #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
- #define YY_NEW_FILE yyrestart(yyin)
- #define YY_END_OF_BUFFER_CHAR 0
- #define YY_BUF_SIZE 16384
- #define EOB_ACT_CONTINUE_SCAN 0
- #define EOB_ACT_END_OF_FILE 1
- #define EOB_ACT_LAST_MATCH 2
- #define yyless(n)
- #define unput(c) yyunput(c, yytext_ptr)
- #define YY_BUFFER_NEW 0
- #define YY_BUFFER_NORMAL 1
- #define YY_BUFFER_EOF_PENDING 2
- #define YY_CURRENT_BUFFER yy_current_buffer
- #define YY_FLUSH_BUFFER yy_flush_buffer(yy_current_buffer)
- #define yy_new_buffer yy_create_buffer
- #define yy_set_interactive(is_interactive)
- #define yy_set_bol(at_bol)
- #define YY_AT_BOL() (yy_current_buffer->yy_at_bol)
- #define yywrap() 1
- #define YY_SKIP_YYWRAP
- #define yytext_ptr yytext
- #define YY_DO_BEFORE_ACTION
- #define YY_NUM_RULES 54
- #define YY_END_OF_BUFFER 55
- #define REJECT reject_used_but_not_detected
- #define yymore() yymore_used_but_not_detected
- #define YY_MORE_ADJ 0
- #define YY_RESTORE_YY_MORE_OFFSET
- #define INITIAL 0
- #define YY_NEVER_INTERACTIVE 1
- #define YY_INPUT(buf, result, max_size) result = ppinput (buf, max_size);
- #define PROCESS_STATE 1
- #define COPY_STATE 2
- #define PSEUDO_STATE 3
- #define YY_NO_PUSH_STATE 1
- #define YY_NO_POP_STATE 1
- #define YY_NO_TOP_STATE 1
- #define YY_READ_BUF_SIZE 8192
- #define ECHO (void) fwrite(yytext, yyleng, 1, yyout)
- #define yyterminate() return YY_NULL
- #define YY_START_STACK_INCR 25
- #define YY_FATAL_ERROR(msg) yy_fatal_error(msg)
- #define YY_DECL int yylex YY_PROTO((void))
- #define YY_USER_ACTION
- #define YY_BREAK break;
- #define YY_RULE_SETUP
- #define YY_EXIT_FAILURE 2
- #define yyless(n)

Typedefs

- typedef struct yy_buffer_state * YY_BUFFER_STATE
- typedef unsigned int yy_size_t
- typedef unsigned char YY_CHAR
- typedef int yy_state_type

Functions

- void yyrestart YY_PROTO ((FILE *input_file))
- void yy_switch_to_buffer YY_PROTO ((YY_BUFFER_STATE new_buffer))
- void yy_load_buffer_state YY_PROTO ((void))
- YY_BUFFER_STATE yy_create_buffer YY_PROTO ((FILE *file, int size))
- void yy_delete_buffer YY_PROTO ((YY_BUFFER_STATE b))
- void yy_init_buffer YY_PROTO ((YY_BUFFER_STATE b, FILE *file))
- YY_BUFFER_STATE yy_scan_buffer YY_PROTO ((char *base, yy_size_t size))
- YY_BUFFER_STATE yy_scan_string YY_PROTO ((yyconst char *yy_str))
- YY_BUFFER_STATE yy_scan_bytes YY_PROTO ((yyconst char *bytes, int len))

Variables

- int yyleng
- FILE * yyin = (FILE *) 0
- FILE * yyout = (FILE *) 0
- char * yytext
- register char * yy_bp
- int size
- FILE * file
- int len

7.10.1 Define Documentation

7.10.1.1 #define BEGIN yy_start = 1 + 2 *

Definition at line 97 of file pplex.c.

7.10.1.2 #define COPY_STATE 2

Definition at line 710 of file pplex.c.

7.10.1.3 #define ECHO (void) fwrite(yytext, yyleng, 1, yyout)

Definition at line 793 of file pplex.c.

7.10.1.4 #define EOB_ACT_CONTINUE_SCAN 0

Definition at line 122 of file pplex.c.

7.10.1.5 #define EOB_ACT_END_OF_FILE 1

Definition at line 123 of file pplex.c.

7.10.1.6 #define EOB_ACT_LAST_MATCH 2

Definition at line 124 of file pplex.c.

7.10.1.7 #define FLEX_SCANNER

Definition at line 25 of file pplex.c.

7.10.1.8 #define INITIAL 0

Definition at line 639 of file pplex.c.

7.10.1.9 #define PROCESS_STATE 1

Definition at line 709 of file pplex.c.

7.10.1.10 #define PSEUDO_STATE 3

Definition at line 711 of file pplex.c.

7.10.1.11 #define REJECT reject_used_but_not_detected

Definition at line 633 of file pplex.c.

7.10.1.12 #define unput(*c*) yyunput(c, yytext_ptr)

Definition at line 153 of file pplex.c.

7.10.1.13 #define YY_AT_BOL( ) (yy_current_buffer->yy_at_bol)

Definition at line 280 of file pplex.c.

7.10.1.14 #define YY_BREAK break;

Definition at line 852 of file pplex.c.

7.10.1.15 #define YY_BUF_SIZE 16384

Definition at line 115 of file pplex.c.

7.10.1.16 #define YY_BUFFER_EOF_PENDING 2

Definition at line 216 of file pplex.c.

7.10.1.17 #define YY_BUFFER_NEW 0

Definition at line 204 of file pplex.c.

7.10.1.18 #define YY_BUFFER_NORMAL 1

Definition at line 205 of file pplex.c.

7.10.1.19 #define yy_create_buffer pp_create_buffer

Definition at line 1 of file pplex.c.

7.10.1.20 #define YY_CURRENT_BUFFER yy_current_buffer

Definition at line 225 of file pplex.c.

7.10.1.21 #define YY_DECL int yylex YY_PROTO((void))

Definition at line 840 of file pplex.c.

7.10.1.22 #define yy_delete_buffer pp_delete_buffer

Definition at line 2 of file pplex.c.

7.10.1.23 #define YY_DO_BEFORE_ACTION

Value:

```
yytext_ptr = yy_bp; \
        yyleng = (int) (yy_cp - yy_bp); \
        yy_hold_char = *yy_cp; \
```

```
*yy_cp = '\0'; \
yy_c_buf_p = yy_cp;
```

Definition at line 299 of file pplex.c.

7.10.1.24 #define YY_END_OF_BUFFER 55

Definition at line 307 of file pplex.c.

7.10.1.25 #define YY_END_OF_BUFFER_CHAR 0

Definition at line 112 of file pplex.c.

7.10.1.26 #define YY_EXIT_FAILURE 2

7.10.1.27 #define YY_FATAL_ERROR(*msg*) yy_fatal_error(msg)

Definition at line 833 of file pplex.c.

7.10.1.28 #define yy_flex_debug pp_flex_debug

Definition at line 6 of file pplex.c.

7.10.1.29 #define YY_FLEX_MAJOR_VERSION 2

Definition at line 26 of file pplex.c.

7.10.1.30 #define YY_FLEX_MINOR_VERSION 5

Definition at line 27 of file pplex.c.

7.10.1.31 #define yy_flush_buffer pp_flush_buffer

Definition at line 8 of file pplex.c.

7.10.1.32 #define YY_FLUSH_BUFFER yy_flush_buffer(yy_current_buffer)

Definition at line 254 of file pplex.c.

7.10.1.33 #define yy_init_buffer pp_init_buffer

Definition at line 7 of file pplex.c.

7.10.1.34 #define YY_INPUT(*buf, result, max_size*) result = ppinput (buf, max_size);

Definition at line 701 of file pplex.c.

7.10.1.35 #define yy_load_buffer_state pp_load_buffer_state

Definition at line 9 of file pplex.c.

7.10.1.36 #define YY_MORE_ADJ 0

Definition at line 635 of file pplex.c.

7.10.1.37 #define YY_NEVER_INTERACTIVE 1

Definition at line 659 of file pplex.c.

7.10.1.38 #define yy_new_buffer yy_create_buffer

Definition at line 264 of file pplex.c.

7.10.1.39 #define YY_NEW_FILE yyrestart(yyin)

Definition at line 110 of file pplex.c.

7.10.1.40 #define YY_NO_POP_STATE 1

Definition at line 763 of file pplex.c.

7.10.1.41 #define YY_NO_PUSH_STATE 1

Definition at line 762 of file pplex.c.

7.10.1.42 #define YY_NO_TOP_STATE 1

Definition at line 764 of file pplex.c.

7.10.1.43 #define YY_NULL 0

Definition at line 84 of file pplex.c.

7.10.1.44 #define YY_NUM_RULES 54

Definition at line 306 of file pplex.c.

7.10.1.45 #define YY_PROTO(*proto*) ()

Definition at line 80 of file pplex.c.

7.10.1.46 #define YY_READ_BUF_SIZE 8192

Definition at line 784 of file pplex.c.

7.10.1.47 #define YY_RESTORE_YY_MORE_OFFSET

Definition at line 636 of file pplex.c.

7.10.1.48 #define YY_RULE_SETUP

Value:

```
if ( yyleng > 0 ) \
                yy_current_buffer->yy_at_bol = \
                                (yytext[yyleng - 1] == '\n'); \
        YY_USER_ACTION
```

Definition at line 855 of file pplex.c.

7.10.1.49 #define YY_SC_TO_UI(*c*) ((unsigned int) (unsigned char) c)

Definition at line 91 of file pplex.c.

7.10.1.50 #define yy_scan_buffer pp_scan_buffer

Definition at line 3 of file pplex.c.

7.10.1.51 #define yy_scan_bytes pp_scan_bytes

Definition at line 5 of file pplex.c.

7.10.1.52 #define yy_scan_string pp_scan_string

Definition at line 4 of file pplex.c.

7.10.1.53 #define yy_set_bol(*at_bol*)

Value:

```
{ \
        if ( ! yy_current_buffer ) \
                yy_current_buffer = yy_create_buffer( yyin, YY_BUF_SIZE ); \
        yy_current_buffer->yy_at_bol = at_bol; \
        }
```

Definition at line 273 of file pplex.c.

7.10.1.54 #define yy_set_interactive(*is_interactive*)

Value:

```
{ \
        if ( ! yy_current_buffer ) \
                yy_current_buffer = yy_create_buffer( yyin, YY_BUF_SIZE ); \
        yy_current_buffer->yy_is_interactive = is_interactive; \
        }
```

Definition at line 266 of file pplex.c.

7.10.1.55 #define YY_SKIP_YYWRAP

Definition at line 284 of file pplex.c.

7.10.1.56 #define YY_START ((yy_start - 1) / 2)

Definition at line 103 of file pplex.c.

7.10.1.57 #define YY_START_STACK_INCR 25

Definition at line 828 of file pplex.c.

7.10.1.58 #define YY_STATE_EOF(*state*) (YY_END_OF_BUFFER + state + 1)

Definition at line 107 of file pplex.c.

7.10.1.59 #define yy_switch_to_buffer pp_switch_to_buffer

Definition at line 10 of file pplex.c.

7.10.1.60 #define YY_USER_ACTION

Definition at line 847 of file pplex.c.

7.10.1.61 #define yyconst

Definition at line 73 of file pplex.c.

7.10.1.62 FILE * yyin ppin

Definition at line 11 of file pplex.c.

7.10.1.63 int yyleng ppleng

Definition at line 12 of file pplex.c.

7.10.1.64 #define yyless(*n*)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
                yytext[yyleng] = yy_hold_char; \
                yy_c_buf_p = yytext + n; \
                yy_hold_char = *yy_c_buf_p; \
                *yy_c_buf_p = '\0'; \
                yyleng = n; \
                } \
        while ( 0 )
```

Definition at line 142 of file pplex.c.

7.10.1.65 #define yyless(*n*)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
                *yy_cp = yy_hold_char; \
                YY_RESTORE_YY_MORE_OFFSET \
                yy_c_buf_p = yy_cp = yy_bp + n - YY_MORE_ADJ; \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
```

Definition at line 142 of file pplex.c.

7.10.1.66 #define yylex pplex

Definition at line 13 of file pplex.c.

7.10.1.67 #define yymore( ) yymore_used_but_not_detected

Definition at line 634 of file pplex.c.

7.10.1.68 FILE * yyout ppout

Definition at line 14 of file pplex.c.

7.10.1.69 #define yyrestart pprestart

Definition at line 15 of file pplex.c.

7.10.1.70 #define YYSTATE YY_START

Definition at line 104 of file pplex.c.

7.10.1.71 #define yyterminate( ) return YY_NULL

Definition at line 823 of file pplex.c.

7.10.1.72 char * yytext pptext

Definition at line 16 of file pplex.c.

7.10.1.73 #define yytext_ptr yytext

Definition at line 289 of file pplex.c.

7.10.1.74 #define yywrap( ) 1

Definition at line 283 of file pplex.c.

7.10.2 Typedef Documentation

7.10.2.1 typedef struct yy_buffer_state* YY_BUFFER_STATE

Definition at line 117 of file pplex.c.

7.10.2.2 typedef unsigned char YY_CHAR

Definition at line 285 of file pplex.c.

7.10.2.3 typedef unsigned int yy_size_t

Definition at line 159 of file pplex.c.

7.10.2.4 typedef int yy_state_type

Definition at line 287 of file pplex.c.

7.10.3 Function Documentation

7.10.3.1 YY_BUFFER_STATE yy_scan_string YY_PROTO ((yyconst char *yy_str))

7.10.3.2 YY_BUFFER_STATE yy_scan_buffer YY_PROTO ((char *base, yy_size_t size))

7.10.3.3 YY_BUFFER_STATE yy_scan_bytes YY_PROTO ((yyconst char *bytes, int len))

7.10.3.4 void yy_switch_to_buffer YY_PROTO ((YY_BUFFER_STATE new_buffer))

7.10.3.5 static int input YY_PROTO ((void))

7.10.3.6 void yy_init_buffer YY_PROTO ((YY_BUFFER_STATE b, FILE *file))

7.10.3.7 void yy_flush_buffer YY_PROTO ((YY_BUFFER_STATE b))

7.10.3.8 void yyrestart YY_PROTO ((FILE *input_file))

7.10.3.9 YY_BUFFER_STATE yy_create_buffer YY_PROTO ((FILE *file, int size))

7.10.4 Variable Documentation

7.10.4.1 FILE* file

Definition at line 1807 of file pplex.c.

7.10.4.2 int len

Definition at line 1929 of file pplex.c.

7.10.4.3 yy_size_t size

Definition at line 1718 of file pplex.c.

7.10.4.4 register char* yy_bp

Definition at line 1570 of file pplex.c.

7.10.4.5 FILE* yyin = (FILE *) 0

Definition at line 286 of file pplex.c.

7.10.4.6 int yyleng

Definition at line 234 of file pplex.c.

7.10.4.7 FILE * yyout = (FILE *) 0

Definition at line 286 of file pplex.c.

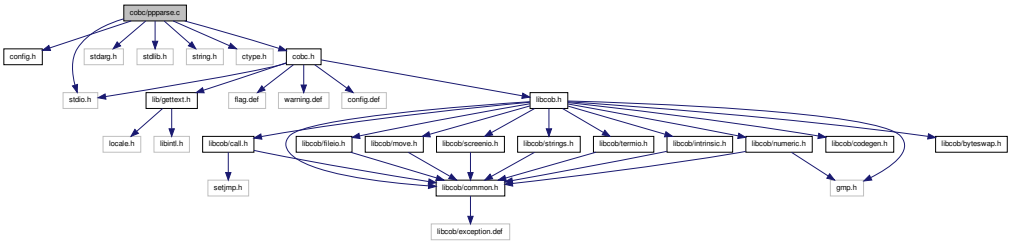
7.10.4.8 char* yytext

Definition at line 637 of file pplex.c.

7.11 cobc/ppparse.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "cobc.h"
```

Include dependency graph for ppparse.c:



Classes

- union YYSTYPE

- union yyalloc

Defines

- #define YYBISON 1
- #define YYSKELETON_NAME "yacc.c"
- #define YYPURE 0
- #define YYLSP_NEEDED 0
- #define yyparse ppparse
- #define yylex pplex
- #define yyerror pperror
- #define yylval pplval
- #define yychar ppchar
- #define yydebug ppdebug
- #define yynerrs ppnerrs
- #define YYTOKENTYPE
- #define TOKEN_EOF 0
- #define COPY 258
- #define REPLACE 259
- #define SUPPRESS 260
- #define PRINTING 261
- #define REPLACING 262
- #define OFF 263
- #define IN 264
- #define OF 265
- #define BY 266
- #define EQEQ 267
- #define TOKEN 268
- #define YYDEBUG 1
- #define YYERROR_VERBOSE 1
- #define pperror cb_error
- #define YYERROR_VERBOSE 1
- #define yystype YYSTYPE
- #define YYSTYPE_IS_DECLARED 1
- #define YYSTYPE_IS_TRIVIAL 1
- #define YYSTACK_ALLOC malloc
- #define YYSTACK_FREE free
- #define YYSTACK_GAP_MAXIMUM (sizeof (union yyalloc) - 1)
- #define YYSTACK_BYTES(N)
- #define YYCOPY(To, From, Count)
- #define YYSTACK_RELOCATE(Stack)
- #define YYFINAL 2
- #define YYLAST 41
- #define YYNTOKENS 17
- #define YYNNTS 15
- #define YYNRULES 31

- #define YYNSTATES 50
- #define YYUNDEFTOK 2
- #define YYMAXUTOK 268
- #define YYTRANSLATE(YYX) ((unsigned int) (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)
- #define YYPACT_NINF -13
- #define YYTABLE_NINF -1
- #define YYSIZE_T unsigned int
- #define yyerrok (yyerrstatus = 0)
- #define yyclearin (yychar = YYEMPTY)
- #define YYEMPTY (-2)
- #define YYEOF 0
- #define YYACCEPT goto yyacceptlab
- #define YYABORT goto yyabortlab
- #define YYERROR goto yyerrlab1
- #define YYFAIL goto yyerrlab
- #define YYRECOVERING() (!!yyerrstatus)
- #define YYBACKUP(Token, Value)
- #define YYTERROR 1
- #define YYERRCODE 256
- #define YYLLOC_DEFAULT(Current, Rhs, N)
- #define YYLEX yylex ()
- #define YYFPRINTF fprintf
- #define YYDPRINTF(Args)
- #define YYDSYMPRINT(Args)
- #define YYDSYMPRINTF(Title, Token, Value, Location)
- #define YY_STACK_PRINT(Bottom, Top)
- #define YY_REDUCE_PRINT(Rule)
- #define YYINITDEPTH 200
- #define YYMAXDEPTH 10000
- #define YYPOPSTACK (yyvsp--, yyssp--)

Typedefs

- typedef union YYSTYPE YYSTYPE
- typedef short yysigned_char

Enumerations

- enum yytokentype {

  TOKEN_EOF = 0, ACCEPT = 258, ACCESS = 259, ADD = 260,

  ADDRESS = 261, ADVANCING = 262, AFTER = 263, ALL = 264,

  ALLOCATE = 265, ALPHABET = 266, ALPHABETIC = 267, ALPHABETIC_-LOWER = 268,

ALPHABETIC_UPPER = 269, ALPHANUMERIC = 270, ALPHANUMERIC_EDITED = 271, ALSO = 272,

ALTER = 273, ALTERNATE = 274, AND = 275, ANY = 276,

ARE = 277, AREA = 278, ARGUMENT_NUMBER = 279, ARGUMENT_VALUE = 280,

AS = 281, ASCENDING = 282, ASSIGN = 283, AT = 284,

AUTO = 285, AUTOMATIC = 286, BACKGROUND_COLOR = 287, BASED = 288,

BEFORE = 289, BELL = 290, BINARY = 291, BINARY_C_LONG = 292,

BINARY_CHAR = 293, BINARY_DOUBLE = 294, BINARY_LONG = 295, BINARY_-SHORT = 296,

BLANK = 297, BLANK_LINE = 298, BLANK_SCREEN = 299, BLINK = 300,

BLOCK = 301, BOTTOM = 302, BY = 303, BYTE_LENGTH = 304,

CALL = 305, CANCEL = 306, CH = 307, CHAINING = 308,

CHARACTER = 309, CHARACTERS = 310, CLASS = 311, CLOSE = 312,

CODE = 313, CODE_SET = 314, COLLATING = 315, COL = 316,

COLS = 317, COLUMN = 318, COLUMNS = 319, COMMA = 320,

COMMAND_LINE = 321, COMMA_DELIM = 322, COMMIT = 323, COMMON = 324,

COMP = 325, COMPUTE = 326, COMP_1 = 327, COMP_2 = 328,

COMP_3 = 329, COMP_4 = 330, COMP_5 = 331, COMP_X = 332,

CONCATENATE_FUNC = 333, CONFIGURATION = 334, CONSTANT = 335, CONTAINS = 336,

CONTENT = 337, CONTINUE = 338, CONTROL = 339, CONTROLS = 340,

CONTROL_FOOTING = 341, CONTROL_HEADING = 342, CONVERTING = 343, CORRESPONDING = 344,

COUNT = 345, CRT = 346, CURRENCY = 347, CURRENT_DATE_FUNC = 348,

CURSOR = 349, CYCLE = 350, DATA = 351, DATE = 352,

DAY = 353, DAY_OF_WEEK = 354, DE = 355, DEBUGGING = 356,

DECIMAL_POINT = 357, DECLARATIVES = 358, DEFAULT = 359, DELETE = 360,

DELIMITED = 361, DELIMITER = 362, DEPENDING = 363, DESCENDING = 364,

DETAIL = 365, DISK = 366, DISPLAY = 367, DIVIDE = 368,

DIVISION = 369, DOWN = 370, DUPLICATES = 371, DYNAMIC = 372,

EBCDIC = 373, ELSE = 374, END = 375, END_ACCEPT = 376,

END_ADD = 377, END_CALL = 378, END_COMPUTE = 379, END_DELETE = 380,

END_DISPLAY = 381, END_DIVIDE = 382, END_EVALUATE = 383, END_-FUNCTION = 384,

END_IF = 385, END_MULTIPLY = 386, END_PERFORM = 387, END_PROGRAM = 388,

END_READ = 389, END_RETURN = 390, END_REWRITE = 391, END_SEARCH = 392,

END_START = 393, END_STRING = 394, END_SUBTRACT = 395, END_UNSTRING = 396,

END_WRITE = 397, ENTRY = 398, ENVIRONMENT = 399, ENVIRONMENT_NAME = 400,

ENVIRONMENT_VALUE = 401, EOL = 402, EOP = 403, EOS = 404,

EQUAL = 405, EQUALS = 406, ERASE = 407, ERROR = 408,

ESCAPE = 409, EVALUATE = 410, EVENT_STATUS = 411, EXCEPTION = 412,

EXCLUSIVE = 413, EXIT = 414, EXTEND = 415, EXTERNAL = 416,

FD = 417, FILE_CONTROL = 418, FILE_ID = 419, FILLER = 420,

FINAL = 421, FIRST = 422, FOOTING = 423, FOR = 424,

FOREGROUND_COLOR = 425, FOREVER = 426, FREE = 427, FROM = 428,

FULL = 429, FUNCTION = 430, FUNCTION_ID = 431, FUNCTION_NAME = 432,

GE = 433, GENERATE = 434, GIVING = 435, GLOBAL = 436,

GO = 437, GOBACK = 438, GREATER = 439, GROUP = 440,

HEADING = 441, HIGHLIGHT = 442, HIGH_VALUE = 443, IDENTIFICATION = 444,

IF = 445, IGNORE = 446, IGNORING = 447, IN = 448,

INDEX = 449, INDEXED = 450, INDICATE = 451, INITIALIZE = 452,

INITIALIZED = 453, INITIATE = 454, INPUT = 455, INPUT_OUTPUT = 456,

INSPECT = 457, INTO = 458, INTRINSIC = 459, INVALID = 460,

INVALID_KEY = 461, IS = 462, I_O = 463, I_O_CONTROL = 464,

JUSTIFIED = 465, KEY = 466, LABEL = 467, LAST = 468,

LAST_DETAIL = 469, LE = 470, LEADING = 471, LEFT = 472,

LENGTH = 473, LESS = 474, LIMIT = 475, LIMITS = 476,

LINAGE = 477, LINAGE_COUNTER = 478, LINE = 479, LINES = 480,

LINKAGE = 481, LITERAL = 482, LOCALE = 483, LOCALE_DT_FUNC = 484,

LOCAL_STORAGE = 485, LOCK = 486, LOWER_CASE_FUNC = 487, LOWLIGHT = 488,

LOW_VALUE = 489, MANUAL = 490, MEMORY = 491, MERGE = 492,

MINUS = 493, MNEMONIC_NAME = 494, MODE = 495, MOVE = 496,

MULTIPLE = 497, MULTIPLY = 498, NATIONAL = 499, NATIONAL_EDITED = 500,

NATIVE = 501, NE = 502, NEGATIVE = 503, NEXT = 504,

NEXT_SENTENCE = 505, NO = 506, NOT = 507, NOT_END = 508,

NOT_EOP = 509, NOT_EXCEPTION = 510, NOT_INVALID_KEY = 511, NOT_OVERFLOW = 512,

NOT_SIZE_ERROR = 513, NO_ADVANCING = 514, NUMBER = 515, NUMBERS = 516,

NUMERIC = 517, NUMERIC_EDITED = 518, NUMVALC_FUNC = 519, OBJECT_-COMPUTER = 520,

OCCURS = 521, OF = 522, OFF = 523, OMITTED = 524,

ON = 525, ONLY = 526, OPEN = 527, OPTIONAL = 528,

OR = 529, ORDER = 530, ORGANIZATION = 531, OTHER = 532,

OUTPUT = 533, OVERFLOW = 534, OVERLINE = 535, PACKED_DECIMAL = 536,

PADDING = 537, PAGE = 538, PAGE_FOOTING = 539, PAGE_HEADING = 540,

PARAGRAPH = 541, PERFORM = 542, PICTURE = 543, PLUS = 544,

POINTER = 545, POSITION = 546, POSITIVE = 547, PRESENT = 548,

PREVIOUS = 549, PRINTER = 550, PRINTING = 551, PROCEDURE = 552,

PROCEDURES = 553, PROCEED = 554, PROGRAM = 555, PROGRAM_ID = 556,

PROGRAM_NAME = 557, PROGRAM_POINTER = 558, PROMPT = 559, QUOTE = 560,

RANDOM = 561, RD = 562, READ = 563, RECORD = 564,

RECORDING = 565, RECORDS = 566, RECURSIVE = 567, REDEFINES = 568,

REEL = 569, REFERENCE = 570, RELATIVE = 571, RELEASE = 572,

REMAINDER = 573, REMOVAL = 574, RENAMES = 575, REPLACING = 576,

REPORT = 577, REPORTING = 578, REPORTS = 579, REPORT_FOOTING = 580,

REPORT_HEADING = 581, REPOSITORY = 582, REQUIRED = 583, RESERVE = 584,

RETURN = 585, RETURNING = 586, REVERSE_FUNC = 587, REVERSE_-VIDEO = 588,

REWIND = 589, REWRITE = 590, RIGHT = 591, ROLLBACK = 592,

ROUNDED = 593, RUN = 594, SAME = 595, SCREEN = 596,

SCREEN_CONTROL = 597, SCROLL = 598, SD = 599, SEARCH = 600,

SECTION = 601, SECURE = 602, SEGMENT_LIMIT = 603, SELECT = 604,

SEMI_COLON = 605, SENTENCE = 606, SEPARATE = 607, SEQUENCE = 608,

SEQUENTIAL = 609, SET = 610, SHARING = 611, SIGN = 612,

SIGNED = 613, SIGNED_INT = 614, SIGNED_LONG = 615, SIGNED_SHORT = 616,

SIZE = 617, SIZE_ERROR = 618, SORT = 619, SORT_MERGE = 620,

SOURCE = 621, SOURCE_COMPUTER = 622, SPACE = 623, SPECIAL_NAMES = 624,

STANDARD = 625, STANDARD_1 = 626, STANDARD_2 = 627, START = 628,

STATUS = 629, STOP = 630, STRING = 631, SUBSTITUTE_FUNC = 632,

SUBSTITUTE_CASE_FUNC = 633, SUBTRACT = 634, SUM = 635, SUPPRESS = 636,

SYMBOLIC = 637, SYNCHRONIZED = 638, TALLYING = 639, TAPE = 640,

TERMINATE = 641, TEST = 642, THAN = 643, THEN = 644,

THRU = 645, TIME = 646, TIMES = 647, TO = 648,

TOK_FALSE = 649, TOK_FILE = 650, TOK_INITIAL = 651, TOK_NULL = 652,

TOK_TRUE = 653, TOP = 654, TRAILING = 655, TRANSFORM = 656,

TRIM_FUNCTION = 657, TYPE = 658, UNDERLINE = 659, UNIT = 660,

UNLOCK = 661, UNSIGNED = 662, UNSIGNED_INT = 663, UNSIGNED_LONG = 664,

UNSIGNED_SHORT = 665, UNSTRING = 666, UNTIL = 667, UP = 668,

UPDATE = 669, UPON = 670, UPON_ARGUMENT_NUMBER = 671, UPON_COMMAND_LINE = 672,

UPON_ENVIRONMENT_NAME = 673, UPON_ENVIRONMENT_VALUE = 674, UPPER_CASE_FUNC = 675, USAGE = 676,

USE = 677, USING = 678, VALUE = 679, VARYING = 680,

WAIT = 681, WHEN = 682, WHEN_COMPILED_FUNC = 683, WITH = 684,

WORD = 685, WORDS = 686, WORKING_STORAGE = 687, WRITE = 688,

YYYYDDD = 689, YYYYMMDD = 690, ZERO = 691, UNARY_SIGN = 692,

TOKEN_EOF = 0, ACCEPT = 258, ACCESS = 259, ADD = 260,

ADDRESS = 261, ADVANCING = 262, AFTER = 263, ALL = 264,

ALLOCATE = 265, ALPHABET = 266, ALPHABETIC = 267, ALPHABETIC_LOWER = 268,

ALPHABETIC_UPPER = 269, ALPHANUMERIC = 270, ALPHANUMERIC_EDITED = 271, ALSO = 272,

ALTER = 273, ALTERNATE = 274, AND = 275, ANY = 276,

ARE = 277, AREA = 278, ARGUMENT_NUMBER = 279, ARGUMENT_VALUE = 280,

AS = 281, ASCENDING = 282, ASSIGN = 283, AT = 284,

AUTO = 285, AUTOMATIC = 286, BACKGROUND_COLOR = 287, BASED = 288,

BEFORE = 289, BELL = 290, BINARY = 291, BINARY_C_LONG = 292,

BINARY_CHAR = 293, BINARY_DOUBLE = 294, BINARY_LONG = 295, BINARY_SHORT = 296,

BLANK = 297, BLANK_LINE = 298, BLANK_SCREEN = 299, BLINK = 300,

BLOCK = 301, BOTTOM = 302, BY = 303, BYTE_LENGTH = 304,

CALL = 305, CANCEL = 306, CH = 307, CHAINING = 308,

CHARACTER = 309, CHARACTERS = 310, CLASS = 311, CLOSE = 312,

CODE = 313, CODE_SET = 314, COLLATING = 315, COL = 316,

COLS = 317, COLUMN = 318, COLUMNS = 319, COMMA = 320,

COMMAND_LINE = 321, COMMA_DELIM = 322, COMMIT = 323, COMMON = 324,

COMP = 325, COMPUTE = 326, COMP_1 = 327, COMP_2 = 328,

COMP_3 = 329, COMP_4 = 330, COMP_5 = 331, COMP_X = 332,

CONCATENATE_FUNC = 333, CONFIGURATION = 334, CONSTANT = 335, CONTAINS = 336,

CONTENT = 337, CONTINUE = 338, CONTROL = 339, CONTROLS = 340,

CONTROL_FOOTING = 341, CONTROL_HEADING = 342, CONVERTING = 343, CORRESPONDING = 344,

COUNT = 345, CRT = 346, CURRENCY = 347, CURRENT_DATE_FUNC = 348,

CURSOR = 349, CYCLE = 350, DATA = 351, DATE = 352,

DAY = 353, DAY_OF_WEEK = 354, DE = 355, DEBUGGING = 356,

DECIMAL_POINT = 357, DECLARATIVES = 358, DEFAULT = 359, DELETE = 360,

DELIMITED = 361, DELIMITER = 362, DEPENDING = 363, DESCENDING = 364,

DETAIL = 365, DISK = 366, DISPLAY = 367, DIVIDE = 368,

DIVISION = 369, DOWN = 370, DUPLICATES = 371, DYNAMIC = 372,

EBCDIC = 373, ELSE = 374, END = 375, END_ACCEPT = 376,

END_ADD = 377, END_CALL = 378, END_COMPUTE = 379, END_DELETE = 380,

END_DISPLAY = 381, END_DIVIDE = 382, END_EVALUATE = 383, END_-FUNCTION = 384,

END_IF = 385, END_MULTIPLY = 386, END_PERFORM = 387, END_PROGRAM = 388,

END_READ = 389, END_RETURN = 390, END_REWRITE = 391, END_SEARCH = 392,

END_START = 393, END_STRING = 394, END_SUBTRACT = 395, END_UNSTRING = 396,

END_WRITE = 397, ENTRY = 398, ENVIRONMENT = 399, ENVIRONMENT_-NAME = 400,

ENVIRONMENT_VALUE = 401, EOL = 402, EOP = 403, EOS = 404,

EQUAL = 405, EQUALS = 406, ERASE = 407, ERROR = 408,

ESCAPE = 409, EVALUATE = 410, EVENT_STATUS = 411, EXCEPTION = 412,

EXCLUSIVE = 413, EXIT = 414, EXTEND = 415, EXTERNAL = 416,

FD = 417, FILE_CONTROL = 418, FILE_ID = 419, FILLER = 420,

FINAL = 421, FIRST = 422, FOOTING = 423, FOR = 424,

FOREGROUND_COLOR = 425, FOREVER = 426, FREE = 427, FROM = 428,

FULL = 429, FUNCTION = 430, FUNCTION_ID = 431, FUNCTION_NAME = 432,

GE = 433, GENERATE = 434, GIVING = 435, GLOBAL = 436,

GO = 437, GOBACK = 438, GREATER = 439, GROUP = 440,

HEADING = 441, HIGHLIGHT = 442, HIGH_VALUE = 443, IDENTIFICATION = 444,

IF = 445, IGNORE = 446, IGNORING = 447, IN = 448,

INDEX = 449, INDEXED = 450, INDICATE = 451, INITIALIZE = 452,

INITIALIZED = 453, INITIATE = 454, INPUT = 455, INPUT_OUTPUT = 456,

INSPECT = 457, INTO = 458, INTRINSIC = 459, INVALID = 460,

INVALID_KEY = 461, IS = 462, I_O = 463, I_O_CONTROL = 464,

JUSTIFIED = 465, KEY = 466, LABEL = 467, LAST = 468,

LAST_DETAIL = 469, LE = 470, LEADING = 471, LEFT = 472,

LENGTH = 473, LESS = 474, LIMIT = 475, LIMITS = 476,

LINAGE = 477, LINAGE_COUNTER = 478, LINE = 479, LINES = 480,

LINKAGE = 481, LITERAL = 482, LOCALE = 483, LOCALE_DT_FUNC = 484,

LOCAL_STORAGE = 485, LOCK = 486, LOWER_CASE_FUNC = 487, LOW-LIGHT = 488,

LOW_VALUE = 489, MANUAL = 490, MEMORY = 491, MERGE = 492,

MINUS = 493, MNEMONIC_NAME = 494, MODE = 495, MOVE = 496,

MULTIPLE = 497, MULTIPLY = 498, NATIONAL = 499, NATIONAL_EDITED = 500,

NATIVE = 501, NE = 502, NEGATIVE = 503, NEXT = 504,

NEXT_SENTENCE = 505, NO = 506, NOT = 507, NOT_END = 508,

NOT_EOP = 509, NOT_EXCEPTION = 510, NOT_INVALID_KEY = 511, NOT_-OVERFLOW = 512,

NOT_SIZE_ERROR = 513, NO_ADVANCING = 514, NUMBER = 515, NUM-BERS = 516,

NUMERIC = 517, NUMERIC_EDITED = 518, NUMVALC_FUNC = 519, OBJECT_-COMPUTER = 520,

OCCURS = 521, OF = 522, OFF = 523, OMITTED = 524,

ON = 525, ONLY = 526, OPEN = 527, OPTIONAL = 528,

OR = 529, ORDER = 530, ORGANIZATION = 531, OTHER = 532,

OUTPUT = 533, OVERFLOW = 534, OVERLINE = 535, PACKED_DECIMAL = 536,

PADDING = 537, PAGE = 538, PAGE_FOOTING = 539, PAGE_HEADING = 540,

PARAGRAPH = 541, PERFORM = 542, PICTURE = 543, PLUS = 544,

POINTER = 545, POSITION = 546, POSITIVE = 547, PRESENT = 548,

PREVIOUS = 549, PRINTER = 550, PRINTING = 551, PROCEDURE = 552,

PROCEDURES = 553, PROCEED = 554, PROGRAM = 555, PROGRAM_ID = 556,

PROGRAM_NAME = 557, PROGRAM_POINTER = 558, PROMPT = 559, QUOTE = 560,

RANDOM = 561, RD = 562, READ = 563, RECORD = 564,

RECORDING = 565, RECORDS = 566, RECURSIVE = 567, REDEFINES = 568,

REEL = 569, REFERENCE = 570, RELATIVE = 571, RELEASE = 572,

REMAINDER = 573, REMOVAL = 574, RENAMES = 575, REPLACING = 576,

REPORT = 577, REPORTING = 578, REPORTS = 579, REPORT_FOOTING = 580,

REPORT_HEADING = 581, REPOSITORY = 582, REQUIRED = 583, RESERVE = 584,

RETURN = 585, RETURNING = 586, REVERSE_FUNC = 587, REVERSE_-VIDEO = 588,

REWIND = 589, REWRITE = 590, RIGHT = 591, ROLLBACK = 592,

ROUNDED = 593, RUN = 594, SAME = 595, SCREEN = 596,

SCREEN_CONTROL = 597, SCROLL = 598, SD = 599, SEARCH = 600,

SECTION = 601, SECURE = 602, SEGMENT_LIMIT = 603, SELECT = 604,

SEMI_COLON = 605, SENTENCE = 606, SEPARATE = 607, SEQUENCE = 608,

SEQUENTIAL = 609, SET = 610, SHARING = 611, SIGN = 612,

SIGNED = 613, SIGNED_INT = 614, SIGNED_LONG = 615, SIGNED_SHORT = 616,

SIZE = 617, SIZE_ERROR = 618, SORT = 619, SORT_MERGE = 620,

SOURCE = 621, SOURCE_COMPUTER = 622, SPACE = 623, SPECIAL_NAMES = 624,

STANDARD = 625, STANDARD_1 = 626, STANDARD_2 = 627, START = 628,

STATUS = 629, STOP = 630, STRING = 631, SUBSTITUTE_FUNC = 632,

SUBSTITUTE_CASE_FUNC = 633, SUBTRACT = 634, SUM = 635, SUPPRESS = 636,

SYMBOLIC = 637, SYNCHRONIZED = 638, TALLYING = 639, TAPE = 640,

TERMINATE = 641, TEST = 642, THAN = 643, THEN = 644,

THRU = 645, TIME = 646, TIMES = 647, TO = 648,

TOK_FALSE = 649, TOK_FILE = 650, TOK_INITIAL = 651, TOK_NULL = 652,

TOK_TRUE = 653, TOP = 654, TRAILING = 655, TRANSFORM = 656,

TRIM_FUNCTION = 657, TYPE = 658, UNDERLINE = 659, UNIT = 660,

UNLOCK = 661, UNSIGNED = 662, UNSIGNED_INT = 663, UNSIGNED_LONG = 664,

UNSIGNED_SHORT = 665, UNSTRING = 666, UNTIL = 667, UP = 668,

UPDATE = 669, UPON = 670, UPON_ARGUMENT_NUMBER = 671, UPON_-COMMAND_LINE = 672,

UPON_ENVIRONMENT_NAME = 673, UPON_ENVIRONMENT_VALUE = 674, UPPER_CASE_FUNC = 675, USAGE = 676,

USE = 677, USING = 678, VALUE = 679, VARYING = 680,

WAIT = 681, WHEN = 682, WHEN_COMPILED_FUNC = 683, WITH = 684,

WORD = 685, WORDS = 686, WORKING_STORAGE = 687, WRITE = 688,

YYYYDDD = 689, YYYYMMDD = 690, ZERO = 691, UNARY_SIGN = 692,

TOKEN_EOF = 0, COPY = 258, REPLACE = 259, SUPPRESS = 260,

PRINTING = 261, REPLACING = 262, OFF = 263, IN = 264,
OF = 265, BY = 266, EQEQ = 267, TOKEN = 268,
TOKEN_EOF = 0, COPY = 258, REPLACE = 259, SUPPRESS = 260,
PRINTING = 261, REPLACING = 262, OFF = 263, IN = 264,
OF = 265, BY = 266, EQEQ = 267, TOKEN = 268 }

Functions

- int yyparse ()

Variables

- int yydebug
- int yychar
- YYSTYPE yylval
- int yynerrs

7.11.1 Define Documentation

7.11.1.1 #define BY 266

Definition at line 88 of file ppparse.c.

7.11.1.2 #define COPY 258

Definition at line 80 of file ppparse.c.

7.11.1.3 #define EQEQ 267

Definition at line 89 of file ppparse.c.

7.11.1.4 #define IN 264

Definition at line 86 of file ppparse.c.

7.11.1.5 #define OF 265

Definition at line 87 of file ppparse.c.

7.11.1.6 #define OFF 263

Definition at line 85 of file ppparse.c.

7.11.1.7 #define pperror cb_error

Definition at line 110 of file ppparse.c.

7.11.1.8 #define PRINTING 261

Definition at line 83 of file ppparse.c.

7.11.1.9 #define REPLACE 259

Definition at line 81 of file ppparse.c.

7.11.1.10 #define REPLACING 262

Definition at line 84 of file ppparse.c.

7.11.1.11 #define SUPPRESS 260

Definition at line 82 of file ppparse.c.

7.11.1.12 #define TOKEN 268

Definition at line 90 of file ppparse.c.

7.11.1.13 #define TOKEN_EOF 0

Definition at line 79 of file ppparse.c.

7.11.1.14 #define YY_REDUCE_PRINT(*Rule*)

Value:

```
do {                                    \
  if (yydebug)                          \
    yy_reduce_print (Rule);             \
} while (0)
```

Definition at line 604 of file ppparse.c.

7.11.1.15 #define YY_STACK_PRINT(*Bottom, Top*)

Value:

```
do {                                                                    \
  if (yydebug)                                                          \
    yy_stack_print ((Bottom), (Top));                                   \
} while (0)
```

Definition at line 574 of file ppparse.c.

7.11.1.16 #define YYABORT goto yyabortlab

Definition at line 472 of file ppparse.c.

7.11.1.17 #define YYACCEPT goto yyacceptlab

Definition at line 471 of file ppparse.c.

7.11.1.18 #define YYBACKUP(*Token, Value*)

Value:

```
do                                                                      \
  if (yychar == YYEMPTY && yylen == 1)                                  \
    {                                                                   \
      yychar = (Token);                                                 \
      yylval = (Value);                                                 \
      yytoken = YYTRANSLATE (yychar);                                   \
      YYPOPSTACK;                                                       \
      goto yybackup;                                                    \
    }                                                                   \
  else                                                                  \
    {                                                                   \
      yyerror ("syntax error: cannot back up");\
      YYERROR;                                                          \
    }                                                                   \
while (0)
```

Definition at line 483 of file ppparse.c.

7.11.1.19 #define YYBISON 1

Definition at line 37 of file ppparse.c.

7.11.1.20 #define yychar ppchar

Definition at line 54 of file ppparse.c.

7.11.1.21 #define yyclearin (yychar = YYEMPTY)

Definition at line 467 of file ppparse.c.

7.11.1.22 #define YYCOPY(*To, From, Count*)

Value:

```
do                                      \
        {                                       \
          register YYSIZE_T yyi;                \
          for (yyi = 0; yyi < (Count); yyi++)   \
            (To)[yyi] = (From)[yyi];            \
        }                                       \
      while (0)
```

Definition at line 213 of file ppparse.c.

7.11.1.23 #define yydebug ppdebug

Definition at line 55 of file ppparse.c.

7.11.1.24 #define YYDEBUG 1

Definition at line 108 of file ppparse.c.

7.11.1.25 #define YYDPRINTF(*Args*)

Value:

```
do {                                            \
  if (yydebug)                                  \
    YYFPRINTF Args;                             \
} while (0)
```

Definition at line 530 of file ppparse.c.

7.11.1.26 #define YYDSYMPRINT(*Args*)

Value:

```
do {                                            \
  if (yydebug)                                  \
    yysymprint Args;                            \
} while (0)
```

Definition at line 536 of file ppparse.c.

7.11.1.27 #define YYDSYMPRINTF(*Title, Token, Value, Location*)

Value:

```
do {                                                                    \
  if (yydebug)                                                          \
    {                                                                   \
      YYFPRINTF (stderr, "%s ", Title);                                 \
      yysymprint (stderr,                                               \
                  Token, Value);        \
      YYFPRINTF (stderr, "\n");                                         \
    }                                                                   \
} while (0)
```

Definition at line 542 of file ppparse.c.

7.11.1.28 #define YYEMPTY (-2)

Definition at line 468 of file ppparse.c.

7.11.1.29 #define YYEOF 0

Definition at line 469 of file ppparse.c.

7.11.1.30 #define YYERRCODE 256

Definition at line 501 of file ppparse.c.

7.11.1.31 #define yyerrok (yyerrstatus = 0)

Definition at line 466 of file ppparse.c.

7.11.1.32 #define yyerror pperror

Definition at line 52 of file ppparse.c.

7.11.1.33 #define YYERROR goto yyerrlab1

Definition at line 473 of file ppparse.c.

7.11.1.34 #define YYERROR_VERBOSE 1

Definition at line 127 of file ppparse.c.

7.11.1.35 #define YYERROR_VERBOSE 1

Definition at line 127 of file ppparse.c.

7.11.1.36 #define YYFAIL goto yyerrlab

Definition at line 479 of file ppparse.c.

7.11.1.37 #define YYFINAL 2

Definition at line 249 of file ppparse.c.

7.11.1.38 #define YYFPRINTF fprintf

Definition at line 527 of file ppparse.c.

7.11.1.39 #define YYINITDEPTH 200

Definition at line 624 of file ppparse.c.

7.11.1.40 #define YYLAST 41

Definition at line 251 of file ppparse.c.

7.11.1.41 #define yylex pplex

Definition at line 51 of file ppparse.c.

7.11.1.42 #define YYLEX yylex ()

Definition at line 519 of file ppparse.c.

7.11.1.43 #define YYLLOC_DEFAULT(*Current, Rhs, N*)

Value:

```
Current.first_line   = Rhs[1].first_line;      \
  Current.first_column = Rhs[1].first_column;    \
  Current.last_line    = Rhs[N].last_line;       \
  Current.last_column  = Rhs[N].last_column;
```

Definition at line 507 of file ppparse.c.

7.11.1.44 #define YYLSP_NEEDED 0

Definition at line 46 of file ppparse.c.

7.11.1.45 #define yylval pplval

Definition at line 53 of file ppparse.c.

7.11.1.46 #define YYMAXDEPTH 10000

Definition at line 639 of file ppparse.c.

7.11.1.47 #define YYMAXUTOK 268

Definition at line 264 of file ppparse.c.

7.11.1.48 #define yynerrs ppnerrs

Definition at line 56 of file ppparse.c.

7.11.1.49 #define YYNNTS 15

Definition at line 256 of file ppparse.c.

7.11.1.50 #define YYNRULES 31

Definition at line 258 of file ppparse.c.

7.11.1.51 #define YYNSTATES 50

Definition at line 260 of file ppparse.c.

7.11.1.52 #define YYNTOKENS 17

Definition at line 254 of file ppparse.c.

7.11.1.53 #define YYPACT_NINF -13

Definition at line 399 of file ppparse.c.

7.11.1.54 int yyparse ppparse

Definition at line 50 of file ppparse.c.

7.11.1.55 #define YYPOPSTACK (yyvsp--, yyssp--)

7.11.1.56 #define YYPURE 0

Definition at line 43 of file ppparse.c.

7.11.1.57 #define YYRECOVERING( ) (!!yyerrstatus)

Definition at line 481 of file ppparse.c.

7.11.1.58 #define YYSIZE_T unsigned int

Definition at line 463 of file ppparse.c.

7.11.1.59 #define YYSKELETON_NAME "yacc.c"

Definition at line 40 of file ppparse.c.

7.11.1.60 #define YYSTACK_ALLOC malloc

Definition at line 180 of file ppparse.c.

7.11.1.61 #define YYSTACK_BYTES(*N*)

Value:

```
((N) * (sizeof (short) + sizeof (YYSTYPE))                \
      + YYSTACK_GAP_MAXIMUM)
```

Definition at line 202 of file ppparse.c.

7.11.1.62 #define YYSTACK_FREE free

Definition at line 181 of file ppparse.c.

7.11.1.63 #define YYSTACK_GAP_MAXIMUM (sizeof (union yyalloc) - 1)

Definition at line 198 of file ppparse.c.

7.11.1.64 #define YYSTACK_RELOCATE(*Stack*)

Value:

```
do                                                                      \
      {                                                                 \
        YYSIZE_T yynewbytes;                                            \
        YYCOPY (&yyptr->Stack, Stack, yysize);                          \
        Stack = &yyptr->Stack;                                          \
        yynewbytes = yystacksize * sizeof (*Stack) + YYSTACK_GAP_MAXIMUM; \
        yyptr += yynewbytes / sizeof (*yyptr);                          \
      }                                                                 \
    while (0)
```

Definition at line 229 of file ppparse.c.

7.11.1.65 #define yystype YYSTYPE

Definition at line 141 of file ppparse.c.

7.11.1.66 #define YYSTYPE_IS_DECLARED 1

Definition at line 142 of file ppparse.c.

7.11.1.67 #define YYSTYPE_IS_TRIVIAL 1

Definition at line 143 of file ppparse.c.

7.11.1.68 #define YYTABLE_NINF -1

Definition at line 420 of file ppparse.c.

7.11.1.69 #define YYTERROR 1

Definition at line 500 of file ppparse.c.

7.11.1.70 #define YYTOKENTYPE

Definition at line 61 of file ppparse.c.

7.11.1.71 #define YYTRANSLATE(*YYX*) ((unsigned int) (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)

Definition at line 266 of file ppparse.c.

7.11.1.72 #define YYUNDEFTOK 2

Definition at line 263 of file ppparse.c.

7.11.2 Typedef Documentation

7.11.2.1 typedef short yysigned_char

Definition at line 245 of file ppparse.c.

7.11.2.2 typedef union YYSTYPE YYSTYPE

7.11.3 Enumeration Type Documentation

7.11.3.1 enum yytokentype

Enumerator:

TOKEN_EOF
ACCEPT
ACCESS
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALLOCATE
ALPHABET
ALPHABETIC
ALPHABETIC_LOWER
ALPHABETIC_UPPER
ALPHANUMERIC
ALPHANUMERIC_EDITED
ALSO
ALTER
ALTERNATE
AND
ANY
ARE
AREA
ARGUMENT_NUMBER
ARGUMENT_VALUE
AS
ASCENDING
ASSIGN

AT
AUTO
AUTOMATIC
BACKGROUND_COLOR
BASED
BEFORE
BELL
BINARY
BINARY_C_LONG
BINARY_CHAR
BINARY_DOUBLE
BINARY_LONG
BINARY_SHORT
BLANK
BLANK_LINE
BLANK_SCREEN
BLINK
BLOCK
BOTTOM
BY
BYTE_LENGTH
CALL
CANCEL
CH
CHAINING
CHARACTER
CHARACTERS
CLASS
CLOSE
CODE
CODE_SET
COLLATING
COL
COLS
COLUMN
COLUMNS
COMMA
COMMAND_LINE

COMMA_DELIM
COMMIT
COMMON
COMP
COMPUTE
COMP_1
COMP_2
COMP_3
COMP_4
COMP_5
COMP_X
CONCATENATE_FUNC
CONFIGURATION
CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONTROL_FOOTING
CONTROL_HEADING
CONVERTING
CORRESPONDING
COUNT
CRT
CURRENCY
CURRENT_DATE_FUNC
CURSOR
CYCLE
DATA
DATE
DAY
DAY_OF_WEEK
DE
DEBUGGING
DECIMAL_POINT
DECLARATIVES
DEFAULT

DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EBCDIC
ELSE
END
END_ACCEPT
END_ADD
END_CALL
END_COMPUTE
END_DELETE
END_DISPLAY
END_DIVIDE
END_EVALUATE
END_FUNCTION
END_IF
END_MULTIPLY
END_PERFORM
END_PROGRAM
END_READ
END_RETURN
END_REWRITE
END_SEARCH
END_START
END_STRING
END_SUBTRACT
END_UNSTRING
END_WRITE

ENTRY
ENVIRONMENT
ENVIRONMENT_NAME
ENVIRONMENT_VALUE
EOL
EOP
EOS
EQUAL
EQUALS
ERASE
ERROR
ESCAPE
EVALUATE
EVENT_STATUS
EXCEPTION
EXCLUSIVE
EXIT
EXTEND
EXTERNAL
FD
FILE_CONTROL
FILE_ID
FILLER
FINAL
FIRST
FOOTING
FOR
FOREGROUND_COLOR
FOREVER
FREE
FROM
FULL
FUNCTION
FUNCTION_ID
FUNCTION_NAME
GE
GENERATE
GIVING

GLOBAL
GO
GOBACK
GREATER
GROUP
HEADING
HIGHLIGHT
HIGH_VALUE
IDENTIFICATION
IF
IGNORE
IGNORING
IN
INDEX
INDEXED
INDICATE
INITIALIZE
INITIALIZED
INITIATE
INPUT
INPUT_OUTPUT
INSPECT
INTO
INTRINSIC
INVALID
INVALID_KEY
IS
I_O
I_O_CONTROL
JUSTIFIED
KEY
LABEL
LAST
LAST_DETAIL
LE
LEADING
LEFT
LENGTH

LESS
LIMIT
LIMITS
LINAGE
LINAGE_COUNTER
LINE
LINES
LINKAGE
LITERAL
LOCALE
LOCALE_DT_FUNC
LOCAL_STORAGE
LOCK
LOWER_CASE_FUNC
LOWLIGHT
LOW_VALUE
MANUAL
MEMORY
MERGE
MINUS
MNEMONIC_NAME
MODE
MOVE
MULTIPLE
MULTIPLY
NATIONAL
NATIONAL_EDITED
NATIVE
NE
NEGATIVE
NEXT
NEXT_SENTENCE
NO
NOT
NOT_END
NOT_EOP
NOT_EXCEPTION
NOT_INVALID_KEY

NOT_OVERFLOW
NOT_SIZE_ERROR
NO_ADVANCING
NUMBER
NUMBERS
NUMERIC
NUMERIC_EDITED
NUMVALC_FUNC
OBJECT_COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
OVERLINE
PACKED_DECIMAL
PADDING
PAGE
PAGE_FOOTING
PAGE_HEADING
PARAGRAPH
PERFORM
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRESENT
PREVIOUS

PRINTER
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM_ID
PROGRAM_NAME
PROGRAM_POINTER
PROMPT
QUOTE
RANDOM
RD
READ
RECORD
RECORDING
RECORDS
RECURSIVE
REDEFINES
REEL
REFERENCE
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
REPORT_FOOTING
REPORT_HEADING
REPOSITORY
REQUIRED
RESERVE
RETURN
RETURNING
REVERSE_FUNC

REVERSE_VIDEO
REWIND
REWRITE
RIGHT
ROLLBACK
ROUNDED
RUN
SAME
SCREEN
SCREEN_CONTROL
SCROLL
SD
SEARCH
SECTION
SECURE
SEGMENT_LIMIT
SELECT
SEMI_COLON
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHARING
SIGN
SIGNED
SIGNED_INT
SIGNED_LONG
SIGNED_SHORT
SIZE
SIZE_ERROR
SORT
SORT_MERGE
SOURCE
SOURCE_COMPUTER
SPACE
SPECIAL_NAMES
STANDARD

STANDARD_1
STANDARD_2
START
STATUS
STOP
STRING
SUBSTITUTE_FUNC
SUBSTITUTE_CASE_FUNC
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNCHRONIZED
TALLYING
TAPE
TERMINATE
TEST
THAN
THEN
THRU
TIME
TIMES
TO
TOK_FALSE
TOK_FILE
TOK_INITIAL
TOK_NULL
TOK_TRUE
TOP
TRAILING
TRANSFORM
TRIM_FUNCTION
TYPE
UNDERLINE
UNIT
UNLOCK
UNSIGNED
UNSIGNED_INT

UNSIGNED_LONG
UNSIGNED_SHORT
UNSTRING
UNTIL
UP
UPDATE
UPON
UPON_ARGUMENT_NUMBER
UPON_COMMAND_LINE
UPON_ENVIRONMENT_NAME
UPON_ENVIRONMENT_VALUE
UPPER_CASE_FUNC
USAGE
USE
USING
VALUE
VARYING
WAIT
WHEN
WHEN_COMPILED_FUNC
WITH
WORD
WORDS
WORKING_STORAGE
WRITE
YYYYDDD
YYYYMMDD
ZERO
UNARY_SIGN
TOKEN_EOF
ACCEPT
ACCESS
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALLOCATE

ALPHABET
ALPHABETIC
ALPHABETIC_LOWER
ALPHABETIC_UPPER
ALPHANUMERIC
ALPHANUMERIC_EDITED
ALSO
ALTER
ALTERNATE
AND
ANY
ARE
AREA
ARGUMENT_NUMBER
ARGUMENT_VALUE
AS
ASCENDING
ASSIGN
AT
AUTO
AUTOMATIC
BACKGROUND_COLOR
BASED
BEFORE
BELL
BINARY
BINARY_C_LONG
BINARY_CHAR
BINARY_DOUBLE
BINARY_LONG
BINARY_SHORT
BLANK
BLANK_LINE
BLANK_SCREEN
BLINK
BLOCK
BOTTOM
BY

BYTE_LENGTH
CALL
CANCEL
CH
CHAINING
CHARACTER
CHARACTERS
CLASS
CLOSE
CODE
CODE_SET
COLLATING
COL
COLS
COLUMN
COLUMNS
COMMA
COMMAND_LINE
COMMA_DELIM
COMMIT
COMMON
COMP
COMPUTE
COMP_1
COMP_2
COMP_3
COMP_4
COMP_5
COMP_X
CONCATENATE_FUNC
CONFIGURATION
CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONTROL_FOOTING

CONTROL_HEADING
CONVERTING
CORRESPONDING
COUNT
CRT
CURRENCY
CURRENT_DATE_FUNC
CURSOR
CYCLE
DATA
DATE
DAY
DAY_OF_WEEK
DE
DEBUGGING
DECIMAL_POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EBCDIC
ELSE
END
END_ACCEPT
END_ADD
END_CALL
END_COMPUTE

END_DELETE

END_DISPLAY

END_DIVIDE

END_EVALUATE

END_FUNCTION

END_IF

END_MULTIPLY

END_PERFORM

END_PROGRAM

END_READ

END_RETURN

END_REWRITE

END_SEARCH

END_START

END_STRING

END_SUBTRACT

END_UNSTRING

END_WRITE

ENTRY

ENVIRONMENT

ENVIRONMENT_NAME

ENVIRONMENT_VALUE

EOL

EOP

EOS

EQUAL

EQUALS

ERASE

ERROR

ESCAPE

EVALUATE

EVENT_STATUS

EXCEPTION

EXCLUSIVE

EXIT

EXTEND

EXTERNAL

FD

FILE_CONTROL
FILE_ID
FILLER
FINAL
FIRST
FOOTING
FOR
FOREGROUND_COLOR
FOREVER
FREE
FROM
FULL
FUNCTION
FUNCTION_ID
FUNCTION_NAME
GE
GENERATE
GIVING
GLOBAL
GO
GOBACK
GREATER
GROUP
HEADING
HIGHLIGHT
HIGH_VALUE
IDENTIFICATION
IF
IGNORE
IGNORING
IN
INDEX
INDEXED
INDICATE
INITIALIZE
INITIALIZED
INITIATE
INPUT

INPUT_OUTPUT
INSPECT
INTO
INTRINSIC
INVALID
INVALID_KEY
IS
I_O
I_O_CONTROL
JUSTIFIED
KEY
LABEL
LAST
LAST_DETAIL
LE
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE_COUNTER
LINE
LINES
LINKAGE
LITERAL
LOCALE
LOCALE_DT_FUNC
LOCAL_STORAGE
LOCK
LOWER_CASE_FUNC
LOWLIGHT
LOW_VALUE
MANUAL
MEMORY
MERGE
MINUS

MNEMONIC_NAME
MODE
MOVE
MULTIPLE
MULTIPLY
NATIONAL
NATIONAL_EDITED
NATIVE
NE
NEGATIVE
NEXT
NEXT_SENTENCE
NO
NOT
NOT_END
NOT_EOP
NOT_EXCEPTION
NOT_INVALID_KEY
NOT_OVERFLOW
NOT_SIZE_ERROR
NO_ADVANCING
NUMBER
NUMBERS
NUMERIC
NUMERIC_EDITED
NUMVALC_FUNC
OBJECT_COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION

OTHER
OUTPUT
OVERFLOW
OVERLINE
PACKED_DECIMAL
PADDING
PAGE
PAGE_FOOTING
PAGE_HEADING
PARAGRAPH
PERFORM
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRESENT
PREVIOUS
PRINTER
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM_ID
PROGRAM_NAME
PROGRAM_POINTER
PROMPT
QUOTE
RANDOM
RD
READ
RECORD
RECORDING
RECORDS
RECURSIVE
REDEFINES
REEL

REFERENCE
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
REPORT_FOOTING
REPORT_HEADING
REPOSITORY
REQUIRED
RESERVE
RETURN
RETURNING
REVERSE_FUNC
REVERSE_VIDEO
REWIND
REWRITE
RIGHT
ROLLBACK
ROUNDED
RUN
SAME
SCREEN
SCREEN_CONTROL
SCROLL
SD
SEARCH
SECTION
SECURE
SEGMENT_LIMIT
SELECT
SEMI_COLON
SENTENCE
SEPARATE

SEQUENCE

SEQUENTIAL

SET

SHARING

SIGN

SIGNED

SIGNED_INT

SIGNED_LONG

SIGNED_SHORT

SIZE

SIZE_ERROR

SORT

SORT_MERGE

SOURCE

SOURCE_COMPUTER

SPACE

SPECIAL_NAMES

STANDARD

STANDARD_1

STANDARD_2

START

STATUS

STOP

STRING

SUBSTITUTE_FUNC

SUBSTITUTE_CASE_FUNC

SUBTRACT

SUM

SUPPRESS

SYMBOLIC

SYNCHRONIZED

TALLYING

TAPE

TERMINATE

TEST

THAN

THEN

THRU

TIME
TIMES
TO
TOK_FALSE
TOK_FILE
TOK_INITIAL
TOK_NULL
TOK_TRUE
TOP
TRAILING
TRANSFORM
TRIM_FUNCTION
TYPE
UNDERLINE
UNIT
UNLOCK
UNSIGNED
UNSIGNED_INT
UNSIGNED_LONG
UNSIGNED_SHORT
UNSTRING
UNTIL
UP
UPDATE
UPON
UPON_ARGUMENT_NUMBER
UPON_COMMAND_LINE
UPON_ENVIRONMENT_NAME
UPON_ENVIRONMENT_VALUE
UPPER_CASE_FUNC
USAGE
USE
USING
VALUE
VARYING
WAIT
WHEN
WHEN_COMPILED_FUNC

WITH

WORD

WORDS

WORKING_STORAGE

WRITE

YYYYDDD

YYYYMMDD

ZERO

UNARY_SIGN

TOKEN_EOF

COPY

REPLACE

SUPPRESS

PRINTING

REPLACING

OFF

IN

OF

BY

EQEQ

TOKEN

TOKEN_EOF

COPY

REPLACE

SUPPRESS

PRINTING

REPLACING

OFF

IN

OF

BY

EQEQ

TOKEN

Definition at line 64 of file ppparse.c.

```
                {
  TOKEN_EOF = 0,
  COPY = 258,
  REPLACE = 259,
  SUPPRESS = 260,
  PRINTING = 261,
  REPLACING = 262,
  OFF = 263,
  IN = 264,
  OF = 265,
  BY = 266,
  EQEQ = 267,
  TOKEN = 268
};
```

7.11.4 Function Documentation

7.11.4.1 int yyparse (void)

7.11.5 Variable Documentation

7.11.5.1 int yychar

Definition at line 782 of file ppparse.c.

7.11.5.2 int yydebug

Definition at line 612 of file ppparse.c.

7.11.5.3 YYSTYPE yylval

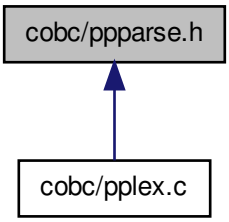
Definition at line 785 of file ppparse.c.

7.11.5.4 int yynerrs

Definition at line 788 of file ppparse.c.

7.12 cobc/ppparse.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- union YYSTYPE

Defines

- #define TOKEN_EOF 0
- #define COPY 258
- #define REPLACE 259
- #define SUPPRESS 260
- #define PRINTING 261
- #define REPLACING 262
- #define OFF 263
- #define IN 264
- #define OF 265
- #define BY 266
- #define EQEQ 267
- #define TOKEN 268
- #define yystype YYSTYPE
- #define YYSTYPE_IS_DECLARED 1
- #define YYSTYPE_IS_TRIVIAL 1

Typedefs

- typedef union YYSTYPE YYSTYPE

Enumerations

- enum yytokentype {

  TOKEN_EOF = 0, ACCEPT = 258, ACCESS = 259, ADD = 260,

  ADDRESS = 261, ADVANCING = 262, AFTER = 263, ALL = 264,

  ALLOCATE = 265, ALPHABET = 266, ALPHABETIC = 267, ALPHABETIC_-
  LOWER = 268,

  ALPHABETIC_UPPER = 269, ALPHANUMERIC = 270, ALPHANUMERIC_EDITED
  = 271, ALSO = 272,

  ALTER = 273, ALTERNATE = 274, AND = 275, ANY = 276,

  ARE = 277, AREA = 278, ARGUMENT_NUMBER = 279, ARGUMENT_VALUE
  = 280,

  AS = 281, ASCENDING = 282, ASSIGN = 283, AT = 284,

  AUTO = 285, AUTOMATIC = 286, BACKGROUND_COLOR = 287, BASED = 288,

  BEFORE = 289, BELL = 290, BINARY = 291, BINARY_C_LONG = 292,

  BINARY_CHAR = 293, BINARY_DOUBLE = 294, BINARY_LONG = 295, BINARY_-
  SHORT = 296,

  BLANK = 297, BLANK_LINE = 298, BLANK_SCREEN = 299, BLINK = 300,

  BLOCK = 301, BOTTOM = 302, BY = 303, BYTE_LENGTH = 304,

  CALL = 305, CANCEL = 306, CH = 307, CHAINING = 308,

  CHARACTER = 309, CHARACTERS = 310, CLASS = 311, CLOSE = 312,

  CODE = 313, CODE_SET = 314, COLLATING = 315, COL = 316,

  COLS = 317, COLUMN = 318, COLUMNS = 319, COMMA = 320,

  COMMAND_LINE = 321, COMMA_DELIM = 322, COMMIT = 323, COMMON =
  324,

  COMP = 325, COMPUTE = 326, COMP_1 = 327, COMP_2 = 328,

  COMP_3 = 329, COMP_4 = 330, COMP_5 = 331, COMP_X = 332,

  CONCATENATE_FUNC = 333, CONFIGURATION = 334, CONSTANT = 335,
  CONTAINS = 336,

  CONTENT = 337, CONTINUE = 338, CONTROL = 339, CONTROLS = 340,

  CONTROL_FOOTING = 341, CONTROL_HEADING = 342, CONVERTING =
  343, CORRESPONDING = 344,

  COUNT = 345, CRT = 346, CURRENCY = 347, CURRENT_DATE_FUNC = 348,

  CURSOR = 349, CYCLE = 350, DATA = 351, DATE = 352,

  DAY = 353, DAY_OF_WEEK = 354, DE = 355, DEBUGGING = 356,

  DECIMAL_POINT = 357, DECLARATIVES = 358, DEFAULT = 359, DELETE =
  360,

  DELIMITED = 361, DELIMITER = 362, DEPENDING = 363, DESCENDING =
  364,

  DETAIL = 365, DISK = 366, DISPLAY = 367, DIVIDE = 368,

DIVISION = 369, DOWN = 370, DUPLICATES = 371, DYNAMIC = 372,

EBCDIC = 373, ELSE = 374, END = 375, END_ACCEPT = 376,

END_ADD = 377, END_CALL = 378, END_COMPUTE = 379, END_DELETE = 380,

END_DISPLAY = 381, END_DIVIDE = 382, END_EVALUATE = 383, END_-FUNCTION = 384,

END_IF = 385, END_MULTIPLY = 386, END_PERFORM = 387, END_PROGRAM = 388,

END_READ = 389, END_RETURN = 390, END_REWRITE = 391, END_SEARCH = 392,

END_START = 393, END_STRING = 394, END_SUBTRACT = 395, END_UNSTRING = 396,

END_WRITE = 397, ENTRY = 398, ENVIRONMENT = 399, ENVIRONMENT_-NAME = 400,

ENVIRONMENT_VALUE = 401, EOL = 402, EOP = 403, EOS = 404,

EQUAL = 405, EQUALS = 406, ERASE = 407, ERROR = 408,

ESCAPE = 409, EVALUATE = 410, EVENT_STATUS = 411, EXCEPTION = 412,

EXCLUSIVE = 413, EXIT = 414, EXTEND = 415, EXTERNAL = 416,

FD = 417, FILE_CONTROL = 418, FILE_ID = 419, FILLER = 420,

FINAL = 421, FIRST = 422, FOOTING = 423, FOR = 424,

FOREGROUND_COLOR = 425, FOREVER = 426, FREE = 427, FROM = 428,

FULL = 429, FUNCTION = 430, FUNCTION_ID = 431, FUNCTION_NAME = 432,

GE = 433, GENERATE = 434, GIVING = 435, GLOBAL = 436,

GO = 437, GOBACK = 438, GREATER = 439, GROUP = 440,

HEADING = 441, HIGHLIGHT = 442, HIGH_VALUE = 443, IDENTIFICATION = 444,

IF = 445, IGNORE = 446, IGNORING = 447, IN = 448,

INDEX = 449, INDEXED = 450, INDICATE = 451, INITIALIZE = 452,

INITIALIZED = 453, INITIATE = 454, INPUT = 455, INPUT_OUTPUT = 456,

INSPECT = 457, INTO = 458, INTRINSIC = 459, INVALID = 460,

INVALID_KEY = 461, IS = 462, I_O = 463, I_O_CONTROL = 464,

JUSTIFIED = 465, KEY = 466, LABEL = 467, LAST = 468,

LAST_DETAIL = 469, LE = 470, LEADING = 471, LEFT = 472,

LENGTH = 473, LESS = 474, LIMIT = 475, LIMITS = 476,

LINAGE = 477, LINAGE_COUNTER = 478, LINE = 479, LINES = 480,

LINKAGE = 481, LITERAL = 482, LOCALE = 483, LOCALE_DT_FUNC = 484,

LOCAL_STORAGE = 485, LOCK = 486, LOWER_CASE_FUNC = 487, LOW-LIGHT = 488,

LOW_VALUE = 489, MANUAL = 490, MEMORY = 491, MERGE = 492,

MINUS = 493, MNEMONIC_NAME = 494, MODE = 495, MOVE = 496,

MULTIPLE = 497, MULTIPLY = 498, NATIONAL = 499, NATIONAL_EDITED = 500,

NATIVE = 501, NE = 502, NEGATIVE = 503, NEXT = 504,

NEXT_SENTENCE = 505, NO = 506, NOT = 507, NOT_END = 508,

NOT_EOP = 509, NOT_EXCEPTION = 510, NOT_INVALID_KEY = 511, NOT_OVERFLOW = 512,

NOT_SIZE_ERROR = 513, NO_ADVANCING = 514, NUMBER = 515, NUMBERS = 516,

NUMERIC = 517, NUMERIC_EDITED = 518, NUMVALC_FUNC = 519, OBJECT_COMPUTER = 520,

OCCURS = 521, OF = 522, OFF = 523, OMITTED = 524,

ON = 525, ONLY = 526, OPEN = 527, OPTIONAL = 528,

OR = 529, ORDER = 530, ORGANIZATION = 531, OTHER = 532,

OUTPUT = 533, OVERFLOW = 534, OVERLINE = 535, PACKED_DECIMAL = 536,

PADDING = 537, PAGE = 538, PAGE_FOOTING = 539, PAGE_HEADING = 540,

PARAGRAPH = 541, PERFORM = 542, PICTURE = 543, PLUS = 544,

POINTER = 545, POSITION = 546, POSITIVE = 547, PRESENT = 548,

PREVIOUS = 549, PRINTER = 550, PRINTING = 551, PROCEDURE = 552,

PROCEDURES = 553, PROCEED = 554, PROGRAM = 555, PROGRAM_ID = 556,

PROGRAM_NAME = 557, PROGRAM_POINTER = 558, PROMPT = 559, QUOTE = 560,

RANDOM = 561, RD = 562, READ = 563, RECORD = 564,

RECORDING = 565, RECORDS = 566, RECURSIVE = 567, REDEFINES = 568,

REEL = 569, REFERENCE = 570, RELATIVE = 571, RELEASE = 572,

REMAINDER = 573, REMOVAL = 574, RENAMES = 575, REPLACING = 576,

REPORT = 577, REPORTING = 578, REPORTS = 579, REPORT_FOOTING = 580,

REPORT_HEADING = 581, REPOSITORY = 582, REQUIRED = 583, RESERVE = 584,

RETURN = 585, RETURNING = 586, REVERSE_FUNC = 587, REVERSE_VIDEO = 588,

REWIND = 589, REWRITE = 590, RIGHT = 591, ROLLBACK = 592,

ROUNDED = 593, RUN = 594, SAME = 595, SCREEN = 596,

SCREEN_CONTROL = 597, SCROLL = 598, SD = 599, SEARCH = 600,

SECTION = 601, SECURE = 602, SEGMENT_LIMIT = 603, SELECT = 604,

SEMI_COLON = 605, SENTENCE = 606, SEPARATE = 607, SEQUENCE = 608,

SEQUENTIAL = 609, SET = 610, SHARING = 611, SIGN = 612,

SIGNED = 613, SIGNED_INT = 614, SIGNED_LONG = 615, SIGNED_SHORT = 616,

SIZE = 617, SIZE_ERROR = 618, SORT = 619, SORT_MERGE = 620,

SOURCE = 621, SOURCE_COMPUTER = 622, SPACE = 623, SPECIAL_NAMES = 624,

STANDARD = 625, STANDARD_1 = 626, STANDARD_2 = 627, START = 628,

STATUS = 629, STOP = 630, STRING = 631, SUBSTITUTE_FUNC = 632,

SUBSTITUTE_CASE_FUNC = 633, SUBTRACT = 634, SUM = 635, SUPPRESS = 636,

SYMBOLIC = 637, SYNCHRONIZED = 638, TALLYING = 639, TAPE = 640,

TERMINATE = 641, TEST = 642, THAN = 643, THEN = 644,

THRU = 645, TIME = 646, TIMES = 647, TO = 648,

TOK_FALSE = 649, TOK_FILE = 650, TOK_INITIAL = 651, TOK_NULL = 652,

TOK_TRUE = 653, TOP = 654, TRAILING = 655, TRANSFORM = 656,

TRIM_FUNCTION = 657, TYPE = 658, UNDERLINE = 659, UNIT = 660,

UNLOCK = 661, UNSIGNED = 662, UNSIGNED_INT = 663, UNSIGNED_LONG = 664,

UNSIGNED_SHORT = 665, UNSTRING = 666, UNTIL = 667, UP = 668,

UPDATE = 669, UPON = 670, UPON_ARGUMENT_NUMBER = 671, UPON_COMMAND_LINE = 672,

UPON_ENVIRONMENT_NAME = 673, UPON_ENVIRONMENT_VALUE = 674, UPPER_CASE_FUNC = 675, USAGE = 676,

USE = 677, USING = 678, VALUE = 679, VARYING = 680,

WAIT = 681, WHEN = 682, WHEN_COMPILED_FUNC = 683, WITH = 684,

WORD = 685, WORDS = 686, WORKING_STORAGE = 687, WRITE = 688,

YYYYDDD = 689, YYYYMMDD = 690, ZERO = 691, UNARY_SIGN = 692,

TOKEN_EOF = 0, ACCEPT = 258, ACCESS = 259, ADD = 260,

ADDRESS = 261, ADVANCING = 262, AFTER = 263, ALL = 264,

ALLOCATE = 265, ALPHABET = 266, ALPHABETIC = 267, ALPHABETIC_LOWER = 268,

ALPHABETIC_UPPER = 269, ALPHANUMERIC = 270, ALPHANUMERIC_EDITED = 271, ALSO = 272,

ALTER = 273, ALTERNATE = 274, AND = 275, ANY = 276,

ARE = 277, AREA = 278, ARGUMENT_NUMBER = 279, ARGUMENT_VALUE = 280,

AS = 281, ASCENDING = 282, ASSIGN = 283, AT = 284,

AUTO = 285, AUTOMATIC = 286, BACKGROUND_COLOR = 287, BASED = 288,

BEFORE = 289, BELL = 290, BINARY = 291, BINARY_C_LONG = 292,

BINARY_CHAR = 293, BINARY_DOUBLE = 294, BINARY_LONG = 295, BINARY_SHORT = 296,

BLANK = 297, BLANK_LINE = 298, BLANK_SCREEN = 299, BLINK = 300,

BLOCK = 301, BOTTOM = 302, BY = 303, BYTE_LENGTH = 304,

CALL = 305, CANCEL = 306, CH = 307, CHAINING = 308,

CHARACTER = 309, CHARACTERS = 310, CLASS = 311, CLOSE = 312,

CODE = 313, CODE_SET = 314, COLLATING = 315, COL = 316,

COLS = 317, COLUMN = 318, COLUMNS = 319, COMMA = 320,

COMMAND_LINE = 321, COMMA_DELIM = 322, COMMIT = 323, COMMON = 324,

COMP = 325, COMPUTE = 326, COMP_1 = 327, COMP_2 = 328,

COMP_3 = 329, COMP_4 = 330, COMP_5 = 331, COMP_X = 332,

CONCATENATE_FUNC = 333, CONFIGURATION = 334, CONSTANT = 335, CONTAINS = 336,

CONTENT = 337, CONTINUE = 338, CONTROL = 339, CONTROLS = 340,

CONTROL_FOOTING = 341, CONTROL_HEADING = 342, CONVERTING = 343, CORRESPONDING = 344,

COUNT = 345, CRT = 346, CURRENCY = 347, CURRENT_DATE_FUNC = 348,

CURSOR = 349, CYCLE = 350, DATA = 351, DATE = 352,

DAY = 353, DAY_OF_WEEK = 354, DE = 355, DEBUGGING = 356,

DECIMAL_POINT = 357, DECLARATIVES = 358, DEFAULT = 359, DELETE = 360,

DELIMITED = 361, DELIMITER = 362, DEPENDING = 363, DESCENDING = 364,

DETAIL = 365, DISK = 366, DISPLAY = 367, DIVIDE = 368,

DIVISION = 369, DOWN = 370, DUPLICATES = 371, DYNAMIC = 372,

EBCDIC = 373, ELSE = 374, END = 375, END_ACCEPT = 376,

END_ADD = 377, END_CALL = 378, END_COMPUTE = 379, END_DELETE = 380,

END_DISPLAY = 381, END_DIVIDE = 382, END_EVALUATE = 383, END_-FUNCTION = 384,

END_IF = 385, END_MULTIPLY = 386, END_PERFORM = 387, END_PROGRAM = 388,

END_READ = 389, END_RETURN = 390, END_REWRITE = 391, END_SEARCH = 392,

END_START = 393, END_STRING = 394, END_SUBTRACT = 395, END_UNSTRING = 396,

END_WRITE = 397, ENTRY = 398, ENVIRONMENT = 399, ENVIRONMENT_-NAME = 400,

ENVIRONMENT_VALUE = 401, EOL = 402, EOP = 403, EOS = 404,

EQUAL = 405, EQUALS = 406, ERASE = 407, ERROR = 408,

ESCAPE = 409, EVALUATE = 410, EVENT_STATUS = 411, EXCEPTION = 412,

EXCLUSIVE = 413, EXIT = 414, EXTEND = 415, EXTERNAL = 416,

FD = 417, FILE_CONTROL = 418, FILE_ID = 419, FILLER = 420,

FINAL = 421, FIRST = 422, FOOTING = 423, FOR = 424,

FOREGROUND_COLOR = 425, FOREVER = 426, FREE = 427, FROM = 428,

FULL = 429, FUNCTION = 430, FUNCTION_ID = 431, FUNCTION_NAME = 432,

GE = 433, GENERATE = 434, GIVING = 435, GLOBAL = 436,

GO = 437, GOBACK = 438, GREATER = 439, GROUP = 440,

HEADING = 441, HIGHLIGHT = 442, HIGH_VALUE = 443, IDENTIFICATION = 444,

IF = 445, IGNORE = 446, IGNORING = 447, IN = 448,

INDEX = 449, INDEXED = 450, INDICATE = 451, INITIALIZE = 452,

INITIALIZED = 453, INITIATE = 454, INPUT = 455, INPUT_OUTPUT = 456,

INSPECT = 457, INTO = 458, INTRINSIC = 459, INVALID = 460,

INVALID_KEY = 461, IS = 462, I_O = 463, I_O_CONTROL = 464,

JUSTIFIED = 465, KEY = 466, LABEL = 467, LAST = 468,

LAST_DETAIL = 469, LE = 470, LEADING = 471, LEFT = 472,

LENGTH = 473, LESS = 474, LIMIT = 475, LIMITS = 476,

LINAGE = 477, LINAGE_COUNTER = 478, LINE = 479, LINES = 480,

LINKAGE = 481, LITERAL = 482, LOCALE = 483, LOCALE_DT_FUNC = 484,

LOCAL_STORAGE = 485, LOCK = 486, LOWER_CASE_FUNC = 487, LOW-LIGHT = 488,

LOW_VALUE = 489, MANUAL = 490, MEMORY = 491, MERGE = 492,

MINUS = 493, MNEMONIC_NAME = 494, MODE = 495, MOVE = 496,

MULTIPLE = 497, MULTIPLY = 498, NATIONAL = 499, NATIONAL_EDITED = 500,

NATIVE = 501, NE = 502, NEGATIVE = 503, NEXT = 504,

NEXT_SENTENCE = 505, NO = 506, NOT = 507, NOT_END = 508,

NOT_EOP = 509, NOT_EXCEPTION = 510, NOT_INVALID_KEY = 511, NOT_-OVERFLOW = 512,

NOT_SIZE_ERROR = 513, NO_ADVANCING = 514, NUMBER = 515, NUM-BERS = 516,

NUMERIC = 517, NUMERIC_EDITED = 518, NUMVALC_FUNC = 519, OBJECT_-COMPUTER = 520,

OCCURS = 521, OF = 522, OFF = 523, OMITTED = 524,

ON = 525, ONLY = 526, OPEN = 527, OPTIONAL = 528,

OR = 529, ORDER = 530, ORGANIZATION = 531, OTHER = 532,

OUTPUT = 533, OVERFLOW = 534, OVERLINE = 535, PACKED_DECIMAL = 536,

PADDING = 537, PAGE = 538, PAGE_FOOTING = 539, PAGE_HEADING = 540,

PARAGRAPH = 541, PERFORM = 542, PICTURE = 543, PLUS = 544,

POINTER = 545, POSITION = 546, POSITIVE = 547, PRESENT = 548,

PREVIOUS = 549, PRINTER = 550, PRINTING = 551, PROCEDURE = 552,

PROCEDURES = 553, PROCEED = 554, PROGRAM = 555, PROGRAM_ID = 556,

PROGRAM_NAME = 557, PROGRAM_POINTER = 558, PROMPT = 559, QUOTE = 560,

RANDOM = 561, RD = 562, READ = 563, RECORD = 564,

RECORDING = 565, RECORDS = 566, RECURSIVE = 567, REDEFINES = 568,

REEL = 569, REFERENCE = 570, RELATIVE = 571, RELEASE = 572,

REMAINDER = 573, REMOVAL = 574, RENAMES = 575, REPLACING = 576,

REPORT = 577, REPORTING = 578, REPORTS = 579, REPORT_FOOTING = 580,

REPORT_HEADING = 581, REPOSITORY = 582, REQUIRED = 583, RESERVE = 584,

RETURN = 585, RETURNING = 586, REVERSE_FUNC = 587, REVERSE_-VIDEO = 588,

REWIND = 589, REWRITE = 590, RIGHT = 591, ROLLBACK = 592,

ROUNDED = 593, RUN = 594, SAME = 595, SCREEN = 596,

SCREEN_CONTROL = 597, SCROLL = 598, SD = 599, SEARCH = 600,

SECTION = 601, SECURE = 602, SEGMENT_LIMIT = 603, SELECT = 604,

SEMI_COLON = 605, SENTENCE = 606, SEPARATE = 607, SEQUENCE = 608,

SEQUENTIAL = 609, SET = 610, SHARING = 611, SIGN = 612,

SIGNED = 613, SIGNED_INT = 614, SIGNED_LONG = 615, SIGNED_SHORT = 616,

SIZE = 617, SIZE_ERROR = 618, SORT = 619, SORT_MERGE = 620,

SOURCE = 621, SOURCE_COMPUTER = 622, SPACE = 623, SPECIAL_NAMES = 624,

STANDARD = 625, STANDARD_1 = 626, STANDARD_2 = 627, START = 628,

STATUS = 629, STOP = 630, STRING = 631, SUBSTITUTE_FUNC = 632,

SUBSTITUTE_CASE_FUNC = 633, SUBTRACT = 634, SUM = 635, SUPPRESS = 636,

SYMBOLIC = 637, SYNCHRONIZED = 638, TALLYING = 639, TAPE = 640,

TERMINATE = 641, TEST = 642, THAN = 643, THEN = 644,

THRU = 645, TIME = 646, TIMES = 647, TO = 648,

TOK_FALSE = 649, TOK_FILE = 650, TOK_INITIAL = 651, TOK_NULL = 652,

TOK_TRUE = 653, TOP = 654, TRAILING = 655, TRANSFORM = 656,

TRIM_FUNCTION = 657, TYPE = 658, UNDERLINE = 659, UNIT = 660,

UNLOCK = 661, UNSIGNED = 662, UNSIGNED_INT = 663, UNSIGNED_LONG = 664,

UNSIGNED_SHORT = 665, UNSTRING = 666, UNTIL = 667, UP = 668,

UPDATE = 669, UPON = 670, UPON_ARGUMENT_NUMBER = 671, UPON_-
COMMAND_LINE = 672,

UPON_ENVIRONMENT_NAME = 673, UPON_ENVIRONMENT_VALUE = 674,
UPPER_CASE_FUNC = 675, USAGE = 676,

USE = 677, USING = 678, VALUE = 679, VARYING = 680,

WAIT = 681, WHEN = 682, WHEN_COMPILED_FUNC = 683, WITH = 684,

WORD = 685, WORDS = 686, WORKING_STORAGE = 687, WRITE = 688,

YYYYDDD = 689, YYYYMMDD = 690, ZERO = 691, UNARY_SIGN = 692,

TOKEN_EOF = 0, COPY = 258, REPLACE = 259, SUPPRESS = 260,

PRINTING = 261, REPLACING = 262, OFF = 263, IN = 264,

OF = 265, BY = 266, EQEQ = 267, TOKEN = 268,

TOKEN_EOF = 0, COPY = 258, REPLACE = 259, SUPPRESS = 260,

PRINTING = 261, REPLACING = 262, OFF = 263, IN = 264,

OF = 265, BY = 266, EQEQ = 267, TOKEN = 268 }

Variables

- YYSTYPE pplval

7.12.1 Define Documentation

7.12.1.1 #define BY 266

Definition at line 55 of file ppparse.h.

7.12.1.2 #define COPY 258

Definition at line 47 of file ppparse.h.

7.12.1.3 #define EQEQ 267

Definition at line 56 of file ppparse.h.

7.12.1.4 #define IN 264

Definition at line 53 of file ppparse.h.

7.12.1.5 #define OF 265

Definition at line 54 of file ppparse.h.

7.12.1.6 #define OFF 263

Definition at line 52 of file ppparse.h.

7.12.1.7 #define PRINTING 261

Definition at line 50 of file ppparse.h.

7.12.1.8 #define REPLACE 259

Definition at line 48 of file ppparse.h.

7.12.1.9 #define REPLACING 262

Definition at line 51 of file ppparse.h.

7.12.1.10 #define SUPPRESS 260

Definition at line 49 of file ppparse.h.

7.12.1.11 #define TOKEN 268

Definition at line 57 of file ppparse.h.

7.12.1.12 #define TOKEN_EOF 0

Definition at line 46 of file ppparse.h.

7.12.1.13 #define yystype YYSTYPE

Definition at line 71 of file ppparse.h.

7.12.1.14 #define YYSTYPE_IS_DECLARED 1

Definition at line 72 of file ppparse.h.

7.12.1.15 #define YYSTYPE_IS_TRIVIAL 1

Definition at line 73 of file ppparse.h.

7.12.2 Typedef Documentation

7.12.2.1 typedef union YYSTYPE YYSTYPE

7.12.3 Enumeration Type Documentation

7.12.3.1 enum yytokentype

Enumerator:

TOKEN_EOF
ACCEPT
ACCESS
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALLOCATE
ALPHABET
ALPHABETIC
ALPHABETIC_LOWER
ALPHABETIC_UPPER
ALPHANUMERIC
ALPHANUMERIC_EDITED
ALSO
ALTER
ALTERNATE
AND
ANY
ARE
AREA
ARGUMENT_NUMBER
ARGUMENT_VALUE
AS
ASCENDING
ASSIGN
AT
AUTO
AUTOMATIC
BACKGROUND_COLOR

BASED
BEFORE
BELL
BINARY
BINARY_C_LONG
BINARY_CHAR
BINARY_DOUBLE
BINARY_LONG
BINARY_SHORT
BLANK
BLANK_LINE
BLANK_SCREEN
BLINK
BLOCK
BOTTOM
BY
BYTE_LENGTH
CALL
CANCEL
CH
CHAINING
CHARACTER
CHARACTERS
CLASS
CLOSE
CODE
CODE_SET
COLLATING
COL
COLS
COLUMN
COLUMNS
COMMA
COMMAND_LINE
COMMA_DELIM
COMMIT
COMMON
COMP

COMPUTE
COMP_1
COMP_2
COMP_3
COMP_4
COMP_5
COMP_X
CONCATENATE_FUNC
CONFIGURATION
CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONTROL_FOOTING
CONTROL_HEADING
CONVERTING
CORRESPONDING
COUNT
CRT
CURRENCY
CURRENT_DATE_FUNC
CURSOR
CYCLE
DATA
DATE
DAY
DAY_OF_WEEK
DE
DEBUGGING
DECIMAL_POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDING

DESCENDING

DETAIL

DISK

DISPLAY

DIVIDE

DIVISION

DOWN

DUPLICATES

DYNAMIC

EBCDIC

ELSE

END

END_ACCEPT

END_ADD

END_CALL

END_COMPUTE

END_DELETE

END_DISPLAY

END_DIVIDE

END_EVALUATE

END_FUNCTION

END_IF

END_MULTIPLY

END_PERFORM

END_PROGRAM

END_READ

END_RETURN

END_REWRITE

END_SEARCH

END_START

END_STRING

END_SUBTRACT

END_UNSTRING

END_WRITE

ENTRY

ENVIRONMENT

ENVIRONMENT_NAME

ENVIRONMENT_VALUE

EOL
EOP
EOS
EQUAL
EQUALS
ERASE
ERROR
ESCAPE
EVALUATE
EVENT_STATUS
EXCEPTION
EXCLUSIVE
EXIT
EXTEND
EXTERNAL
FD
FILE_CONTROL
FILE_ID
FILLER
FINAL
FIRST
FOOTING
FOR
FOREGROUND_COLOR
FOREVER
FREE
FROM
FULL
FUNCTION
FUNCTION_ID
FUNCTION_NAME
GE
GENERATE
GIVING
GLOBAL
GO
GOBACK
GREATER

GROUP
HEADING
HIGHLIGHT
HIGH_VALUE
IDENTIFICATION
IF
IGNORE
IGNORING
IN
INDEX
INDEXED
INDICATE
INITIALIZE
INITIALIZED
INITIATE
INPUT
INPUT_OUTPUT
INSPECT
INTO
INTRINSIC
INVALID
INVALID_KEY
IS
I_O
I_O_CONTROL
JUSTIFIED
KEY
LABEL
LAST
LAST_DETAIL
LE
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE

LINAGE_COUNTER
LINE
LINES
LINKAGE
LITERAL
LOCALE
LOCALE_DT_FUNC
LOCAL_STORAGE
LOCK
LOWER_CASE_FUNC
LOWLIGHT
LOW_VALUE
MANUAL
MEMORY
MERGE
MINUS
MNEMONIC_NAME
MODE
MOVE
MULTIPLE
MULTIPLY
NATIONAL
NATIONAL_EDITED
NATIVE
NE
NEGATIVE
NEXT
NEXT_SENTENCE
NO
NOT
NOT_END
NOT_EOP
NOT_EXCEPTION
NOT_INVALID_KEY
NOT_OVERFLOW
NOT_SIZE_ERROR
NO_ADVANCING
NUMBER

NUMBERS
NUMERIC
NUMERIC_EDITED
NUMVALC_FUNC
OBJECT_COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
OVERLINE
PACKED_DECIMAL
PADDING
PAGE
PAGE_FOOTING
PAGE_HEADING
PARAGRAPH
PERFORM
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRESENT
PREVIOUS
PRINTER
PRINTING
PROCEDURE
PROCEDURES

PROCEED
PROGRAM
PROGRAM_ID
PROGRAM_NAME
PROGRAM_POINTER
PROMPT
QUOTE
RANDOM
RD
READ
RECORD
RECORDING
RECORDS
RECURSIVE
REDEFINES
REEL
REFERENCE
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
REPORT_FOOTING
REPORT_HEADING
REPOSITORY
REQUIRED
RESERVE
RETURN
RETURNING
REVERSE_FUNC
REVERSE_VIDEO
REWIND
REWRITE
RIGHT

ROLLBACK
ROUNDED
RUN
SAME
SCREEN
SCREEN_CONTROL
SCROLL
SD
SEARCH
SECTION
SECURE
SEGMENT_LIMIT
SELECT
SEMI_COLON
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHARING
SIGN
SIGNED
SIGNED_INT
SIGNED_LONG
SIGNED_SHORT
SIZE
SIZE_ERROR
SORT
SORT_MERGE
SOURCE
SOURCE_COMPUTER
SPACE
SPECIAL_NAMES
STANDARD
STANDARD_1
STANDARD_2
START
STATUS

STOP
STRING
SUBSTITUTE_FUNC
SUBSTITUTE_CASE_FUNC
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNCHRONIZED
TALLYING
TAPE
TERMINATE
TEST
THAN
THEN
THRU
TIME
TIMES
TO
TOK_FALSE
TOK_FILE
TOK_INITIAL
TOK_NULL
TOK_TRUE
TOP
TRAILING
TRANSFORM
TRIM_FUNCTION
TYPE
UNDERLINE
UNIT
UNLOCK
UNSIGNED
UNSIGNED_INT
UNSIGNED_LONG
UNSIGNED_SHORT
UNSTRING
UNTIL

UP
UPDATE
UPON
UPON_ARGUMENT_NUMBER
UPON_COMMAND_LINE
UPON_ENVIRONMENT_NAME
UPON_ENVIRONMENT_VALUE
UPPER_CASE_FUNC
USAGE
USE
USING
VALUE
VARYING
WAIT
WHEN
WHEN_COMPILED_FUNC
WITH
WORD
WORDS
WORKING_STORAGE
WRITE
YYYYDDD
YYYYMMDD
ZERO
UNARY_SIGN
TOKEN_EOF
ACCEPT
ACCESS
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALLOCATE
ALPHABET
ALPHABETIC
ALPHABETIC_LOWER
ALPHABETIC_UPPER

ALPHANUMERIC

ALPHANUMERIC_EDITED

ALSO

ALTER

ALTERNATE

AND

ANY

ARE

AREA

ARGUMENT_NUMBER

ARGUMENT_VALUE

AS

ASCENDING

ASSIGN

AT

AUTO

AUTOMATIC

BACKGROUND_COLOR

BASED

BEFORE

BELL

BINARY

BINARY_C_LONG

BINARY_CHAR

BINARY_DOUBLE

BINARY_LONG

BINARY_SHORT

BLANK

BLANK_LINE

BLANK_SCREEN

BLINK

BLOCK

BOTTOM

BY

BYTE_LENGTH

CALL

CANCEL

CH

CHAINING
CHARACTER
CHARACTERS
CLASS
CLOSE
CODE
CODE_SET
COLLATING
COL
COLS
COLUMN
COLUMNS
COMMA
COMMAND_LINE
COMMA_DELIM
COMMIT
COMMON
COMP
COMPUTE
COMP_1
COMP_2
COMP_3
COMP_4
COMP_5
COMP_X
CONCATENATE_FUNC
CONFIGURATION
CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONTROL_FOOTING
CONTROL_HEADING
CONVERTING
CORRESPONDING
COUNT

CRT
CURRENCY
CURRENT_DATE_FUNC
CURSOR
CYCLE
DATA
DATE
DAY
DAY_OF_WEEK
DE
DEBUGGING
DECIMAL_POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EBCDIC
ELSE
END
END_ACCEPT
END_ADD
END_CALL
END_COMPUTE
END_DELETE
END_DISPLAY
END_DIVIDE
END_EVALUATE

END_FUNCTION
END_IF
END_MULTIPLY
END_PERFORM
END_PROGRAM
END_READ
END_RETURN
END_REWRITE
END_SEARCH
END_START
END_STRING
END_SUBTRACT
END_UNSTRING
END_WRITE
ENTRY
ENVIRONMENT
ENVIRONMENT_NAME
ENVIRONMENT_VALUE
EOL
EOP
EOS
EQUAL
EQUALS
ERASE
ERROR
ESCAPE
EVALUATE
EVENT_STATUS
EXCEPTION
EXCLUSIVE
EXIT
EXTEND
EXTERNAL
FD
FILE_CONTROL
FILE_ID
FILLER
FINAL

FIRST
FOOTING
FOR
FOREGROUND_COLOR
FOREVER
FREE
FROM
FULL
FUNCTION
FUNCTION_ID
FUNCTION_NAME
GE
GENERATE
GIVING
GLOBAL
GO
GOBACK
GREATER
GROUP
HEADING
HIGHLIGHT
HIGH_VALUE
IDENTIFICATION
IF
IGNORE
IGNORING
IN
INDEX
INDEXED
INDICATE
INITIALIZE
INITIALIZED
INITIATE
INPUT
INPUT_OUTPUT
INSPECT
INTO
INTRINSIC

INVALID
INVALID_KEY
IS
I_O
I_O_CONTROL
JUSTIFIED
KEY
LABEL
LAST
LAST_DETAIL
LE
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE_COUNTER
LINE
LINES
LINKAGE
LITERAL
LOCALE
LOCALE_DT_FUNC
LOCAL_STORAGE
LOCK
LOWER_CASE_FUNC
LOWLIGHT
LOW_VALUE
MANUAL
MEMORY
MERGE
MINUS
MNEMONIC_NAME
MODE
MOVE
MULTIPLE

MULTIPLY
NATIONAL
NATIONAL_EDITED
NATIVE
NE
NEGATIVE
NEXT
NEXT_SENTENCE
NO
NOT
NOT_END
NOT_EOP
NOT_EXCEPTION
NOT_INVALID_KEY
NOT_OVERFLOW
NOT_SIZE_ERROR
NO_ADVANCING
NUMBER
NUMBERS
NUMERIC
NUMERIC_EDITED
NUMVALC_FUNC
OBJECT_COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
OVERLINE

PACKED_DECIMAL
PADDING
PAGE
PAGE_FOOTING
PAGE_HEADING
PARAGRAPH
PERFORM
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRESENT
PREVIOUS
PRINTER
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM_ID
PROGRAM_NAME
PROGRAM_POINTER
PROMPT
QUOTE
RANDOM
RD
READ
RECORD
RECORDING
RECORDS
RECURSIVE
REDEFINES
REEL
REFERENCE
RELATIVE
RELEASE
REMAINDER

REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
REPORT_FOOTING
REPORT_HEADING
REPOSITORY
REQUIRED
RESERVE
RETURN
RETURNING
REVERSE_FUNC
REVERSE_VIDEO
REWIND
REWRITE
RIGHT
ROLLBACK
ROUNDED
RUN
SAME
SCREEN
SCREEN_CONTROL
SCROLL
SD
SEARCH
SECTION
SECURE
SEGMENT_LIMIT
SELECT
SEMI_COLON
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHARING

SIGN
SIGNED
SIGNED_INT
SIGNED_LONG
SIGNED_SHORT
SIZE
SIZE_ERROR
SORT
SORT_MERGE
SOURCE
SOURCE_COMPUTER
SPACE
SPECIAL_NAMES
STANDARD
STANDARD_1
STANDARD_2
START
STATUS
STOP
STRING
SUBSTITUTE_FUNC
SUBSTITUTE_CASE_FUNC
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNCHRONIZED
TALLYING
TAPE
TERMINATE
TEST
THAN
THEN
THRU
TIME
TIMES
TO
TOK_FALSE

TOK_FILE
TOK_INITIAL
TOK_NULL
TOK_TRUE
TOP
TRAILING
TRANSFORM
TRIM_FUNCTION
TYPE
UNDERLINE
UNIT
UNLOCK
UNSIGNED
UNSIGNED_INT
UNSIGNED_LONG
UNSIGNED_SHORT
UNSTRING
UNTIL
UP
UPDATE
UPON
UPON_ARGUMENT_NUMBER
UPON_COMMAND_LINE
UPON_ENVIRONMENT_NAME
UPON_ENVIRONMENT_VALUE
UPPER_CASE_FUNC
USAGE
USE
USING
VALUE
VARYING
WAIT
WHEN
WHEN_COMPILED_FUNC
WITH
WORD
WORDS
WORKING_STORAGE

WRITE
YYYYDDD
YYYYMMDD
ZERO
UNARY_SIGN
TOKEN_EOF
COPY
REPLACE
SUPPRESS
PRINTING
REPLACING
OFF
IN
OF
BY
EQEQ
TOKEN
TOKEN_EOF
COPY
REPLACE
SUPPRESS
PRINTING
REPLACING
OFF
IN
OF
BY
EQEQ
TOKEN

Definition at line 31 of file ppparse.h.

```
                {
  TOKEN_EOF = 0,
  COPY = 258,
  REPLACE = 259,
  SUPPRESS = 260,
  PRINTING = 261,
  REPLACING = 262,
  OFF = 263,
  IN = 264,
  OF = 265,
  BY = 266,
  EQEQ = 267,
  TOKEN = 268
};
```

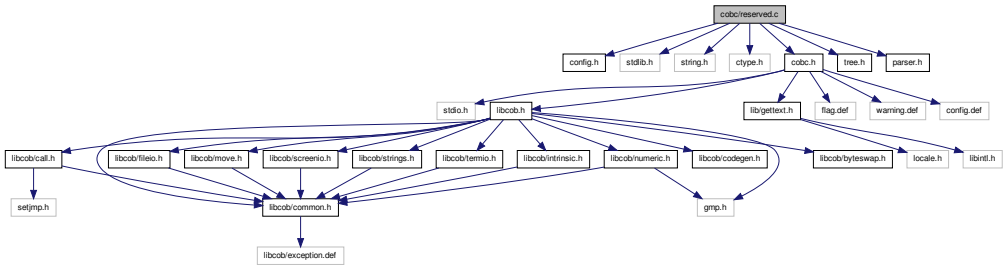
7.12.4 Variable Documentation

7.12.4.1 YYSTYPE pplval

7.13 cobc/reserved.c File Reference

```
#include "config.h"
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "cobc.h"
#include "tree.h"
#include "parser.h"
```

Include dependency graph for reserved.c:



Classes

- struct reserved

Defines

- #define NUM_RESERVED_WORDS sizeof (reserved_words) / sizeof (struct reserved)
- #define NUM_INTRINSICS sizeof(function_list) / sizeof(struct cb_intrinsic_table)

Functions

- cb_tree lookup_system_name (const char *name)
- int lookup_reserved_word (const char *name)
- struct cb_intrinsic_table * lookup_intrinsic (const char *name, const int checkres)

- void cb_list_reserved (void)
- void cb_list_intrinsics (void)
- void cb_list_mnemonics (void)
- void cb_init_reserved (void)

7.13.1 Define Documentation

7.13.1.1 #define NUM_INTRINSICS sizeof(function_list) / sizeof(struct cb_intrinsic_table)

Definition at line 878 of file reserved.c.

7.13.1.2 #define NUM_RESERVED_WORDS sizeof (reserved_words) / sizeof (struct reserved)

Definition at line 615 of file reserved.c.

7.13.2 Function Documentation

7.13.2.1 void cb_init_reserved (void)

Definition at line 1019 of file reserved.c.

```
{
        int     i;

        /* build system-name table */
        for (i = 0; system_table[i].name != NULL; ++i) {
                system_table[i].node =
                  cb_build_system_name (system_table[i].category, system_table[i]
      .token);
        }
}
```

7.13.2.2 void cb_list_intrinsics (void)

Definition at line 979 of file reserved.c.

```
{
        const char      *s;
        size_t          i;
        size_t          n;

        printf ("Intrinsic Function (Implemented Y/N)\n\n");
        for (i = 0; i < NUM_INTRINSICS; ++i) {
                n = strlen (function_list[i].name);
                switch (n / 8) {
                case 0:
                        s = "\t\t\t\t";
                        break;
                case 1:
```

```
                        s = "\t\t\t";
                        break;
                case 2:
                        s = "\t\t";
                        break;
                default:
                        s = "\t";
                        break;
                }
                printf ("%s%s(%s)\n", function_list[i].name, s,
                        function_list[i].implemented ? "Y" : "N");
        }
}
```

7.13.2.3 void cb_list_mnemonics (void)

Definition at line 1008 of file reserved.c.

```
{
        size_t          i;

        printf ("Mnemonic names\n\n");
        for (i = 0; system_table[i].name != NULL; ++i) {
                printf ("%s\n", system_table[i].name);
        }
}
```

7.13.2.4 void cb_list_reserved (void)

Definition at line 950 of file reserved.c.

```
{
        const char      *s;
        size_t  i;
        size_t  n;

        printf ("Reserved Words (Parsed Y/N)\n\n");
        for (i = 0; i < NUM_RESERVED_WORDS; ++i) {
                n = strlen (reserved_words[i].name);
                switch (n / 8) {
                case 0:
                        s = "\t\t\t\t";
                        break;
                case 1:
                        s = "\t\t\t";
                        break;
                case 2:
                        s = "\t\t";
                        break;
                default:
                        s = "\t";
                        break;
                }
                printf ("%s%s(%s)\n", reserved_words[i].name, s,
                        reserved_words[i].token != -1 ? "Y" : "N");
        }
}
```

7.13.2.5 struct cb_intrinsic_table* lookup_intrinsic (const char * *name*, const int *checkres*) [read]

Definition at line 929 of file reserved.c.

```
{
        struct cb_intrinsic_table       *cbp;
        struct noreserve                *noresptr;

        if (checkres) {
                for (noresptr = norestab; noresptr; noresptr = noresptr->next) {
                        if (strcasecmp (name, noresptr->noresword) == 0) {
                                return NULL;
                        }
                }
        }
        cbp = bsearch (name, function_list, NUM_INTRINSICS,
                        sizeof (struct cb_intrinsic_table), intrinsic_comp);
        if (cbp && cbp->implemented) {
                return cbp;
        }
        return NULL;
}
```

7.13.2.6 int lookup_reserved_word (const char * *name*)

Definition at line 906 of file reserved.c.

```
{
        struct reserved *p;
        struct noreserve        *noresptr;

        p = bsearch (name, reserved_words, NUM_RESERVED_WORDS,
                        sizeof (struct reserved), reserve_comp);
        if (!p) {
                return 0;
        }
        for (noresptr = norestab; noresptr; noresptr = noresptr->next) {
                if (strcasecmp (name, noresptr->noresword) == 0) {
                        return 0;
                }
        }
        if (p->token != -1) {
                return p->token;
        }
        cb_error (_("'%s' reserved word, but not supported yet"), name);
        return 0;
}
```

7.13.2.7 cb_tree lookup_system_name (const char * *name*)

Definition at line 893 of file reserved.c.

```
{
```

```
        int     i;

        for (i = 0; system_table[i].name != NULL; ++i) {
                if (strcasecmp (name, system_table[i].name) == 0) {
                        return system_table[i].node;
                }
        }
        return cb_error_node;
}
```

7.13.3 Variable Documentation

7.13.3.1 enum cb_system_name_category category

Definition at line 33 of file reserved.c.

7.13.3.2 const char* name

Definition at line 32 of file reserved.c.

7.13.3.3 cb_tree node

Definition at line 35 of file reserved.c.

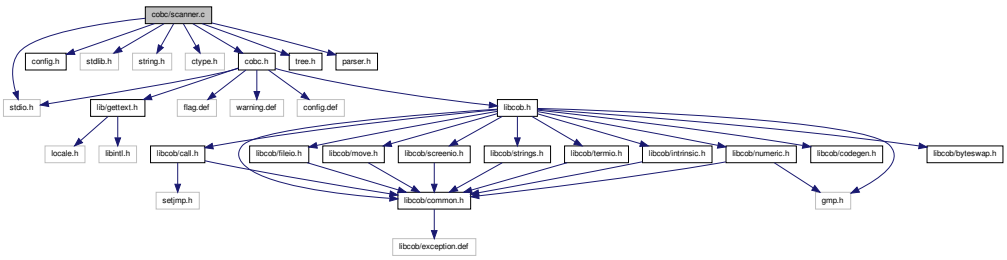
7.13.3.4 const int token

Definition at line 34 of file reserved.c.

7.14 cobc/scanner.c File Reference

```
#include <stdio.h>
#include "config.h"
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "cobc.h"
#include "tree.h"
#include "parser.h"
```

Include dependency graph for scanner.c:



Classes

- struct yy_buffer_state
- struct cb_level_78

Defines

- #define FLEX_SCANNER
- #define YY_FLEX_MAJOR_VERSION 2
- #define YY_FLEX_MINOR_VERSION 5
- #define yyconst
- #define YY_PROTO(proto) ()
- #define YY_NULL 0
- #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)
- #define BEGIN yy_start = 1 + 2 ∗
- #define YY_START ((yy_start - 1) / 2)
- #define YYSTATE YY_START
- #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
- #define YY_NEW_FILE yyrestart(yyin)
- #define YY_END_OF_BUFFER_CHAR 0
- #define YY_BUF_SIZE 16384
- #define EOB_ACT_CONTINUE_SCAN 0
- #define EOB_ACT_END_OF_FILE 1
- #define EOB_ACT_LAST_MATCH 2
- #define yyless(n)
- #define unput(c) yyunput(c, yytext_ptr)
- #define YY_BUFFER_NEW 0
- #define YY_BUFFER_NORMAL 1
- #define YY_BUFFER_EOF_PENDING 2
- #define YY_CURRENT_BUFFER yy_current_buffer
- #define YY_FLUSH_BUFFER yy_flush_buffer(yy_current_buffer)
- #define yy_new_buffer yy_create_buffer

- #define yy_set_interactive(is_interactive)
- #define yy_set_bol(at_bol)
- #define YY_AT_BOL() (yy_current_buffer->yy_at_bol)
- #define yywrap() 1
- #define YY_SKIP_YYWRAP
- #define yytext_ptr yytext
- #define YY_DO_BEFORE_ACTION
- #define YY_NUM_RULES 99
- #define YY_END_OF_BUFFER 100
- #define REJECT reject_used_but_not_detected
- #define yymore() yymore_used_but_not_detected
- #define YY_MORE_ADJ 0
- #define YY_RESTORE_YY_MORE_OFFSET
- #define INITIAL 0
- #define YY_NEVER_INTERACTIVE 1
- #define SET_LOCATION(x)
- #define DECIMAL_IS_PERIOD 1
- #define DECIMAL_IS_COMMA 2
- #define PICTURE_STATE 3
- #define FUNCTION_STATE 4
- #define YY_NO_PUSH_STATE 1
- #define YY_NO_POP_STATE 1
- #define YY_NO_TOP_STATE 1
- #define YY_READ_BUF_SIZE 8192
- #define ECHO (void) fwrite(yytext, yyleng, 1, yyout)
- #define YY_INPUT(buf, result, max_size)
- #define yyterminate() return YY_NULL
- #define YY_START_STACK_INCR 25
- #define YY_FATAL_ERROR(msg) yy_fatal_error(msg)
- #define YY_DECL int yylex YY_PROTO((void))
- #define YY_USER_ACTION
- #define YY_BREAK break;
- #define YY_RULE_SETUP
- #define YY_EXIT_FAILURE 2
- #define yyless(n)

Typedefs

- typedef struct yy_buffer_state * YY_BUFFER_STATE
- typedef unsigned int yy_size_t
- typedef unsigned char YY_CHAR
- typedef int yy_state_type

Functions

- void yyrestart YY_PROTO ((FILE *input_file))
- void yy_switch_to_buffer YY_PROTO ((YY_BUFFER_STATE new_buffer))
- void yy_load_buffer_state YY_PROTO ((void))
- YY_BUFFER_STATE yy_create_buffer YY_PROTO ((FILE *file, int size))
- void yy_delete_buffer YY_PROTO ((YY_BUFFER_STATE b))
- void yy_init_buffer YY_PROTO ((YY_BUFFER_STATE b, FILE *file))
- YY_BUFFER_STATE yy_scan_buffer YY_PROTO ((char *base, yy_size_t size))
- YY_BUFFER_STATE yy_scan_string YY_PROTO ((yyconst char *yy_str))
- YY_BUFFER_STATE yy_scan_bytes YY_PROTO ((yyconst char *bytes, int len))

Variables

- int yyleng
- FILE * yyin = (FILE *) 0
- FILE * yyout = (FILE *) 0
- char * yytext
- register char * yy_bp
- int size
- FILE * file
- int len

7.14.1 Define Documentation

7.14.1.1 #define BEGIN yy_start = 1 + 2 *

Definition at line 80 of file scanner.c.

7.14.1.2 #define DECIMAL_IS_COMMA 2

Definition at line 925 of file scanner.c.

7.14.1.3 #define DECIMAL_IS_PERIOD 1

Definition at line 924 of file scanner.c.

7.14.1.4 #define ECHO (void) fwrite(yytext, yyleng, 1, yyout)

Definition at line 1010 of file scanner.c.

7.14.1.5 #define EOB_ACT_CONTINUE_SCAN 0

Definition at line 105 of file scanner.c.

7.14.1.6 #define EOB_ACT_END_OF_FILE 1

Definition at line 106 of file scanner.c.

7.14.1.7 #define EOB_ACT_LAST_MATCH 2

Definition at line 107 of file scanner.c.

7.14.1.8 #define FLEX_SCANNER

Definition at line 8 of file scanner.c.

7.14.1.9 #define FUNCTION_STATE 4

Definition at line 928 of file scanner.c.

7.14.1.10 #define INITIAL 0

Definition at line 864 of file scanner.c.

7.14.1.11 #define PICTURE_STATE 3

Definition at line 927 of file scanner.c.

7.14.1.12 #define REJECT reject_used_but_not_detected

Definition at line 858 of file scanner.c.

7.14.1.13 #define SET_LOCATION(*x*)

Value:

```
(x)->source_file = (unsigned char *)cb_source_file;             \
  (x)->source_line = cb_source_line
```

Definition at line 897 of file scanner.c.

7.14.1.14 #define unput(*c*) yyunput(c, yytext_ptr)

Definition at line 136 of file scanner.c.

7.14.1.15 #define YY_AT_BOL( ) (yy_current_buffer->yy_at_bol)

Definition at line 263 of file scanner.c.

7.14.1.16 #define YY_BREAK break;

Definition at line 1069 of file scanner.c.

7.14.1.17 #define YY_BUF_SIZE 16384

Definition at line 98 of file scanner.c.

7.14.1.18 #define YY_BUFFER_EOF_PENDING 2

Definition at line 199 of file scanner.c.

7.14.1.19 #define YY_BUFFER_NEW 0

Definition at line 187 of file scanner.c.

7.14.1.20 #define YY_BUFFER_NORMAL 1

Definition at line 188 of file scanner.c.

7.14.1.21 #define YY_CURRENT_BUFFER yy_current_buffer

Definition at line 208 of file scanner.c.

7.14.1.22 #define YY_DECL int yylex YY_PROTO((void))

Definition at line 1057 of file scanner.c.

7.14.1.23 #define YY_DO_BEFORE_ACTION

Value:

```
yytext_ptr = yy_bp; \
        yyleng = (int) (yy_cp - yy_bp); \
        yy_hold_char = *yy_cp; \
        *yy_cp = '\0'; \
        yy_c_buf_p = yy_cp;
```

Definition at line 282 of file scanner.c.

7.14.1.24 #define YY_END_OF_BUFFER 100

Definition at line 290 of file scanner.c.

7.14.1.25 #define YY_END_OF_BUFFER_CHAR 0

Definition at line 95 of file scanner.c.

7.14.1.26 #define YY_EXIT_FAILURE 2

7.14.1.27 #define YY_FATAL_ERROR(*msg*) yy_fatal_error(msg)

Definition at line 1050 of file scanner.c.

7.14.1.28 #define YY_FLEX_MAJOR_VERSION 2

Definition at line 9 of file scanner.c.

7.14.1.29 #define YY_FLEX_MINOR_VERSION 5

Definition at line 10 of file scanner.c.

7.14.1.30 #define YY_FLUSH_BUFFER yy_flush_buffer(yy_current_buffer)

Definition at line 237 of file scanner.c.

7.14.1.31 #define YY_INPUT(*buf, result, max_size*)

Value:

```
if ( yy_current_buffer->yy_is_interactive ) \
                {\
                int c = '*', n; \
                for ( n = 0; n < max_size && \
                             (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
                        buf[n] = (char) c; \
                if ( c == '\n' ) \
                        buf[n++] = (char) c; \
                if ( c == EOF && ferror( yyin ) ) \
                        YY_FATAL_ERROR( "input in flex scanner failed" ); \
                result = n; \
                } \
        else if ( ((result = fread( buf, 1, max_size, yyin )) == 0) \
                  && ferror( yyin ) ) \
                YY_FATAL_ERROR( "input in flex scanner failed" );
```

Definition at line 1017 of file scanner.c.

7.14.1.32 #define YY_MORE_ADJ 0

Definition at line 860 of file scanner.c.

7.14.1.33 #define YY_NEVER_INTERACTIVE 1

Definition at line 884 of file scanner.c.

7.14.1.34 #define yy_new_buffer yy_create_buffer

Definition at line 247 of file scanner.c.

7.14.1.35 #define YY_NEW_FILE yyrestart(yyin)

Definition at line 93 of file scanner.c.

7.14.1.36 #define YY_NO_POP_STATE 1

Definition at line 980 of file scanner.c.

7.14.1.37 #define YY_NO_PUSH_STATE 1

Definition at line 979 of file scanner.c.

7.14.1.38 #define YY_NO_TOP_STATE 1

Definition at line 981 of file scanner.c.

7.14.1.39 #define YY_NULL 0

Definition at line 67 of file scanner.c.

7.14.1.40 #define YY_NUM_RULES 99

Definition at line 289 of file scanner.c.

7.14.1.41 #define YY_PROTO(*proto*) ()

Definition at line 63 of file scanner.c.

7.14.1.42 #define YY_READ_BUF_SIZE 8192

Definition at line 1001 of file scanner.c.

7.14.1.43 #define YY_RESTORE_YY_MORE_OFFSET

Definition at line 861 of file scanner.c.

7.14.1.44 #define YY_RULE_SETUP

Value:

```
if ( yyleng > 0 ) \
                yy_current_buffer->yy_at_bol = \
                                (yytext[yyleng - 1] == '\n'); \
        YY_USER_ACTION
```

Definition at line 1072 of file scanner.c.

7.14.1.45 #define YY_SC_TO_UI(*c*) ((unsigned int) (unsigned char) c)

Definition at line 74 of file scanner.c.

7.14.1.46 #define yy_set_bol(*at_bol*)

Value:

```
{ \
        if ( ! yy_current_buffer ) \
                yy_current_buffer = yy_create_buffer( yyin, YY_BUF_SIZE ); \
        yy_current_buffer->yy_at_bol = at_bol; \
        }
```

Definition at line 256 of file scanner.c.

7.14.1.47 #define yy_set_interactive(*is_interactive*)

Value:

```
{ \
        if ( ! yy_current_buffer ) \
                yy_current_buffer = yy_create_buffer( yyin, YY_BUF_SIZE ); \
        yy_current_buffer->yy_is_interactive = is_interactive; \
        }
```

Definition at line 249 of file scanner.c.

7.14.1.48 #define YY_SKIP_YYWRAP

Definition at line 267 of file scanner.c.

7.14.1.49 #define YY_START ((yy_start - 1) / 2)

Definition at line 86 of file scanner.c.

7.14.1.50 #define YY_START_STACK_INCR 25

Definition at line 1045 of file scanner.c.

7.14.1.51 #define YY_STATE_EOF(*state*) (YY_END_OF_BUFFER + state + 1)

Definition at line 90 of file scanner.c.

7.14.1.52 #define YY_USER_ACTION

Definition at line 1064 of file scanner.c.

7.14.1.53 #define yyconst

Definition at line 56 of file scanner.c.

7.14.1.54 #define yyless(*n*)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
                yytext[yyleng] = yy_hold_char; \
                yy_c_buf_p = yytext + n; \
                yy_hold_char = *yy_c_buf_p; \
                *yy_c_buf_p = '\0'; \
                yyleng = n; \
                } \
        while ( 0 )
```

Definition at line 125 of file scanner.c.

7.14.1.55 #define yyless(*n*)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
                *yy_cp = yy_hold_char; \
                YY_RESTORE_YY_MORE_OFFSET \
                yy_c_buf_p = yy_cp = yy_bp + n - YY_MORE_ADJ; \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
```

Definition at line 125 of file scanner.c.

7.14.1.56 #define yymore( ) yymore_used_but_not_detected

Definition at line 859 of file scanner.c.

7.14.1.57 #define YYSTATE YY_START

Definition at line 87 of file scanner.c.

7.14.1.58 #define yyterminate( ) return YY_NULL

Definition at line 1040 of file scanner.c.

7.14.1.59 #define yytext_ptr yytext

Definition at line 272 of file scanner.c.

7.14.1.60 #define yywrap( ) 1

Definition at line 266 of file scanner.c.

7.14.2 Typedef Documentation

7.14.2.1 typedef struct yy_buffer_state* YY_BUFFER_STATE

Definition at line 100 of file scanner.c.

7.14.2.2 typedef unsigned char YY_CHAR

Definition at line 268 of file scanner.c.

7.14.2.3 typedef unsigned int yy_size_t

Definition at line 142 of file scanner.c.

7.14.2.4 typedef int yy_state_type

Definition at line 270 of file scanner.c.

7.14.3 Function Documentation

7.14.3.1 YY_BUFFER_STATE yy_scan_string YY_PROTO ((yyconst char *yy_str))

7.14.3.2 YY_BUFFER_STATE yy_scan_buffer YY_PROTO ((char *base, yy_size_t size))

7.14.3.3 void yy_delete_buffer YY_PROTO ((YY_BUFFER_STATE b))

7.14.3.4 void yy_switch_to_buffer YY_PROTO ((YY_BUFFER_STATE new_buffer))

7.14.3.5 void yy_load_buffer_state YY_PROTO ((void))

7.14.3.6 void yy_init_buffer YY_PROTO ((YY_BUFFER_STATE b, FILE *file))

7.14.3.7 YY_BUFFER_STATE yy_scan_bytes YY_PROTO ((yyconst char *bytes, int len))

7.14.3.8 void yyrestart YY_PROTO ((FILE *input_file))

7.14.3.9 YY_BUFFER_STATE yy_create_buffer YY_PROTO ((FILE *file, int size))

7.14.4 Variable Documentation

7.14.4.1 FILE* file

Definition at line 2622 of file scanner.c.

7.14.4.2 int len

Definition at line 2744 of file scanner.c.

7.14.4.3 yy_size_t size

Definition at line 2533 of file scanner.c.

7.14.4.4 register char* yy_bp

Definition at line 2385 of file scanner.c.

7.14.4.5 FILE* yyin = (FILE *) 0

Definition at line 269 of file scanner.c.

7.14.4.6 int yyleng

Definition at line 217 of file scanner.c.

7.14.4.7 FILE * yyout = (FILE *) 0

Definition at line 269 of file scanner.c.

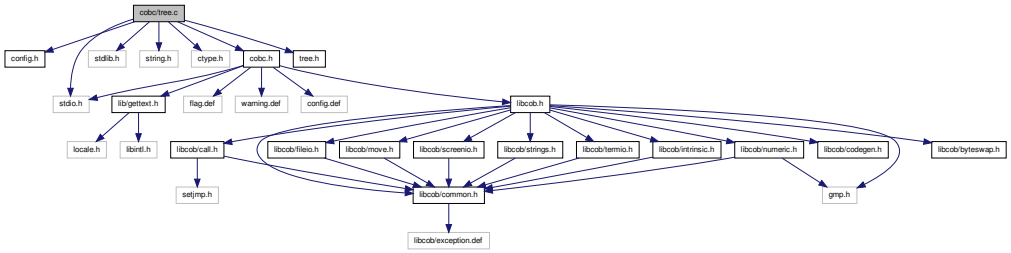
7.14.4.8 char* yytext

Definition at line 862 of file scanner.c.

7.15 cobc/tree.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "cobc.h"
#include "tree.h"
```

Include dependency graph for tree.c:



Classes

- struct **int_node**

Defines

- #define PIC_ALPHABETIC 0x01
- #define PIC_NUMERIC 0x02
- #define PIC_NATIONAL 0x04
- #define PIC_EDITED 0x08
- #define PIC_ALPHANUMERIC (PIC_ALPHABETIC | PIC_NUMERIC)

- #define PIC_ALPHABETIC_EDITED (PIC_ALPHABETIC | PIC_EDITED)
- #define PIC_ALPHANUMERIC_EDITED (PIC_ALPHANUMERIC | PIC_EDITED)
- #define PIC_NUMERIC_EDITED (PIC_NUMERIC | PIC_EDITED)
- #define PIC_NATIONAL_EDITED (PIC_NATIONAL | PIC_EDITED)

Functions

- struct cb_literal * build_literal (enum cb_category category, const unsigned char *data, size_t size)
- char * cb_name (cb_tree x)
- enum cb_class cb_tree_class (cb_tree x)
- enum cb_category cb_tree_category (cb_tree x)
- int cb_tree_type (cb_tree x)
- int cb_fits_int (cb_tree x)
- int cb_fits_long_long (cb_tree x)
- int cb_get_int (cb_tree x)
- long long cb_get_long_long (cb_tree x)
- void cb_init_constants (void)
- cb_tree cb_build_list (cb_tree purpose, cb_tree value, cb_tree rest)
- cb_tree cb_list_append (cb_tree l1, cb_tree l2)
- cb_tree cb_list_add (cb_tree l, cb_tree x)
- cb_tree cb_list_reverse (cb_tree l)
- int cb_list_length (cb_tree l)
- void cb_list_map (cb_tree(*func)(cb_tree x), cb_tree l)
- struct cb_program * cb_build_program (struct cb_program *last_program, int nest_level)
- cb_tree cb_int (int n)
- cb_tree cb_build_string (const unsigned char *data, size_t size)
- cb_tree cb_build_alphabet_name (cb_tree name, enum cb_alphabet_name_type type)
- cb_tree cb_build_class_name (cb_tree name, cb_tree list)
- cb_tree cb_build_locale_name (cb_tree name, cb_tree list)
- cb_tree cb_build_system_name (enum cb_system_name_category category, int token)
- cb_tree cb_build_numeric_literal (int sign, const unsigned char *data, int scale)
- cb_tree cb_build_alphanumeric_literal (const unsigned char *data, size_t size)
- cb_tree cb_concat_literals (cb_tree x1, cb_tree x2)
- cb_tree cb_build_decimal (int id)
- cb_tree cb_build_picture (const char *str)
- cb_tree cb_build_field (cb_tree name)
- cb_tree cb_build_implicit_field (cb_tree name, int len)
- cb_tree cb_build_constant (cb_tree name, cb_tree value)
- struct cb_field * cb_field (cb_tree x)
- struct cb_field * cb_field_add (struct cb_field *f, struct cb_field *p)
- int cb_field_size (cb_tree x)
- struct cb_field * cb_field_founder (struct cb_field *f)

- struct cb_field * cb_field_variable_size (struct cb_field *f)
- struct cb_field * cb_field_variable_address (struct cb_field *f)
- int cb_field_subordinate (struct cb_field *p, struct cb_field *f)
- struct cb_file * build_file (cb_tree name)
- void validate_file (struct cb_file *f, cb_tree name)
- void finalize_file (struct cb_file *f, struct cb_field *records)
- cb_tree cb_build_reference (const char *name)
- cb_tree cb_build_filler (void)
- cb_tree cb_build_field_reference (struct cb_field *f, cb_tree ref)
- const char * cb_define (cb_tree name, cb_tree val)
- void cb_define_system_name (const char *name)
- cb_tree cb_ref (cb_tree x)
- cb_tree cb_build_binary_op (cb_tree x, int op, cb_tree y)
- cb_tree cb_build_binary_list (cb_tree l, int op)
- cb_tree cb_build_funcall (const char *name, int argc, cb_tree a1, cb_tree a2, cb_tree a3, cb_tree a4, cb_tree a5, cb_tree a6, cb_tree a7)
- cb_tree cb_build_cast (enum cb_cast_type type, cb_tree val)
- cb_tree cb_build_label (cb_tree name, struct cb_label *section)
- cb_tree cb_build_assign (cb_tree var, cb_tree val)
- cb_tree cb_build_initialize (cb_tree var, cb_tree val, cb_tree rep, cb_tree def, int flag)
- cb_tree cb_build_search (int flag_all, cb_tree table, cb_tree var, cb_tree end_-stmt, cb_tree whens)
- cb_tree cb_build_call (cb_tree name, cb_tree args, cb_tree stmt1, cb_tree stmt2, cb_tree returning, int is_system_call)
- cb_tree cb_build_goto (cb_tree target, cb_tree depending)
- cb_tree cb_build_if (cb_tree test, cb_tree stmt1, cb_tree stmt2)
- cb_tree cb_build_perform (int type)
- cb_tree cb_build_perform_varying (cb_tree name, cb_tree from, cb_tree by, cb_-tree until)
- struct cb_statement * cb_build_statement (const char *name)
- cb_tree cb_build_continue (void)
- cb_tree cb_build_any_intrinsic (cb_tree args)
- cb_tree cb_build_intrinsic (cb_tree name, cb_tree args, cb_tree refmod)

Variables

- cb_tree cb_any
- cb_tree cb_true
- cb_tree cb_false
- cb_tree cb_null
- cb_tree cb_zero
- cb_tree cb_one
- cb_tree cb_space
- cb_tree cb_low
- cb_tree cb_high

- cb_tree cb_norm_low
- cb_tree cb_norm_high
- cb_tree cb_quote
- cb_tree cb_int0
- cb_tree cb_int1
- cb_tree cb_int2
- cb_tree cb_int3
- cb_tree cb_int4
- cb_tree cb_int5
- cb_tree cb_i [8]
- cb_tree cb_error_node
- cb_tree cb_intr_whencomp
- cb_tree cb_intr_pi
- cb_tree cb_intr_e
- cb_tree cb_standard_error_handler
- size_t gen_screen_ptr = 0

7.15.1 Define Documentation

7.15.1.1 #define PIC_ALPHABETIC 0x01

Definition at line 31 of file tree.c.

7.15.1.2 #define PIC_ALPHABETIC_EDITED (PIC_ALPHABETIC | PIC_EDITED)

Definition at line 36 of file tree.c.

7.15.1.3 #define PIC_ALPHANUMERIC (PIC_ALPHABETIC | PIC_NUMERIC)

Definition at line 35 of file tree.c.

7.15.1.4 #define PIC_ALPHANUMERIC_EDITED (PIC_ALPHANUMERIC | PIC_EDITED)

Definition at line 37 of file tree.c.

7.15.1.5 #define PIC_EDITED 0x08

Definition at line 34 of file tree.c.

7.15.1.6 #define PIC_NATIONAL 0x04

Definition at line 33 of file tree.c.

7.15.1.7 #define PIC_NATIONAL_EDITED (PIC_NATIONAL | PIC_EDITED)

Definition at line 39 of file tree.c.

7.15.1.8 #define PIC_NUMERIC 0x02

Definition at line 32 of file tree.c.

7.15.1.9 #define PIC_NUMERIC_EDITED (PIC_NUMERIC | PIC_EDITED)

Definition at line 38 of file tree.c.

7.15.2 Function Documentation

7.15.2.1 struct cb_file* build_file (cb_tree *name*) [read]

Definition at line 1577 of file tree.c.

```
{
        struct cb_file *p;

        p = make_tree (CB_TAG_FILE, CB_CATEGORY_UNKNOWN, sizeof (struct cb_file))
      ;
        p->name = cb_define (name, CB_TREE (p));
        p->cname = to_cname (p->name);

        p->organization = COB_ORG_SEQUENTIAL;
        p->access_mode = COB_ACCESS_SEQUENTIAL;
        p->handler = CB_LABEL (cb_standard_error_handler);
        p->handler_prog = current_program;
        return p;
}
```

7.15.2.2 struct cb_literal* build_literal (enum cb_category *category*, const unsigned char * *data*, size_t *size*) [read]

Definition at line 426 of file tree.c.

```
{
        struct cb_literal *p;

        p = make_tree (CB_TAG_LITERAL, category, sizeof (struct cb_literal));
        p->data = cobc_malloc ((size_t) (size + 1));
        p->size = size;
        memcpy (p->data, data, (size_t) size);
        /* RXW - malloc zeroes
        p->data[size] = 0;
        */
        return p;
}
```

7.15.2.3 cb_tree cb_build_alphabet_name (cb_tree *name*, enum cb_alphabet_name_type *type*)

Definition at line 923 of file tree.c.

```
{
        struct cb_alphabet_name *p;

        p = make_tree (CB_TAG_ALPHABET_NAME, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_alphabet_name));
        p->name = cb_define (name, CB_TREE (p));
        p->cname = to_cname (p->name);
        p->type = type;
        return CB_TREE (p);
}
```

7.15.2.4 cb_tree cb_build_alphanumeric_literal (const unsigned char * *data*, size_t *size*)

Definition at line 999 of file tree.c.

```
{
        return CB_TREE (build_literal (CB_CATEGORY_ALPHANUMERIC, data, size));
}
```

7.15.2.5 cb_tree cb_build_any_intrinsic (cb_tree *args*)

Definition at line 2253 of file tree.c.

```
{
        struct cb_intrinsic_table       *cbp;

        cbp = lookup_intrinsic ("LENGTH", 0);
        return make_intrinsic (NULL, cbp, args, NULL, NULL);
}
```

7.15.2.6 cb_tree cb_build_assign (cb_tree *var*, cb_tree *val*)

Definition at line 2098 of file tree.c.

```
{
        struct cb_assign *p;

        p = make_tree (CB_TAG_ASSIGN, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_assign));
        p->var = var;
        p->val = val;
        return CB_TREE (p);
}
```

7.15.2.7 cb_tree cb_build_binary_list (cb_tree *l*, int *op*)

Definition at line 2019 of file tree.c.

```
{
        cb_tree e;

        e = CB_VALUE (l);
        for (l = CB_CHAIN (l); l; l = CB_CHAIN (l)) {
                e = cb_build_binary_op (e, op, CB_VALUE (l));
        }
        return e;
}
```

7.15.2.8 cb_tree cb_build_binary_op (cb_tree *x*, int *op*, cb_tree *y*)

Definition at line 1954 of file tree.c.

```
{
        struct cb_binary_op     *p;
        enum cb_category        category = CB_CATEGORY_UNKNOWN;

        switch (op) {
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
                /* arithmetic operators */
                if (CB_TREE_CLASS (x) == CB_CLASS_POINTER ||
                    CB_TREE_CLASS (y) == CB_CLASS_POINTER) {
                        category = CB_CATEGORY_DATA_POINTER;
                        break;
                }
                x = cb_check_numeric_value (x);
                y = cb_check_numeric_value (y);
                if (x == cb_error_node || y == cb_error_node) {
                        return cb_error_node;
                }
                category = CB_CATEGORY_NUMERIC;
                break;

        case '=':
        case '~':
        case '<':
        case '>':
        case '[':
        case ']':
                /* relational operators */
                category = CB_CATEGORY_BOOLEAN;
                break;

        case '!':
        case '&':
        case '|':
                /* logical operators */
                if (CB_TREE_CLASS (x) != CB_CLASS_BOOLEAN ||
                    (y && CB_TREE_CLASS (y) != CB_CLASS_BOOLEAN)) {
```

```
                        cb_error (_("Invalid expression"));
                        return cb_error_node;
                }
                category = CB_CATEGORY_BOOLEAN;
                break;

        case '@':
                /* parentheses */
                category = CB_TREE_CATEGORY (x);
                break;

        default:
                fprintf (stderr, "Unexpected operator -> %d\n", op);
                ABORT ();
        }

        p = make_tree (CB_TAG_BINARY_OP, category, sizeof (struct cb_binary_op));

        p->op = op;
        p->x = x;
        p->y = y;
        return CB_TREE (p);
}
```

7.15.2.9 cb_tree cb_build_call (cb_tree *name*, cb_tree *args*, cb_tree *stmt1*, cb_tree *stmt2*, cb_tree *returning*, int *is_system_call*)

Definition at line 2149 of file tree.c.

```
{
        struct cb_call *p;

        p = make_tree (CB_TAG_CALL, CB_CATEGORY_UNKNOWN, sizeof (struct cb_call))
      ;
        p->name = name;
        p->args = args;
        p->stmt1 = stmt1;
        p->stmt2 = stmt2;
        p->returning = returning;
        p->is_system = is_system_call;
        return CB_TREE (p);
}
```

7.15.2.10 cb_tree cb_build_cast (enum cb_cast_type *type*, cb_tree *val*)

Definition at line 2060 of file tree.c.

```
{
        struct cb_cast          *p;
        enum cb_category        category;

        if (type == CB_CAST_INTEGER) {
                category = CB_CATEGORY_NUMERIC;
        } else {
                category = CB_CATEGORY_UNKNOWN;
```

```
        }
        p = make_tree (CB_TAG_CAST, category, sizeof (struct cb_cast));
        p->type = type;
        p->val = val;
        return CB_TREE (p);
}
```

7.15.2.11 cb_tree cb_build_class_name (cb_tree *name,* cb_tree *list*)

Definition at line 939 of file tree.c.

```
{
        struct cb_class_name    *p;
        char                    buff[COB_MINI_BUFF];

        p = make_tree (CB_TAG_CLASS_NAME, CB_CATEGORY_BOOLEAN, sizeof (struct
      cb_class_name));
        p->name = cb_define (name, CB_TREE (p));
        snprintf (buff, COB_MINI_MAX, "is_%s", to_cname (p->name));
        p->cname = strdup (buff);
        p->list = list;
        return CB_TREE (p);
}
```

7.15.2.12 cb_tree cb_build_constant (cb_tree *name,* cb_tree *value*)

Definition at line 1446 of file tree.c.

```
{
        cb_tree x;

        x = cb_build_field (name);
        x->category = cb_tree_category (value);
        CB_FIELD (x)->storage = CB_STORAGE_CONSTANT;
        CB_FIELD (x)->values = cb_list_init (value);
        return x;
}
```

7.15.2.13 cb_tree cb_build_continue (void)

Definition at line 2240 of file tree.c.

```
{
        struct cb_continue *p;

        p = make_tree (CB_TAG_CONTINUE, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_continue));
        return CB_TREE (p);
}
```

7.15.2.14 cb_tree cb_build_decimal (int *id*)

Definition at line 1076 of file tree.c.

```
{
        struct cb_decimal *p;

        p = make_tree (CB_TAG_DECIMAL, CB_CATEGORY_NUMERIC, sizeof (struct
      cb_decimal));
        p->id = id;
        return CB_TREE (p);
}
```

7.15.2.15 cb_tree cb_build_field (cb_tree *name*)

Definition at line 1417 of file tree.c.

```
{
        struct cb_field *p;

        p = make_tree (CB_TAG_FIELD, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_field));
        p->id = cb_field_id++;
        p->name = cb_define (name, CB_TREE (p));
        p->ename = NULL;
        p->usage = CB_USAGE_DISPLAY;
        p->storage = CB_STORAGE_WORKING;
        p->occurs_max = 1;
        return CB_TREE (p);
}
```

7.15.2.16 cb_tree cb_build_field_reference (struct cb_field * *f,* cb_tree *ref*)

Definition at line 1752 of file tree.c.

```
{
        cb_tree         x;
        struct cb_word  *word;

        x = cb_build_reference (f->name);
        word = CB_REFERENCE (x)->word;
        if (ref) {
                memcpy (x, ref, sizeof (struct cb_reference));
        }
        x->category = CB_CATEGORY_UNKNOWN;
        CB_REFERENCE (x)->word = word;
        CB_REFERENCE (x)->value = CB_TREE (f);
        return x;
}
```

7.15.2.17 cb_tree cb_build_filler (void)

Definition at line 1740 of file tree.c.

```
{
        cb_tree         x;
        char            name[16];

        sprintf (name, "WORK$%d", filler_id++);
        x = cb_build_reference (name);
        x->source_line = cb_source_line;
        return x;
}
```

7.15.2.18 cb_tree cb_build_funcall (const char * *name*, int *argc*, cb_tree *a1*, cb_tree *a2*, cb_tree *a3*, cb_tree *a4*, cb_tree *a5*, cb_tree *a6*, cb_tree *a7*)

Definition at line 2035 of file tree.c.

```
{
        struct cb_funcall *p;

        p = make_tree (CB_TAG_FUNCALL, CB_CATEGORY_BOOLEAN, sizeof (struct
      cb_funcall));
        p->name = name;
        p->argc = argc;
        p->varcnt = 0;
        p->screenptr = gen_screen_ptr;
        p->argv[0] = a1;
        p->argv[1] = a2;
        p->argv[2] = a3;
        p->argv[3] = a4;
        p->argv[4] = a5;
        p->argv[5] = a6;
        p->argv[6] = a7;
        return CB_TREE (p);
}
```

7.15.2.19 cb_tree cb_build_goto (cb_tree *target*, cb_tree *depending*)

Definition at line 2168 of file tree.c.

```
{
        struct cb_goto *p;

        p = make_tree (CB_TAG_GOTO, CB_CATEGORY_UNKNOWN, sizeof (struct cb_goto))
      ;
        p->target = target;
        p->depending = depending;
        return CB_TREE (p);
}
```

7.15.2.20 cb_tree cb_build_if (cb_tree *test*, cb_tree *stmt1*, cb_tree *stmt2*)

Definition at line 2183 of file tree.c.

```
{
        struct cb_if *p;

        p = make_tree (CB_TAG_IF, CB_CATEGORY_UNKNOWN, sizeof (struct cb_if));
        p->test = test;
        p->stmt1 = stmt1;
        p->stmt2 = stmt2;
        return CB_TREE (p);
}
```

7.15.2.21 cb_tree cb_build_implicit_field (cb_tree *name*, int *len*)

Definition at line 1432 of file tree.c.

```
{
        cb_tree x;
        char    pic[32];

        x = cb_build_field (name);
        memset (pic, 0, sizeof(pic));
        sprintf (pic, "X(%d)", len);
        CB_FIELD (x)->pic = CB_PICTURE (cb_build_picture (pic));
        cb_validate_field (CB_FIELD (x));
        return x;
}
```

7.15.2.22 cb_tree cb_build_initialize (cb_tree *var*, cb_tree *val*, cb_tree *rep*, cb_tree *def*, int *flag*)

Definition at line 2113 of file tree.c.

```
{
        struct cb_initialize *p;

        p = make_tree (CB_TAG_INITIALIZE, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_initialize));
        p->var = var;
        p->val = val;
        p->rep = rep;
        p->def = def;
        p->flag_statement = flag;
        return CB_TREE (p);
}
```

7.15.2.23 cb_tree cb_build_intrinsic (cb_tree *name*, cb_tree *args*, cb_tree *refmod*)

Definition at line 2262 of file tree.c.

```
{
        struct cb_intrinsic_table       *cbp;
        cb_tree                         x;
        int                             numargs;

        numargs = cb_list_length (args);

        cbp = lookup_intrinsic (CB_NAME (name), 0);
        if (cbp) {
                if ((cbp->args != -1 && numargs != cbp->args) ||
                    (cbp->args == -1 && cbp->intr_enum != CB_INTR_RANDOM && numar
      gs < 1)) {
                        cb_error_x (name, _("FUNCTION %s has wrong number of argu
      ments"), cbp->name);
                        return cb_error_node;
                }
                if (refmod) {
                        if (!cbp->refmod) {
                                cb_error_x (name, _("FUNCTION %s can not have ref
      erence modification"), cbp->name);
                                return cb_error_node;
                        }
                        if (CB_LITERAL_P(CB_PAIR_X(refmod)) &&
                            cb_get_int (CB_PAIR_X(refmod))< 1) {
                                cb_error_x (name, _("FUNCTION %s has invalid refe
      rence modification"), cbp->name);
                                return cb_error_node;
                        }
                        if (CB_PAIR_Y(refmod) && CB_LITERAL_P(CB_PAIR_Y(refmod))
      &&
                            cb_get_int (CB_PAIR_Y(refmod))< 1) {
                                cb_error_x (name, _("FUNCTION %s has invalid refe
      rence modification"), cbp->name);
                                return cb_error_node;
                        }
                }
                /* cb_tree      x; */
                switch (cbp->intr_enum) {
                case CB_INTR_LENGTH:
                case CB_INTR_BYTE_LENGTH:
                        x = CB_VALUE (args);
                        if (CB_INTRINSIC_P (x)) {
                                return make_intrinsic (name, cbp, args, NULL,
      NULL);
                        } else if ((CB_FIELD_P (x) || CB_REFERENCE_P (x)) &&
                                    cb_field(x)->flag_any_length) {
                                return make_intrinsic (name, cbp, args, NULL,
      NULL);
                        } else {
                                return cb_build_length (CB_VALUE (args));
                        }

                case CB_INTR_WHEN_COMPILED:
                        if (refmod) {
                                return make_intrinsic (name, cbp,
                                        cb_list_init (cb_intr_whencomp), NULL, re
      fmod);
                        } else {
                                return cb_intr_whencomp;
                        }
                case CB_INTR_PI:
                        return cb_intr_pi;
```

```
                case CB_INTR_E:
                        return cb_intr_e;

                case CB_INTR_LOWER_CASE:
                case CB_INTR_UPPER_CASE:
                case CB_INTR_REVERSE:
/* RXW Why did I do this ? - still do not know
                        if (CB_INTRINSIC_P (CB_VALUE (args))) {
                                return make_intrinsic (name, cbp, args, cb_int0);

                        } else {
                                return make_intrinsic (name, cbp, args,
                                                       cb_build_length (CB_VALUE
      (args)));
                        }
RXW */

                case CB_INTR_ABS:
                case CB_INTR_ACOS:
                case CB_INTR_ANNUITY:
                case CB_INTR_ASIN:
                case CB_INTR_ATAN:
                case CB_INTR_CHAR:
                case CB_INTR_COMBINED_DATETIME:
                case CB_INTR_COS:
                case CB_INTR_CURRENT_DATE:
                case CB_INTR_DATE_OF_INTEGER:
                case CB_INTR_DAY_OF_INTEGER:
                case CB_INTR_EXCEPTION_FILE:
                case CB_INTR_EXCEPTION_LOCATION:
                case CB_INTR_EXCEPTION_STATUS:
                case CB_INTR_EXCEPTION_STATEMENT:
                case CB_INTR_EXP:
                case CB_INTR_EXP10:
                case CB_INTR_FACTORIAL:
                case CB_INTR_FRACTION_PART:
                case CB_INTR_INTEGER:
                case CB_INTR_INTEGER_OF_DATE:
                case CB_INTR_INTEGER_OF_DAY:
                case CB_INTR_INTEGER_PART:
                case CB_INTR_LOCALE_DATE:
                case CB_INTR_LOCALE_TIME:
                case CB_INTR_LOCALE_TIME_FROM_SECS:
                case CB_INTR_LOG:
                case CB_INTR_LOG10:
                case CB_INTR_MOD:
                case CB_INTR_NUMVAL:
                case CB_INTR_NUMVAL_C:
                case CB_INTR_ORD:
                case CB_INTR_REM:
                case CB_INTR_SECONDS_FROM_FORMATTED_TIME:
                case CB_INTR_SECONDS_PAST_MIDNIGHT:
                case CB_INTR_SIGN:
                case CB_INTR_SIN:
                case CB_INTR_SQRT:
                case CB_INTR_STORED_CHAR_LENGTH:
                case CB_INTR_TAN:
                case CB_INTR_TEST_DATE_YYYYMMDD:
                case CB_INTR_TEST_DAY_YYYYDDD:
                case CB_INTR_TRIM:
                        return make_intrinsic (name, cbp, args, NULL, refmod);
```

```
                case CB_INTR_CONCATENATE:
                        return make_intrinsic (name, cbp, args, cb_int1, refmod);

                case CB_INTR_DATE_TO_YYYYMMDD:
                case CB_INTR_DAY_TO_YYYYDDD:
                case CB_INTR_MAX:
                case CB_INTR_MEAN:
                case CB_INTR_MEDIAN:
                case CB_INTR_MIDRANGE:
                case CB_INTR_MIN:
                case CB_INTR_ORD_MAX:
                case CB_INTR_ORD_MIN:
                case CB_INTR_PRESENT_VALUE:
                case CB_INTR_RANDOM:
                case CB_INTR_RANGE:
                case CB_INTR_STANDARD_DEVIATION:
                case CB_INTR_SUM:
                case CB_INTR_VARIANCE:
                case CB_INTR_YEAR_TO_YYYY:
                        return make_intrinsic (name, cbp, args, cb_int1, NULL);
                case CB_INTR_SUBSTITUTE:
                case CB_INTR_SUBSTITUTE_CASE:
                        if (numargs < 3 || (numargs % 2) == 0) {
                                cb_error_x (name, _("FUNCTION %s has wrong number
       of arguments"), cbp->name);
                                return cb_error_node;
                        }
                        return make_intrinsic (name, cbp, args, cb_int1, refmod);


                default:
                        break;
                }
        }
        cb_error_x (name, _("FUNCTION %s not implemented"), CB_NAME (name));
        return cb_error_node;
}
```

7.15.2.24 cb_tree cb_build_label (cb_tree *name*, struct cb_label * *section*)

Definition at line 2081 of file tree.c.

```
{
        struct cb_label *p;

        p = make_tree (CB_TAG_LABEL, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_label));
        p->id = cb_id++;
        p->name = (const unsigned char *)cb_define (name, CB_TREE (p));
        p->orig_name = p->name;
        p->section = section;
        return CB_TREE (p);
}
```

7.15.2.25 cb_tree cb_build_list (cb_tree *purpose*, cb_tree *value*, cb_tree *rest*)

Definition at line 769 of file tree.c.

```
{
        struct cb_list *p;

        p = make_tree (CB_TAG_LIST, CB_CATEGORY_UNKNOWN, sizeof (struct cb_list))
      ;
        p->purpose = purpose;
        p->value = value;
        p->chain = rest;
        return CB_TREE (p);
}
```

7.15.2.26 cb_tree cb_build_locale_name (cb_tree *name*, cb_tree *list*)

Definition at line 957 of file tree.c.

```
{
        struct cb_class_name    *p;

        p = make_tree (CB_TAG_LOCALE_NAME, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_locale_name));
        p->name = cb_define (name, CB_TREE (p));
        p->cname = to_cname (p->name);
        p->list = list;
        return CB_TREE (p);
}
```

7.15.2.27 cb_tree cb_build_numeric_literal (int *sign*, const unsigned char * *data*, int *scale*)

Definition at line 988 of file tree.c.

```
{
        struct cb_literal *p;

        p = build_literal (CB_CATEGORY_NUMERIC, data, strlen ((char *)data));
        p->sign = (char)sign;
        p->scale = (char)scale;
        return CB_TREE (p);
}
```

7.15.2.28 cb_tree cb_build_perform (int *type*)

Definition at line 2199 of file tree.c.

```
{
        struct cb_perform *p;

        p = make_tree (CB_TAG_PERFORM, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_perform));
        p->type = type;
        return CB_TREE (p);
}
```

7.15.2.29 cb_tree cb_build_perform_varying (cb_tree *name*, cb_tree *from*, cb_tree *by*, cb_tree *until*)

Definition at line 2209 of file tree.c.

```
{
        struct cb_perform_varying *p;

        p = make_tree (CB_TAG_PERFORM_VARYING, CB_CATEGORY_UNKNOWN, sizeof (struc
      t cb_perform_varying));
        p->name = name;
        p->from = from;
        p->step = name ? cb_build_add (name, by, cb_high) : NULL;
        p->until = until;
        return CB_TREE (p);
}
```

7.15.2.30 cb_tree cb_build_picture (const char * *str*)

Definition at line 1090 of file tree.c.

```
{
        struct cb_picture       *pic;
        const char              *p;
        size_t                  idx = 0;
        size_t                  buffcnt = 0;
        size_t                  at_beginning;
        size_t                  at_end;
        size_t                  p_char_seen;
        size_t                  s_char_seen;
        int                     category = 0;
        int                     size = 0;
        int                     allocated = 0;
        int                     digits = 0;
        int                     scale = 0;
        int                     s_count = 0;
        int                     v_count = 0;
        int                     i;
        int                     n;
        unsigned char           c;
        unsigned char           lastonechar = 0;
        unsigned char           lasttwochar = 0;
        unsigned char           buff[COB_SMALL_BUFF];

        pic = make_tree (CB_TAG_PICTURE, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_picture));
        if (strlen (str) > 50) {
                goto error;
        }
        memset (buff, 0, sizeof (buff));
        p_char_seen = 0;
        s_char_seen = 0;
        for (p = str; *p; p++) {
                n = 1;
                c = *p;
repeat:
                /* count the number of repeated chars */
                while (p[1] == c) {
```

```
        p++, n++;
}

/* add parenthesized numbers */
if (p[1] == '(') {
        i = 0;
        p += 2;
        for (; *p == '0'; p++) {
                ;
        }
        for (; *p != ')'; p++) {
                if (!isdigit (*p)) {
                        goto error;
                } else {
                        allocated++;
                        if (allocated > 9) {
                                goto error;
                        }
                        i = i * 10 + (*p - '0');
                }
        }
        if (i == 0) {
                goto error;
        }
        n += i - 1;
        goto repeat;
}

/* check grammar and category */
/* FIXME: need more error check */
switch (c) {
case 'A':
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        category |= PIC_ALPHABETIC;
        break;

case 'X':
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        category |= PIC_ALPHANUMERIC;
        break;

case '9':
        category |= PIC_NUMERIC;
        digits += n;
        if (v_count) {
                scale += n;
        }
        break;

case 'N':
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        category |= PIC_NATIONAL;
        break;

case 'S':
        category |= PIC_NUMERIC;
```

```
                if (category & PIC_ALPHABETIC) {
                        goto error;
                }
                s_count += n;
                if (s_count > 1 || idx != 0) {
                        goto error;
                }
                s_char_seen = 1;
                continue;

        case ',':
        case '.':
                category |= PIC_NUMERIC_EDITED;
                if (s_char_seen || p_char_seen) {
                        goto error;
                }
                if (c != current_program->decimal_point) {
                        break;
                }
                /* fall through */
        case 'V':
                category |= PIC_NUMERIC;
                if (category & PIC_ALPHABETIC) {
                        goto error;
                }
                v_count += n;
                if (v_count > 1) {
                        goto error;
                }
                break;

        case 'P':
                category |= PIC_NUMERIC;
                if (category & PIC_ALPHABETIC) {
                        goto error;
                }
                if (p_char_seen) {
                        goto error;
                }
                at_beginning = 0;
                at_end = 0;
                switch (buffcnt) {
                case 0:
                        /* P..... */
                        at_beginning = 1;
                        break;
                case 1:
                        /* VP.... */
                        /* SP.... */
                        if (lastonechar == 'V' || lastonechar == 'S') {
                                at_beginning = 1;
                        }
                        break;
                case 2:
                        /* SVP... */
                        if (lasttwochar == 'S' && lastonechar == 'V') {
                                at_beginning = 1;
                        }
                        break;
                }
                if (p[1] == 0 || (p[1] == 'V' && p[2] == 0)) {
                        /* .....P */
```

```
                /* ....PV */
                at_end = 1;
        }
        if (!at_beginning && !at_end) {
                goto error;
        }
        p_char_seen = 1;
        if (at_beginning) {
                v_count++;      /* implicit V */
        }
        digits += n;
        if (v_count) {
                scale += n;
        } else {
                scale -= n;
        }
        break;

case '0':
case 'B':
case '/':
        category |= PIC_EDITED;
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        break;

case '*':
case 'Z':
        category |= PIC_NUMERIC_EDITED;
        if (category & PIC_ALPHABETIC) {
                goto error;
        }
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        digits += n;
        if (v_count) {
                scale += n;
        }
        break;

case '+':
case '-':
        category |= PIC_NUMERIC_EDITED;
        if (category & PIC_ALPHABETIC) {
                goto error;
        }
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        digits += n - 1;
        s_count++;
        /* FIXME: need more check */
        break;

case 'C':
        category |= PIC_NUMERIC_EDITED;
        if (!(p[1] == 'R' && p[2] == 0)) {
                goto error;
        }
        if (s_char_seen || p_char_seen) {
```

```
                        goto error;
                }
                p++;
                s_count++;
                break;

        case 'D':
                category |= PIC_NUMERIC_EDITED;
                if (!(p[1] == 'B' && p[2] == 0)) {
                        goto error;
                }
                if (s_char_seen || p_char_seen) {
                        goto error;
                }
                p++;
                s_count++;
                break;

        default:
                if (c == current_program->currency_symbol) {
                        category |= PIC_NUMERIC_EDITED;
                        digits += n - 1;
                        /* FIXME: need more check */
                        break;
                }

                goto error;
        }

        /* calculate size */
        if (c != 'V' && c != 'P') {
                size += n;
        }
        if (c == 'C' || c == 'D' || c == 'N') {
                size += n;
        }

        /* store in the buffer */
        buff[idx++] = c;
        lasttwochar = lastonechar;
        lastonechar = c;
        memcpy (&buff[idx], (unsigned char *)&n, sizeof(int));
        idx += sizeof(int);
        ++buffcnt;
}
buff[idx] = 0;

if (size == 0 && v_count) {
        goto error;
}
/* set picture */
pic->orig = strdup (str);
pic->size = size;
pic->digits = (unsigned char)digits;
pic->scale = (signed char)scale;
pic->have_sign = (unsigned char)s_count;

/* set picture category */
switch (category) {
case PIC_ALPHABETIC:
        pic->category = CB_CATEGORY_ALPHABETIC;
        break;
```

```
        case PIC_NUMERIC:
                pic->category = CB_CATEGORY_NUMERIC;
                if (digits > 36) {
                        cb_error (_("Numeric field cannot be larger than 36 digit
      s"));
                }
                break;
        case PIC_ALPHANUMERIC:
        case PIC_NATIONAL:
                pic->category = CB_CATEGORY_ALPHANUMERIC;
                break;
        case PIC_NUMERIC_EDITED:
                pic->str = cobc_malloc (idx + 1);
                memcpy (pic->str, buff, idx);
                pic->category = CB_CATEGORY_NUMERIC_EDITED;
                pic->lenstr = idx;
                break;
        case PIC_EDITED:
        case PIC_ALPHABETIC_EDITED:
        case PIC_ALPHANUMERIC_EDITED:
        case PIC_NATIONAL_EDITED:
                pic->str = cobc_malloc (idx + 1);
                memcpy (pic->str, buff, idx);
                pic->category = CB_CATEGORY_ALPHANUMERIC_EDITED;
                pic->lenstr = idx;
                break;
        default:
                goto error;
        }
        goto end;

error:
        cb_error (_("Invalid picture string - '%s'"), str);

end:
        return CB_TREE (pic);
}
```

7.15.2.31 struct cb_program* cb_build_program (struct cb_program * *last_program*, int *nest_level*) [read]

Definition at line 841 of file tree.c.

```
{
        struct cb_program *p;

        cb_reset_78 ();
        cb_reset_in_procedure ();
        cb_clear_real_field ();
        p = cobc_malloc (sizeof (struct cb_program));
        p->next_program = last_program;
        p->nested_level = nest_level;
        p->decimal_point = '.';
        p->currency_symbol = '$';
        p->numeric_separator = ',';
        if (nest_level) {
                p->global_file_list = last_program->global_file_list;
                p->collating_sequence = last_program->collating_sequence;
                p->function_spec_list = last_program->function_spec_list;
```

```
                p->class_spec_list = last_program->class_spec_list;
                p->interface_spec_list = last_program->interface_spec_list;
                p->program_spec_list = last_program->program_spec_list;
                p->property_spec_list = last_program->property_spec_list;
                p->alphabet_name_list = last_program->alphabet_name_list;
                p->class_name_list = last_program->class_name_list;
                p->locale_list = last_program->locale_list;
                p->symbolic_list = last_program->symbolic_list;
                p->decimal_point = last_program->decimal_point;
                p->numeric_separator = last_program->numeric_separator;
                p->currency_symbol = last_program->currency_symbol;
                p->cb_return_code = last_program->cb_return_code;
        } else {
                functions_are_all = cb_flag_functions_all;
        }
        return p;
}
```

7.15.2.32 cb_tree cb_build_reference (const char * *name*)

Definition at line 1730 of file tree.c.

```
{
        struct cb_reference *p;

        p = make_tree (CB_TAG_REFERENCE, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_reference));
        p->word = lookup_word (name);
        return CB_TREE (p);
}
```

7.15.2.33 cb_tree cb_build_search (int *flag_all*, cb_tree *table*, cb_tree *var*, cb_tree *end_stmt*, cb_tree *whens*)

Definition at line 2131 of file tree.c.

```
{
        struct cb_search *p;

        p = make_tree (CB_TAG_SEARCH, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_search));
        p->flag_all = flag_all;
        p->table = table;
        p->var = var;
        p->end_stmt = end_stmt;
        p->whens = whens;
        return CB_TREE (p);
}
```

7.15.2.34 struct cb_statement* cb_build_statement (const char * *name*) [read]

Definition at line 2226 of file tree.c.

```
{
        struct cb_statement *p;

        p = make_tree (CB_TAG_STATEMENT, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_statement));
        p->name = name;
        return p;
}
```

7.15.2.35 cb_tree cb_build_string (const unsigned char * *data*, size_t *size*)

Definition at line 908 of file tree.c.

```
{
        struct cb_string *p;

        p = make_tree (CB_TAG_STRING, CB_CATEGORY_ALPHANUMERIC, sizeof (struct
      cb_string));
        p->size = size;
        p->data = data;
        return CB_TREE (p);
}
```

7.15.2.36 cb_tree cb_build_system_name (enum cb_system_name_category *category*, int *token*)

Definition at line 973 of file tree.c.

```
{
        struct cb_system_name *p;

        p = make_tree (CB_TAG_SYSTEM_NAME, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_system_name));
        p->category = category;
        p->token = token;
        return CB_TREE (p);
}
```

7.15.2.37 cb_tree cb_concat_literals (cb_tree *x1*, cb_tree *x2*)

Definition at line 1005 of file tree.c.

```
{
        unsigned char           *buff;
        cb_tree                 x;
        unsigned char           *data1;
        unsigned char           *data2;
        size_t                  size1;
        size_t                  size2;

        if (x1 == cb_error_node || x2 == cb_error_node) {
```

```
                return cb_error_node;
        }
        if (CB_LITERAL_P (x1)) {
                data1 = CB_LITERAL (x1)->data;
                size1 = CB_LITERAL (x1)->size;
        } else if (CB_CONST_P (x1)) {
                size1 = 1;
                if (x1 == cb_space) {
                        data1 = (unsigned char *)" ";
                } else if (x1 == cb_zero) {
                        data1 = (unsigned char *)"0";
                } else if (x1 == cb_quote) {
                        data1 = (unsigned char *)"\"";
                } else if (x1 == cb_norm_low) {
                        data1 = (unsigned char *)"\0";
                } else if (x1 == cb_norm_high) {
                        data1 = (unsigned char *)"\255";
                } else if (x1 == cb_null) {
                        data1 = (unsigned char *)"\0";
                } else {
                        return cb_error_node;
                }
        } else {
                return cb_error_node;
        }
        if (CB_LITERAL_P (x2)) {
                data2 = CB_LITERAL (x2)->data;
                size2 = CB_LITERAL (x2)->size;
        } else if (CB_CONST_P (x2)) {
                size2 = 1;
                if (x2 == cb_space) {
                        data2 = (unsigned char *)" ";
                } else if (x2 == cb_zero) {
                        data2 = (unsigned char *)"0";
                } else if (x2 == cb_quote) {
                        data2 = (unsigned char *)"\"";
                } else if (x2 == cb_norm_low) {
                        data2 = (unsigned char *)"\0";
                } else if (x2 == cb_norm_high) {
                        data2 = (unsigned char *)"\255";
                } else if (x2 == cb_null) {
                        data2 = (unsigned char *)"\0";
                } else {
                        return cb_error_node;
                }
        } else {
                return cb_error_node;
        }
        buff = cobc_malloc (size1 + size2 + 3);
        memcpy (buff, data1, size1);
        memcpy (buff + size1, data2, size2);
        x = cb_build_alphanumeric_literal (buff, size1 + size2);
        free (buff);
        return x;
}
```

7.15.2.38 const char* cb_define (cb_tree *name*, cb_tree *val*)

Definition at line 1769 of file tree.c.

```
{
        struct cb_word *w;

        w = CB_REFERENCE (name)->word;
        w->items = cb_list_add (w->items, val);
        w->count++;
        val->source_file = name->source_file;
        val->source_line = name->source_line;
        CB_REFERENCE (name)->value = val;
        return w->name;
}
```

7.15.2.39 void cb_define_system_name (const char * *name*)

Definition at line 1783 of file tree.c.

```
{
        cb_tree x;

        x = cb_build_reference (name);
        if (CB_REFERENCE (x)->word->count == 0) {
                cb_define (x, lookup_system_name (name));
        }
}
```

7.15.2.40 struct cb_field* cb_field (cb_tree *x*) [read]

Definition at line 1458 of file tree.c.

```
{
        if (CB_REFERENCE_P (x)) {
                return CB_FIELD (cb_ref (x));
        } else {
                return CB_FIELD (x);
        }
}
```

7.15.2.41 struct cb_field* cb_field_add (struct cb_field * *f*, struct cb_field * *p*) [read]

Definition at line 1468 of file tree.c.

```
{
        struct cb_field *t;

        if (f == NULL) {
                return p;
        }
        for (t = f; t->sister; t = t->sister) {
                ;
        }
        t->sister = p;
        return f;
}
```

7.15.2.42 struct cb_field* cb_field_founder (struct cb_field * f) [read]

Definition at line 1521 of file tree.c.

```
{
        while (f->parent) {
                f = f->parent;
        }
        return f;
}
```

7.15.2.43 int cb_field_size (cb_tree x)

Definition at line 1483 of file tree.c.

```
{
        struct cb_reference     *r;
        struct cb_field         *f;

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_LITERAL:
                return CB_LITERAL (x)->size;
        case CB_TAG_FIELD:
                return CB_FIELD (x)->size;
        case CB_TAG_REFERENCE:
                r = CB_REFERENCE (x);
                f = CB_FIELD (r->value);

                if (r->length) {
                        if (CB_LITERAL_P (r->length)) {
                                return cb_get_int (r->length);
                        } else {
                                return -1;
                        }
                } else if (r->offset) {
                        if (CB_LITERAL_P (r->offset)) {
                                return f->size - cb_get_int (r->offset) + 1;
                        } else {
                                return -1;
                        }
                } else {
                        return f->size;
                }
        default:
                fprintf (stderr, "Unexpected tree tag %d\n", CB_TREE_TAG (x));
                ABORT ();
        }
/* NOT REACHED */
        return 0;
}
```

7.15.2.44 int cb_field_subordinate (struct cb_field * p, struct cb_field * f)

Definition at line 1562 of file tree.c.

```
{
        for (p = p->parent; p; p = p->parent) {
                if (p == f) {
                        return 1;
                }
        }
        return 0;
}
```

7.15.2.45 struct cb_field* cb_field_variable_address (struct cb_field * f) [read]

Definition at line 1545 of file tree.c.

```
{
        struct cb_field *p;

        for (p = f->parent; p; f = f->parent, p = f->parent) {
                for (p = p->children; p != f; p = p->sister) {
                        if (p->occurs_depending || cb_field_variable_size (p)) {
                                return p;
                        }
                }
        }
        return NULL;
}
```

7.15.2.46 struct cb_field* cb_field_variable_size (struct cb_field * f) [read]

Definition at line 1530 of file tree.c.

```
{
        struct cb_field *p;

        for (f = f->children; f; f = f->sister) {
                if (f->occurs_depending) {
                        return f;
                } else if ((p = cb_field_variable_size (f)) != NULL) {
                        return p;
                }
        }
        return NULL;
}
```

7.15.2.47 int cb_fits_int (cb_tree x)

Definition at line 587 of file tree.c.

```
{
        struct cb_literal       *l;
        struct cb_field         *f;

        switch (CB_TREE_TAG (x)) {
```

```
        case CB_TAG_LITERAL:
                l = CB_LITERAL (x);
                if (l->scale <= 0 && l->size < 10) {
                        return 1;
                }
                return 0;
        case CB_TAG_FIELD:
                f = CB_FIELD (x);
                switch (f->usage) {
                case CB_USAGE_INDEX:
                case CB_USAGE_LENGTH:
                        return 1;
                case CB_USAGE_BINARY:
                case CB_USAGE_COMP_5:
                case CB_USAGE_COMP_X:
                        if (f->pic->scale <= 0 && f->size <= (int)sizeof (int)) {

                                return 1;
                        }
                        return 0;
                case CB_USAGE_DISPLAY:
                        if (f->size < 10) {
                                if (!f->pic || f->pic->scale <= 0) {
                                        return 1;
                                }
                        }
                        return 0;
                case CB_USAGE_PACKED:
                        if (f->pic->scale <= 0 && f->pic->digits < 10) {
                                return 1;
                        }
                        return 0;
                default:
                        return 0;
                }
        case CB_TAG_REFERENCE:
                return cb_fits_int (CB_REFERENCE (x)->value);
        default:
                return 0;
        }
}
```

7.15.2.48 int cb_fits_long_long (cb_tree *x*)

Definition at line 635 of file tree.c.

```
{
        struct cb_literal       *l;
        struct cb_field         *f;

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_LITERAL:
                l = CB_LITERAL (x);
                if (l->scale <= 0 && l->size < 19) {
                        return 1;
                }
                return 0;
        case CB_TAG_FIELD:
                f = CB_FIELD (x);
```

```
                switch (f->usage) {
                case CB_USAGE_INDEX:
                case CB_USAGE_LENGTH:
                        return 1;
                case CB_USAGE_BINARY:
                case CB_USAGE_COMP_5:
                case CB_USAGE_COMP_X:
                        if (f->pic->scale <= 0 && f->size <= (int)sizeof (long lo
      ng)) {
                                return 1;
                        }
                        return 0;
                case CB_USAGE_DISPLAY:
                        if (f->pic->scale <= 0 && f->size < 19) {
                                return 1;
                        }
                        return 0;
                default:
                        return 0;
                }
        case CB_TAG_REFERENCE:
                return cb_fits_long_long (CB_REFERENCE (x)->value);
        default:
                return 0;
        }
}
```

7.15.2.49 int cb_get_int (cb_tree *x*)

Definition at line 676 of file tree.c.

```
{
        struct cb_literal       *l;
        size_t                  i;
        int                     val = 0;

        l = CB_LITERAL (x);
        for (i = 0; i < l->size; i++) {
                if (l->data[i] != '0') {
                        break;
                }
        }

/* RXWRXW
        if (l->size - i >= 10) {
                ABORT ();
        }
*/

        for (; i < l->size; i++) {
                val = val * 10 + l->data[i] - '0';
        }
        if (l->sign < 0) {
                val = -val;
        }
        return val;
}
```

7.15.2.50 long long cb_get_long_long (cb_tree *x*)

Definition at line 705 of file tree.c.

```
{
        struct cb_literal       *l;
        size_t                  i;
        long long               val = 0;

        l = CB_LITERAL (x);
        for (i = 0; i < l->size; i++) {
                if (l->data[i] != '0') {
                        break;
                }
        }

        if (l->size - i >= 19) {
                ABORT ();
        }

        for (; i < l->size; i++) {
                val = val * 10 + l->data[i] - '0';
        }
        if (l->sign < 0) {
                val = -val;
        }
        return val;
}
```

7.15.2.51 void cb_init_constants (void)

Definition at line 732 of file tree.c.

```
{
        char    *s;
        int     i;

        cb_error_node = make_constant (CB_CATEGORY_UNKNOWN, NULL);
        cb_any = make_constant (CB_CATEGORY_UNKNOWN, NULL);
        cb_true = make_constant (CB_CATEGORY_BOOLEAN, "1");
        cb_false = make_constant (CB_CATEGORY_BOOLEAN, "0");
        cb_null = make_constant (CB_CATEGORY_DATA_POINTER, "0");
        cb_zero = make_constant (CB_CATEGORY_NUMERIC, "&cob_zero");
        cb_one = make_constant (CB_CATEGORY_NUMERIC, "&cob_one");
        cb_space = make_constant (CB_CATEGORY_ALPHANUMERIC, "&cob_space");
        cb_low = make_constant (CB_CATEGORY_ALPHANUMERIC, "&cob_low");
        cb_norm_low = cb_low;
        cb_high = make_constant (CB_CATEGORY_ALPHANUMERIC, "&cob_high");
        cb_norm_high = cb_high;
        cb_quote = make_constant (CB_CATEGORY_ALPHANUMERIC, "&cob_quote");
        cb_int0 = cb_int (0);
        cb_int1 = cb_int (1);
        cb_int2 = cb_int (2);
        cb_int3 = cb_int (3);
        cb_int4 = cb_int (4);
        cb_int5 = cb_int (5);
        for (i = 1; i < 8; i++) {
                s = cobc_malloc (4);
```

```
                sprintf (s, "i%d", i);
                cb_i[i] = make_constant (CB_CATEGORY_NUMERIC, s);
        }
        cb_standard_error_handler = make_constant_label ("Default Error Handler")
      ;
}
```

7.15.2.52 cb_tree cb_int (int *n*)

Definition at line 881 of file tree.c.

```
{
        struct cb_integer       *x;
        struct int_node         *p;

        for (p = int_node_table; p; p = p->next) {
                if (p->n == n) {
                        return p->node;
                }
        }

        x = make_tree (CB_TAG_INTEGER, CB_CATEGORY_NUMERIC, sizeof (struct
      cb_integer));
        x->val = n;

        p = cobc_malloc (sizeof (struct int_node));
        p->n = n;
        p->node = CB_TREE (x);
        p->next = int_node_table;
        int_node_table = p;
        return p->node;
}
```

7.15.2.53 cb_tree cb_list_add (cb_tree *l*, cb_tree *x*)

Definition at line 798 of file tree.c.

```
{
        return cb_list_append (l, cb_list_init (x));
}
```

7.15.2.54 cb_tree cb_list_append (cb_tree *l1*, cb_tree *l2*)

Definition at line 781 of file tree.c.

```
{
        cb_tree l;

        if (l1 == NULL) {
                return l2;
        } else {
                l = l1;
```

```
                while (CB_CHAIN (l)) {
                        l = CB_CHAIN (l);
                }
                CB_CHAIN (l) = l2;
                return l1;
        }
}
```

7.15.2.55 int cb_list_length (cb_tree *l*)

Definition at line 818 of file tree.c.

```
{
        int n = 0;

        for (; l; l = CB_CHAIN (l)) {
                n++;
        }
        return n;
}
```

7.15.2.56 void cb_list_map (cb_tree(*)(cb_tree x) *func,* cb_tree *l*)

Definition at line 829 of file tree.c.

```
{
        for (; l; l = CB_CHAIN (l)) {
                CB_VALUE (l) = func (CB_VALUE (l));
        }
}
```

7.15.2.57 cb_tree cb_list_reverse (cb_tree *l*)

Definition at line 804 of file tree.c.

```
{
        cb_tree next;
        cb_tree last = NULL;

        for (; l; l = next) {
                next = CB_CHAIN (l);
                CB_CHAIN (l) = last;
                last = l;
        }
        return last;
}
```

7.15.2.58 char* cb_name (cb_tree *x*)

Definition at line 441 of file tree.c.

```
{
        if (!treenamebuff) {
                treenamebuff = cobc_malloc (COB_NORMAL_BUFF);
        }
        cb_name_1 (treenamebuff, x);
        return treenamebuff;
}
```

7.15.2.59 cb_tree cb_ref (cb_tree *x*)

Definition at line 1794 of file tree.c.

```
{
        struct cb_reference     *r;
        struct cb_field         *p;
        struct cb_label         *s;
        cb_tree                 candidate = NULL;
        cb_tree                 items;
        cb_tree                 cb1;
        cb_tree                 cb2;
        cb_tree                 v;
        cb_tree                 c;
        struct cb_program       *prog;
        struct cb_word          *w;
        size_t                  val;
        size_t                  ambiguous = 0;

        r = CB_REFERENCE (x);
        /* if this reference has already been resolved (and the value
           has been cached), then just return the value */
        if (r->value) {
                return r->value;
        }
        /* resolve the value */

        items = r->word->items;
        for (; items; items = CB_CHAIN (items)) {
                /* find a candidate value by resolving qualification */
                v = CB_VALUE (items);
                c = r->chain;
                switch (CB_TREE_TAG (v)) {
                case CB_TAG_FIELD:
                        /* in case the value is a field, it might be qualified
                           by its parent names and a file name */
                        if (CB_FIELD (v)->flag_indexed_by) {
                                p = CB_FIELD (v)->index_qual;
                        } else {
                                p = CB_FIELD (v)->parent;
                        }
                        /* resolve by parents */
                        for (; p; p = p->parent) {
                                if (c && strcasecmp (CB_NAME (c), p->name) == 0)
      {
                                        c = CB_REFERENCE (c)->chain;
```

```
                        }
                }

                /* resolve by file */
                if (c && CB_REFERENCE (c)->chain == NULL) {
                        if (CB_REFERENCE (c)->word->count == 1 &&
CB_FILE_P (cb_ref (c))
                            && (CB_FILE (cb_ref (c)) == cb_field_founder
(CB_FIELD (v))->file)) {
                                c = CB_REFERENCE (c)->chain;
                        }
                }

                break;
        case CB_TAG_LABEL:
                /* in case the value is a label, it might be qualified
                   by its section name */
                s = CB_LABEL (v)->section;

                /* unqualified paragraph name referenced within the secti
on
                   is resolved without ambiguity check if not duplicated
*/
                if (c == NULL && r->offset && s == CB_LABEL (r->offset))
{
                        for (cb1 = CB_CHAIN (items); cb1; cb1 = CB_CHAIN
(cb1)) {
                                cb2 = CB_VALUE (cb1);
                                if (s == CB_LABEL (cb2)->section) {
                                        ambiguous_error (x);
                                        goto error;
                                }
                        }
                        candidate = v;
                        goto end;
                }

                /* resolve by section name */
                if (c && s && strcasecmp (CB_NAME (c), (char *)s->name) =
= 0) {
                        c = CB_REFERENCE (c)->chain;
                }

                break;
        default:
                /* other values cannot be qualified */
                break;
        }

        /* a well qualified value is a good candidate */
        if (c == NULL) {
                if (candidate == NULL) {
                        /* keep the first candidate */
                        candidate = v;
                } else {
                        /* multiple candidates and possibly ambiguous */
                        ambiguous = 1;
                        /* continue search because the reference might no
t
                           be ambiguous and exit loop by "goto end" later
 */
                }
```

```
                }
        }

        /* there is no candidate */
        if (candidate == NULL) {
                if (current_program->nested_level > 0) {
                        /* Nested program - check parents for GLOBAL candidate */

                        ambiguous = 0;
                        val = hash ((const unsigned char *)r->word->name);
                        prog = current_program->next_program;
                        for (; prog; prog = prog->next_program) {
                                if (prog->nested_level >= current_program->
      nested_level) {
                                        continue;
                                }
                                for (w = prog->word_table[val]; w; w = w->next) {

                                        if (strcasecmp (r->word->name, w->name) =
      = 0) {
                                                candidate = global_check (r, w->
      items, &ambiguous);
                                                if (candidate) {
                                                        if (ambiguous) {
                                                                ambiguous_error (
      x);
                                                                goto error;
                                                        }
                                                        if (CB_FILE_P(candidate))
       {
                                                                current_program->
      gen_file_error = 1;
                                                        }
                                                        goto end;
                                                }
                                        }
                                }
                                if (prog->nested_level == 0) {
                                        break;
                                }
                        }
                }
                undefined_error (x);
                goto error;
        }

        /* the reference is ambiguous */
        if (ambiguous) {
                ambiguous_error (x);
                goto error;
        }

end:
        if (CB_FIELD_P (candidate)) {
                CB_FIELD (candidate)->count++;
                if (CB_FIELD (candidate)->flag_invalid) {
                        goto error;
                }
        }

        r->value = candidate;
        return r->value;
```

```
error:
        r->value = cb_error_node;
        return cb_error_node;
}
```

7.15.2.60 enum cb_category cb_tree_category (cb_tree *x*)

Definition at line 458 of file tree.c.

```
{
        struct cb_cast          *p;
        struct cb_reference     *r;
        struct cb_field         *f;

        if (x == cb_error_node) {
                return 0;
        }
        if (x->category != CB_CATEGORY_UNKNOWN) {
                return x->category;
        }

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_CAST:
                p = CB_CAST (x);
                switch (p->type) {
                case CB_CAST_ADDRESS:
                case CB_CAST_ADDR_OF_ADDR:
                        x->category = CB_CATEGORY_DATA_POINTER;
                        break;
                case CB_CAST_PROGRAM_POINTER:
                        x->category = CB_CATEGORY_PROGRAM_POINTER;
                        break;
                default:
                        fprintf (stderr, "Unexpected cast type -> %d\n", p->type)
      ;
                        ABORT ();
                }
                break;
          case CB_TAG_REFERENCE:
                r = CB_REFERENCE (x);
                if (r->offset) {
                        x->category = CB_CATEGORY_ALPHANUMERIC;
                } else {
                        x->category = cb_tree_category (r->value);
                }
                break;
          case CB_TAG_FIELD:
                f = CB_FIELD (x);
                if (f->children) {
                        x->category = CB_CATEGORY_ALPHANUMERIC;
                } else if (f->usage == CB_USAGE_POINTER && f->level != 88) {
                        x->category = CB_CATEGORY_DATA_POINTER;
                } else if (f->usage == CB_USAGE_PROGRAM_POINTER && f->level != 88
      ) {
                        x->category = CB_CATEGORY_PROGRAM_POINTER;
                } else {
                        switch (f->level) {
                        case 66:
```

```
                                if (f->rename_thru) {
                                        x->category = CB_CATEGORY_ALPHANUMERIC;
                                } else {
                                        x->category = cb_tree_category (CB_TREE (
      f->redefines));
                                }
                                break;
                        case 88:
                                x->category = CB_CATEGORY_BOOLEAN;
                                break;
                        default:
                                x->category = f->pic->category;
                                break;
                        }
                }
                break;
        case CB_TAG_ALPHABET_NAME:
        case CB_TAG_LOCALE_NAME:
                x->category = CB_CATEGORY_ALPHANUMERIC;
                break;
        case CB_TAG_BINARY_OP:
                x->category = CB_CATEGORY_BOOLEAN;
                break;
        default:
                fprintf (stderr, "Unknown tree tag %d Category %d\n",
      CB_TREE_TAG (x), x->category);
                ABORT ();
        }

        return x->category;
}
```

7.15.2.61 enum cb_class cb_tree_class (cb_tree *x*)

Definition at line 451 of file tree.c.

```
{

        return category_to_class_table[CB_TREE_CATEGORY (x)];
}
```

7.15.2.62 int cb_tree_type (cb_tree *x*)

Definition at line 537 of file tree.c.

```
{
        struct cb_field *f;

        f = cb_field (x);
        if (f->children) {
                return COB_TYPE_GROUP;
        }

        switch (CB_TREE_CATEGORY (x)) {
        case CB_CATEGORY_ALPHABETIC:
```

```
        case CB_CATEGORY_ALPHANUMERIC:
                return COB_TYPE_ALPHANUMERIC;
        case CB_CATEGORY_ALPHANUMERIC_EDITED:
                return COB_TYPE_ALPHANUMERIC_EDITED;
        case CB_CATEGORY_NUMERIC:
                switch (f->usage) {
                case CB_USAGE_DISPLAY:
                        return COB_TYPE_NUMERIC_DISPLAY;
                case CB_USAGE_BINARY:
                case CB_USAGE_COMP_5:
                case CB_USAGE_COMP_X:
                case CB_USAGE_INDEX:
                case CB_USAGE_LENGTH:
                        return COB_TYPE_NUMERIC_BINARY;
                case CB_USAGE_FLOAT:
                        return COB_TYPE_NUMERIC_FLOAT;
                case CB_USAGE_DOUBLE:
                        return COB_TYPE_NUMERIC_DOUBLE;
                case CB_USAGE_PACKED:
                        return COB_TYPE_NUMERIC_PACKED;
                default:
                        fprintf (stderr, "Unexpected numeric usage -> %d\n", f->
      usage);
                        ABORT ();
                }
        case CB_CATEGORY_NUMERIC_EDITED:
                return COB_TYPE_NUMERIC_EDITED;
        case CB_CATEGORY_OBJECT_REFERENCE:
        case CB_CATEGORY_DATA_POINTER:
        case CB_CATEGORY_PROGRAM_POINTER:
                return COB_TYPE_NUMERIC_BINARY;
        default:
                fprintf (stderr, "Unexpected category -> %d\n", CB_TREE_CATEGORY
      (x));
                ABORT ();
        }
/* NOT REACHED */
        return 0;
}
```

7.15.2.63 void finalize_file (struct cb_file * *f*, struct cb_field * *records*)

Definition at line 1611 of file tree.c.

```
{
        struct cb_field *p;
        struct cb_field *v;
        cb_tree         l;
        cb_tree         x;
        char            buff[COB_MINI_BUFF];

        if (f->special) {
                f->organization = COB_ORG_LINE_SEQUENTIAL;
        }
        if (f->fileid_assign && !f->assign) {
                f->assign = cb_build_alphanumeric_literal ((unsigned char *)f->
      name,
                                                           strlen (f->name));
        }
```

```
  /* check the record size if it is limited */
  for (p = records; p; p = p->sister) {
          if (f->record_min > 0) {
                  if (p->size < f->record_min) {
                          cb_error (_("Record size too small '%s'"), p->
name);
                  }
          }
          if (f->record_max > 0) {
                  if (p->size > f->record_max) {
                          cb_error (_("Record size too large '%s' (%d)"),
                                  p->name, p->size);
                  }
          }
  }

  /* compute the record size */
  if (f->record_min == 0) {
          if (records) {
                  f->record_min = records->size;
          } else {
                  f->record_min = 0;
          }
  }
  for (p = records; p; p = p->sister) {
          v = cb_field_variable_size (p);
          if (v && v->offset + v->size * v->occurs_min < f->record_min) {
                  f->record_min = v->offset + v->size * v->occurs_min;
          }
          if (p->size < f->record_min) {
                  f->record_min = p->size;
          }
          if (p->size > f->record_max) {
                  f->record_max = p->size;
          }
  }

  if (f->same_clause) {
          for (l = current_program->file_list; l; l = CB_CHAIN (l)) {
                  if (CB_FILE (CB_VALUE (l))->same_clause == f->
same_clause) {
                          if (CB_FILE (CB_VALUE (l))->finalized) {
                                  if (f->record_max > CB_FILE (CB_VALUE (l)
)->record->memory_size) {
                                          CB_FILE (CB_VALUE (l))->record->m
emory_size =
                                              f->record_max;
                                  }
                                  f->record = CB_FILE (CB_VALUE (l))->recor
d;
                                  for (p = records; p; p = p->sister) {
                                          p->file = f;
                                          p->redefines = f->record;
                                  }
                                  for (p = f->record->sister; p; p = p->
sister) {
                                          if (!p->sister) {
                                                  p->sister = records;
                                                  break;
                                          }
                                  }
```

```
                                        f->finalized = 1;
                                        return;
                                }
                        }
                }
        }
        /* create record */
        snprintf (buff, COB_MINI_MAX, "%s_record", f->name);
        if (f->record_max == 0) {
                f->record_max = 32;
                f->record_min = 32;
        }
        if (f->organization == COB_ORG_LINE_SEQUENTIAL) {
                f->record_min = 0;
        }
        f->record = CB_FIELD (cb_build_implicit_field (cb_build_reference (buff),

                              f->record_max));
        f->record->sister = records;
        f->record->count++;
        if (f->external) {
                has_external = 1;
                f->record->flag_external = 1;
        }

        for (p = records; p; p = p->sister) {
                p->file = f;
                p->redefines = f->record;
        }
        f->finalized = 1;
        if (f->linage) {
                snprintf (buff, COB_MINI_MAX, "LC_%s", f->name);
                x = cb_build_field (cb_build_reference (buff));
                CB_FIELD (x)->pic = CB_PICTURE (cb_build_picture ("9(9)"));
                CB_FIELD (x)->usage = CB_USAGE_COMP_5;
                CB_FIELD (x)->values = cb_list_init (cb_zero);
                CB_FIELD (x)->count++;
                cb_validate_field (CB_FIELD (x));
                f->linage_ctr = cb_build_field_reference (CB_FIELD (x), NULL);
                current_program->working_storage =
                    cb_field_add (current_program->working_storage, CB_FIELD (x))
      ;
        }
}
```

7.15.2.64 void validate_file (struct cb_file * *f*, cb_tree *name*)

Definition at line 1593 of file tree.c.

```
{
        /* check RECORD/RELATIVE KEY clause */
        switch (f->organization) {
        case COB_ORG_INDEXED:
                if (f->key == NULL) {
                        file_error (name, "RECORD KEY");
                }
                break;
        case COB_ORG_RELATIVE:
                if (f->key == NULL && f->access_mode != COB_ACCESS_SEQUENTIAL) {
```

```
                        file_error (name, "RELATIVE KEY");
                }
                break;
        }
}
```

7.15.3 Variable Documentation

7.15.3.1 cb_tree cb_any

Definition at line 74 of file tree.c.

7.15.3.2 cb_tree cb_error_node

Definition at line 93 of file tree.c.

7.15.3.3 cb_tree cb_false

Definition at line 76 of file tree.c.

7.15.3.4 cb_tree cb_high

Definition at line 82 of file tree.c.

7.15.3.5 cb_tree cb_i[8]

Definition at line 92 of file tree.c.

7.15.3.6 cb_tree cb_int0

Definition at line 86 of file tree.c.

7.15.3.7 cb_tree cb_int1

Definition at line 87 of file tree.c.

7.15.3.8 cb_tree cb_int2

Definition at line 88 of file tree.c.

7.15.3.9 cb_tree cb_int3

Definition at line 89 of file tree.c.

7.15.3.10 cb_tree cb_int4

Definition at line 90 of file tree.c.

7.15.3.11 cb_tree cb_int5

Definition at line 91 of file tree.c.

7.15.3.12 cb_tree cb_intr_e

Definition at line 97 of file tree.c.

7.15.3.13 cb_tree cb_intr_pi

Definition at line 96 of file tree.c.

7.15.3.14 cb_tree cb_intr_whencomp

Definition at line 95 of file tree.c.

7.15.3.15 cb_tree cb_low

Definition at line 81 of file tree.c.

7.15.3.16 cb_tree cb_norm_high

Definition at line 84 of file tree.c.

7.15.3.17 cb_tree cb_norm_low

Definition at line 83 of file tree.c.

7.15.3.18 cb_tree cb_null

Definition at line 77 of file tree.c.

7.15.3.19 cb_tree cb_one

Definition at line 79 of file tree.c.

7.15.3.20 cb_tree cb_quote

Definition at line 85 of file tree.c.

7.15.3.21 cb_tree cb_space

Definition at line 80 of file tree.c.

7.15.3.22 cb_tree cb_standard_error_handler

Definition at line 99 of file tree.c.

7.15.3.23 cb_tree cb_true

Definition at line 75 of file tree.c.

7.15.3.24 cb_tree cb_zero

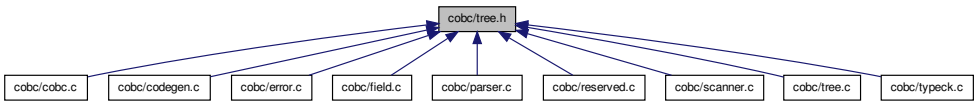
Definition at line 78 of file tree.c.

7.15.3.25 size_t gen_screen_ptr = 0

Definition at line 101 of file tree.c.

7.16 cobc/tree.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct cb_tree_common
- struct cb_const
- struct cb_integer
- struct cb_string
- struct cb_alphabet_name

- struct cb_class_name
- struct cb_locale_name
- struct cb_system_name
- struct cb_literal
- struct cb_decimal
- struct cb_picture
- struct cb_field
- struct cb_field::cb_key
- struct cb_label
- struct handler_struct
- struct cb_alt_key
- struct cb_file
- struct cb_word
- struct cb_reference
- struct cb_binary_op
- struct cb_funcall
- struct cb_cast
- struct cb_assign
- struct cb_intrinsic_table
- struct cb_intrinsic
- struct cb_initialize
- struct cb_search
- struct cb_call
- struct cb_goto
- struct cb_if
- struct cb_perform_varying
- struct cb_perform
- struct cb_statement
- struct cb_continue
- struct cb_list
- struct cb_program

Defines

- #define YYSTYPE cb_tree
- #define CB_BEFORE cb_int0
- #define CB_AFTER cb_int1
- #define COB_MAX_SUBSCRIPTS 16
- #define CB_PREFIX_ATTR "a_"
- #define CB_PREFIX_BASE "b_"
- #define CB_PREFIX_CONST "c_"
- #define CB_PREFIX_DECIMAL "d_"
- #define CB_PREFIX_FIELD "f_"
- #define CB_PREFIX_FILE "h_"
- #define CB_PREFIX_KEYS "k_"
- #define CB_PREFIX_LABEL "l_"

- #define CB_PREFIX_SEQUENCE "s_"
- #define CB_PROGRAM_TYPE 0
- #define CB_FUNCTION_TYPE 1
- #define CB_TREE(x) ((struct cb_tree_common *) (x))
- #define CB_TREE_TAG(x) (CB_TREE (x)->tag)
- #define CB_TREE_CLASS(x) cb_tree_class (CB_TREE (x))
- #define CB_TREE_CATEGORY(x) cb_tree_category (CB_TREE (x))
- #define CB_TREE_CAST(tg, ty, x) ((ty *) (x))
- #define CB_CONST(x) (CB_TREE_CAST (CB_TAG_CONST, struct cb_const, x))
- #define CB_CONST_P(x) (CB_TREE_TAG (x) == CB_TAG_CONST)
- #define CB_INTEGER(x) (CB_TREE_CAST (CB_TAG_INTEGER, struct cb_integer, x))
- #define CB_INTEGER_P(x) (CB_TREE_TAG (x) == CB_TAG_INTEGER)
- #define CB_STRING(x) (CB_TREE_CAST (CB_TAG_STRING, struct cb_string, x))
- #define CB_STRING_P(x) (CB_TREE_TAG (x) == CB_TAG_STRING)
- #define cb_build_string0(str) cb_build_string (str, strlen ((char *)str))
- #define CB_ALPHABET_NAME(x) (CB_TREE_CAST (CB_TAG_ALPHABET_NAME, struct cb_alphabet_name, x))
- #define CB_ALPHABET_NAME_P(x) (CB_TREE_TAG (x) == CB_TAG_ALPHABET_NAME)
- #define CB_CLASS_NAME(x) (CB_TREE_CAST (CB_TAG_CLASS_NAME, struct cb_class_name, x))
- #define CB_CLASS_NAME_P(x) (CB_TREE_TAG (x) == CB_TAG_CLASS_NAME)
- #define CB_LOCALE_NAME(x) (CB_TREE_CAST (CB_TAG_LOCALE_NAME, struct cb_locale_name, x))
- #define CB_LOCALE_NAME_P(x) (CB_TREE_TAG (x) == CB_TAG_LOCALE_NAME)
- #define CB_SYSTEM_NAME(x) (CB_TREE_CAST (CB_TAG_SYSTEM_NAME, struct cb_system_name, x))
- #define CB_SYSTEM_NAME_P(x) (CB_TREE_TAG (x) == CB_TAG_SYSTEM_NAME)
- #define CB_LITERAL(x) (CB_TREE_CAST (CB_TAG_LITERAL, struct cb_literal, x))
- #define CB_LITERAL_P(x) (CB_TREE_TAG (x) == CB_TAG_LITERAL)
- #define CB_NUMERIC_LITERAL_P(x) (CB_LITERAL_P (x) && CB_TREE_CATEGORY (x) == CB_CATEGORY_NUMERIC)
- #define CB_DECIMAL(x) (CB_TREE_CAST (CB_TAG_DECIMAL, struct cb_decimal, x))
- #define CB_DECIMAL_P(x) (CB_TREE_TAG (x) == CB_TAG_DECIMAL)
- #define CB_PICTURE(x) (CB_TREE_CAST (CB_TAG_PICTURE, struct cb_picture, x))
- #define CB_PICTURE_P(x) (CB_TREE_TAG (x) == CB_TAG_PICTURE)
- #define CB_FIELD(x) (CB_TREE_CAST (CB_TAG_FIELD, struct cb_field, x))
- #define CB_FIELD_P(x) (CB_TREE_TAG (x) == CB_TAG_FIELD)
- #define CB_REF_OR_FIELD_P(x) ((CB_FIELD_P (x) || CB_REFERENCE_P (x)))

- #define CB_INDEX_P(x)
- #define CB_LABEL(x) (CB_TREE_CAST (CB_TAG_LABEL, struct cb_label, x))
- #define CB_LABEL_P(x) (CB_TREE_TAG (x) == CB_TAG_LABEL)
- #define CB_FILE(x) (CB_TREE_CAST (CB_TAG_FILE, struct cb_file, x))
- #define CB_FILE_P(x) (CB_TREE_TAG (x) == CB_TAG_FILE)
- #define CB_WORD_HASH_SIZE 133
- #define CB_REFERENCE(x) (CB_TREE_CAST (CB_TAG_REFERENCE, struct cb_reference, x))
- #define CB_REFERENCE_P(x) (CB_TREE_TAG (x) == CB_TAG_REFERENCE)
- #define CB_NAME(x) (CB_REFERENCE (x)->word->name)
- #define CB_BINARY_OP(x) (CB_TREE_CAST (CB_TAG_BINARY_OP, struct cb_binary_op, x))
- #define CB_BINARY_OP_P(x) (CB_TREE_TAG (x) == CB_TAG_BINARY_OP)
- #define cb_build_parenthesis(x) cb_build_binary_op (x, '@', NULL)
- #define cb_build_negation(x) cb_build_binary_op (x, '!', NULL)
- #define CB_FUNCALL(x) (CB_TREE_CAST (CB_TAG_FUNCALL, struct cb_funcall, x))
- #define CB_FUNCALL_P(x) (CB_TREE_TAG (x) == CB_TAG_FUNCALL)
- #define cb_build_funcall_0(f) cb_build_funcall(f, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL)
- #define cb_build_funcall_1(f, a1) cb_build_funcall(f, 1, a1, NULL, NULL, NULL, NULL, NULL, NULL)
- #define cb_build_funcall_2(f, a1, a2) cb_build_funcall(f, 2, a1, a2, NULL, NULL, NULL, NULL, NULL)
- #define cb_build_funcall_3(f, a1, a2, a3) cb_build_funcall(f, 3, a1, a2, a3, NULL, NULL, NULL, NULL)
- #define cb_build_funcall_4(f, a1, a2, a3, a4) cb_build_funcall(f, 4, a1, a2, a3, a4, NULL, NULL, NULL)
- #define cb_build_funcall_5(f, a1, a2, a3, a4, a5) cb_build_funcall(f, 5, a1, a2, a3, a4, a5, NULL, NULL)
- #define cb_build_funcall_6(f, a1, a2, a3, a4, a5, a6) cb_build_funcall(f, 6, a1, a2, a3, a4, a5, a6, NULL)
- #define cb_build_funcall_7(f, a1, a2, a3, a4, a5, a6, a7) cb_build_funcall(f, 7, a1, a2, a3, a4, a5, a6, a7)
- #define CB_CAST(x) (CB_TREE_CAST (CB_TAG_CAST, struct cb_cast, x))
- #define CB_CAST_P(x) (CB_TREE_TAG (x) == CB_TAG_CAST)
- #define cb_build_cast_integer(x) cb_build_cast (CB_CAST_INTEGER, x)
- #define cb_build_cast_address(x) cb_build_cast (CB_CAST_ADDRESS, x)
- #define cb_build_cast_addr_of_addr(x) cb_build_cast (CB_CAST_ADDR_OF_ADDR, x)
- #define cb_build_cast_length(x) cb_build_cast (CB_CAST_LENGTH, x)
- #define cb_build_cast_ppointer(x) cb_build_cast (CB_CAST_PROGRAM_POINTER, x)
- #define CB_ASSIGN(x) (CB_TREE_CAST (CB_TAG_ASSIGN, struct cb_assign, x))
- #define CB_ASSIGN_P(x) (CB_TREE_TAG (x) == CB_TAG_ASSIGN)
- #define CB_INTRINSIC(x) (CB_TREE_CAST (CB_TAG_INTRINSIC, struct cb_intrinsic, x))

- #define CB_INTRINSIC_P(x) (CB_TREE_TAG (x) == CB_TAG_INTRINSIC)
- #define CB_INITIALIZE(x) (CB_TREE_CAST (CB_TAG_INITIALIZE, struct cb_initialize, x))
- #define CB_INITIALIZE_P(x) (CB_TREE_TAG (x) == CB_TAG_INITIALIZE)
- #define CB_SEARCH(x) (CB_TREE_CAST (CB_TAG_SEARCH, struct cb_search, x))
- #define CB_SEARCH_P(x) (CB_TREE_TAG (x) == CB_TAG_SEARCH)
- #define CB_CALL_BY_REFERENCE 1
- #define CB_CALL_BY_CONTENT 2
- #define CB_CALL_BY_VALUE 3
- #define CB_CALL(x) (CB_TREE_CAST (CB_TAG_CALL, struct cb_call, x))
- #define CB_CALL_P(x) (CB_TREE_TAG (x) == CB_TAG_CALL)
- #define CB_GOTO(x) (CB_TREE_CAST (CB_TAG_GOTO, struct cb_goto, x))
- #define CB_GOTO_P(x) (CB_TREE_TAG (x) == CB_TAG_GOTO)
- #define CB_IF(x) (CB_TREE_CAST (CB_TAG_IF, struct cb_if, x))
- #define CB_IF_P(x) (CB_TREE_TAG (x) == CB_TAG_IF)
- #define CB_PERFORM_VARYING(x) (CB_TREE_CAST (CB_TAG_PERFORM_VARYING, struct cb_perform_varying, x))
- #define CB_PERFORM(x) (CB_TREE_CAST (CB_TAG_PERFORM, struct cb_perform, x))
- #define CB_PERFORM_P(x) (CB_TREE_TAG (x) == CB_TAG_PERFORM)
- #define CB_STATEMENT(x) (CB_TREE_CAST (CB_TAG_STATEMENT, struct cb_statement, x))
- #define CB_STATEMENT_P(x) (CB_TREE_TAG (x) == CB_TAG_STATEMENT)
- #define CB_CONTINUE(x) (CB_TREE_CAST (CB_TAG_CONTINUE, struct cb_continue, x))
- #define CB_CONTINUE_P(x) (CB_TREE_TAG (x) == CB_TAG_CONTINUE)
- #define CB_LIST(x) (CB_TREE_CAST (CB_TAG_LIST, struct cb_list, x))
- #define CB_LIST_P(x) (CB_TREE_TAG (x) == CB_TAG_LIST)
- #define CB_PURPOSE(x) (CB_LIST (x)->purpose)
- #define CB_VALUE(x) (CB_LIST (x)->value)
- #define CB_CHAIN(x) (CB_LIST (x)->chain)
- #define CB_SIZES(x) (CB_LIST (x)->sizes)
- #define CB_PURPOSE_INT(x) (CB_INTEGER (CB_PURPOSE (x))->val)
- #define CB_SIZE_AUTO 0
- #define CB_SIZE_1 1
- #define CB_SIZE_2 2
- #define CB_SIZE_4 3
- #define CB_SIZE_8 4
- #define CB_SIZE_UNSIGNED 8
- #define CB_SIZES_INT(x) ((CB_LIST (x)->sizes) & 0x07)
- #define CB_SIZES_INT_UNSIGNED(x) ((CB_LIST (x)->sizes) & CB_SIZE_UNSIGNED)
- #define cb_list_init(x) cb_build_list (NULL, x, NULL)
- #define cb_cons(x, l) cb_build_list (NULL, x, l)
- #define CB_PAIR_P(x) (CB_LIST_P (x) && CB_PAIR_X (x))
- #define CB_PAIR_X(x) CB_PURPOSE (x)
- #define CB_PAIR_Y(x) CB_VALUE (x)
- #define cb_build_pair(x, y) cb_build_list (x, y, NULL)

Typedefs

- typedef struct cb_tree_common * cb_tree

Enumerations

- enum cb_tag {

  CB_TAG_CONST, CB_TAG_INTEGER, CB_TAG_STRING, CB_TAG_ALPHABET_NAME,

  CB_TAG_CLASS_NAME, CB_TAG_LOCALE_NAME, CB_TAG_SYSTEM_NAME, CB_TAG_LITERAL,

  CB_TAG_DECIMAL, CB_TAG_FIELD, CB_TAG_FILE, CB_TAG_REFERENCE,

  CB_TAG_BINARY_OP, CB_TAG_FUNCALL, CB_TAG_CAST, CB_TAG_INTRINSIC,

  CB_TAG_LABEL, CB_TAG_ASSIGN, CB_TAG_INITIALIZE, CB_TAG_SEARCH,

  CB_TAG_CALL, CB_TAG_GOTO, CB_TAG_IF, CB_TAG_PERFORM,

  CB_TAG_STATEMENT, CB_TAG_CONTINUE, CB_TAG_PERFORM_VARYING, CB_TAG_PICTURE,

  CB_TAG_LIST }
- enum cb_alphabet_name_type {

  CB_ALPHABET_NATIVE, CB_ALPHABET_STANDARD_1, CB_ALPHABET_STANDARD_2, CB_ALPHABET_EBCDIC,

  CB_ALPHABET_CUSTOM }
- enum cb_system_name_category {

  CB_CALL_CONVENTION_NAME, CB_CODE_NAME, CB_COMPUTER_NAME, CB_DEVICE_NAME,

  CB_ENTRY_CONVENTION_NAME, CB_EXTERNAL_LOCALE_NAME, CB_FEATURE_NAME, CB_LIBRARY_NAME,

  CB_SWITCH_NAME, CB_TEXT_NAME }
- enum cb_device_name { CB_DEVICE_SYSIN, CB_DEVICE_SYSOUT, CB_DEVICE_SYSERR, CB_DEVICE_CONSOLE }
- enum cb_feature_name {

  CB_FEATURE_FORMFEED, CB_FEATURE_C01, CB_FEATURE_C02, CB_FEATURE_C03,

  CB_FEATURE_C04, CB_FEATURE_C05, CB_FEATURE_C06, CB_FEATURE_C07,

  CB_FEATURE_C08, CB_FEATURE_C09, CB_FEATURE_C10, CB_FEATURE_C11,

  CB_FEATURE_C12 }
- enum cb_switch_name {

  CB_SWITCH_1, CB_SWITCH_2, CB_SWITCH_3, CB_SWITCH_4,

  CB_SWITCH_5, CB_SWITCH_6, CB_SWITCH_7, CB_SWITCH_8 }

- enum cb_class {

  CB_CLASS_UNKNOWN, CB_CLASS_ALPHABETIC, CB_CLASS_ALPHANUMERIC, CB_CLASS_BOOLEAN,

  CB_CLASS_INDEX, CB_CLASS_NATIONAL, CB_CLASS_NUMERIC, CB_CLASS_OBJECT,

  CB_CLASS_POINTER }
- enum cb_category {

  CB_CATEGORY_UNKNOWN, CB_CATEGORY_ALPHABETIC, CB_CATEGORY_ALPHANUMERIC, CB_CATEGORY_ALPHANUMERIC_EDITED,

  CB_CATEGORY_BOOLEAN, CB_CATEGORY_INDEX, CB_CATEGORY_NATIONAL, CB_CATEGORY_NATIONAL_EDITED,

  CB_CATEGORY_NUMERIC, CB_CATEGORY_NUMERIC_EDITED, CB_CATEGORY_OBJECT_REFERENCE, CB_CATEGORY_DATA_POINTER,

  CB_CATEGORY_PROGRAM_POINTER }
- enum cb_storage {

  CB_STORAGE_CONSTANT, CB_STORAGE_FILE, CB_STORAGE_WORKING, CB_STORAGE_LOCAL,

  CB_STORAGE_LINKAGE, CB_STORAGE_SCREEN, CB_STORAGE_REPORT, CB_STORAGE_COMMUNICATION }
- enum cb_usage {

  CB_USAGE_BINARY, CB_USAGE_BIT, CB_USAGE_COMP_5, CB_USAGE_COMP_X,

  CB_USAGE_DISPLAY, CB_USAGE_FLOAT, CB_USAGE_DOUBLE, CB_USAGE_INDEX,

  CB_USAGE_NATIONAL, CB_USAGE_OBJECT, CB_USAGE_PACKED, CB_USAGE_POINTER,

  CB_USAGE_PROGRAM, CB_USAGE_LENGTH, CB_USAGE_PROGRAM_POINTER, CB_USAGE_UNSIGNED_CHAR,

  CB_USAGE_SIGNED_CHAR, CB_USAGE_UNSIGNED_SHORT, CB_USAGE_SIGNED_SHORT, CB_USAGE_UNSIGNED_INT,

  CB_USAGE_SIGNED_INT, CB_USAGE_UNSIGNED_LONG, CB_USAGE_SIGNED_LONG }
- enum cb_operand_type { CB_SENDING_OPERAND, CB_RECEIVING_OPERAND }
- enum cb_cast_type {

  CB_CAST_INTEGER, CB_CAST_ADDRESS, CB_CAST_ADDR_OF_ADDR, CB_CAST_LENGTH,

  CB_CAST_PROGRAM_POINTER }
- enum cb_intr_enum {

  CB_INTR_ABS = 1, CB_INTR_ACOS, CB_INTR_ANNUITY, CB_INTR_ASIN,

  CB_INTR_ATAN, CB_INTR_BOOLEAN_OF_INTEGER, CB_INTR_BYTE_LENGTH, CB_INTR_CHAR,

CB_INTR_CHAR_NATIONAL, CB_INTR_COMBINED_DATETIME, CB_INTR_CONCATENATE, CB_INTR_COS,

CB_INTR_CURRENT_DATE, CB_INTR_DATE_OF_INTEGER, CB_INTR_DATE_-TO_YYYYMMDD, CB_INTR_DAY_OF_INTEGER,

CB_INTR_DAY_TO_YYYYDDD, CB_INTR_DISPLAY_OF, CB_INTR_E, CB_INTR_-EXCEPTION_FILE,

CB_INTR_EXCEPTION_FILE_N, CB_INTR_EXCEPTION_LOCATION, CB_INTR_-EXCEPTION_LOCATION_N, CB_INTR_EXCEPTION_STATEMENT,

CB_INTR_EXCEPTION_STATUS, CB_INTR_EXP, CB_INTR_EXP10, CB_INTR_-FACTORIAL,

CB_INTR_FRACTION_PART, CB_INTR_HIGHEST_ALGEBRAIC, CB_INTR_INTEGER, CB_INTR_INTEGER_OF_BOOLEAN,

CB_INTR_INTEGER_OF_DATE, CB_INTR_INTEGER_OF_DAY, CB_INTR_INTEGER_-PART, CB_INTR_LENGTH,

CB_INTR_LOCALE_COMPARE, CB_INTR_LOCALE_DATE, CB_INTR_LOCALE_-TIME, CB_INTR_LOCALE_TIME_FROM_SECS,

CB_INTR_LOG, CB_INTR_LOG10, CB_INTR_LOWER_CASE, CB_INTR_LOWEST_-ALGEBRAIC,

CB_INTR_MAX, CB_INTR_MEAN, CB_INTR_MEDIAN, CB_INTR_MIDRANGE,

CB_INTR_MIN, CB_INTR_MOD, CB_INTR_NATIONAL_OF, CB_INTR_NUMVAL,

CB_INTR_NUMVAL_C, CB_INTR_NUMVAL_F, CB_INTR_ORD, CB_INTR_ORD_-MAX,

CB_INTR_ORD_MIN, CB_INTR_PI, CB_INTR_PRESENT_VALUE, CB_INTR_-RANDOM,

CB_INTR_RANGE, CB_INTR_REM, CB_INTR_REVERSE, CB_INTR_SECONDS_-FROM_FORMATTED_TIME,

CB_INTR_SECONDS_PAST_MIDNIGHT, CB_INTR_SIGN, CB_INTR_SIN, CB_-INTR_SQRT,

CB_INTR_STANDARD_COMPARE, CB_INTR_STANDARD_DEVIATION, CB_INTR_-STORED_CHAR_LENGTH, CB_INTR_SUBSTITUTE,

CB_INTR_SUBSTITUTE_CASE, CB_INTR_SUM, CB_INTR_TAN, CB_INTR_-TEST_DATE_YYYYMMDD,

CB_INTR_TEST_DAY_YYYYDDD, CB_INTR_TEST_NUMVAL, CB_INTR_TEST_-NUMVAL_C, CB_INTR_TEST_NUMVAL_F,

CB_INTR_TRIM, CB_INTR_UPPER_CASE, CB_INTR_VARIANCE, CB_INTR_-WHEN_COMPILED,

CB_INTR_YEAR_TO_YYYY }

- enum cb_perform_type {

CB_PERFORM_EXIT, CB_PERFORM_ONCE, CB_PERFORM_TIMES, CB_PERFORM_-UNTIL,

CB_PERFORM_FOREVER }

Functions

- char * cb_name (cb_tree x)
- enum cb_class cb_tree_class (cb_tree x)
- enum cb_category cb_tree_category (cb_tree x)
- int cb_tree_type (cb_tree x)
- int cb_fits_int (cb_tree x)
- int cb_fits_long_long (cb_tree x)
- int cb_get_int (cb_tree x)
- long long cb_get_long_long (cb_tree x)
- void cb_init_constants (void)
- cb_tree cb_int (int n)
- cb_tree cb_build_string (const unsigned char *data, size_t size)
- cb_tree cb_build_alphabet_name (cb_tree name, enum cb_alphabet_name_type type)
- cb_tree cb_build_class_name (cb_tree name, cb_tree list)
- cb_tree cb_build_locale_name (cb_tree name, cb_tree list)
- cb_tree cb_build_system_name (enum cb_system_name_category category, int token)
- cb_tree cb_build_numeric_literal (int sign, const unsigned char *data, int scale)
- cb_tree cb_build_alphanumeric_literal (const unsigned char *data, size_t size)
- cb_tree cb_concat_literals (cb_tree x1, cb_tree x2)
- cb_tree cb_build_decimal (int id)
- cb_tree cb_build_picture (const char *str)
- cb_tree cb_build_field (cb_tree name)
- cb_tree cb_build_implicit_field (cb_tree name, int len)
- cb_tree cb_build_constant (cb_tree name, cb_tree value)
- struct cb_field * cb_field (cb_tree x)
- struct cb_field * cb_field_add (struct cb_field *f, struct cb_field *p)
- int cb_field_size (cb_tree x)
- struct cb_field * cb_field_founder (struct cb_field *f)
- struct cb_field * cb_field_variable_size (struct cb_field *f)
- struct cb_field * cb_field_variable_address (struct cb_field *f)
- int cb_field_subordinate (struct cb_field *p, struct cb_field *f)
- cb_tree cb_build_label (cb_tree name, struct cb_label *section)
- struct cb_file * build_file (cb_tree name)
- void validate_file (struct cb_file *f, cb_tree name)
- void finalize_file (struct cb_file *f, struct cb_field *records)
- cb_tree cb_build_filler (void)
- cb_tree cb_build_reference (const char *name)
- cb_tree cb_build_field_reference (struct cb_field *f, cb_tree ref)
- const char * cb_define (cb_tree name, cb_tree val)
- void cb_define_system_name (const char *name)
- cb_tree cb_ref (cb_tree x)
- cb_tree cb_build_binary_op (cb_tree x, int op, cb_tree y)
- cb_tree cb_build_binary_list (cb_tree l, int op)

- cb_tree cb_build_funcall (const char *name, int argc, cb_tree a1, cb_tree a2, cb_tree a3, cb_tree a4, cb_tree a5, cb_tree a6, cb_tree a7)
- cb_tree cb_build_cast (enum cb_cast_type type, cb_tree val)
- cb_tree cb_build_assign (cb_tree var, cb_tree val)
- struct cb_intrinsic_table * lookup_intrinsic (const char *name, const int checkres)
- cb_tree cb_build_intrinsic (cb_tree name, cb_tree args, cb_tree refmod)
- cb_tree cb_build_any_intrinsic (cb_tree args)
- cb_tree cb_build_initialize (cb_tree var, cb_tree val, cb_tree rep, cb_tree def, int flag)
- cb_tree cb_build_search (int flag_all, cb_tree table, cb_tree var, cb_tree end_-stmt, cb_tree whens)
- cb_tree cb_build_call (cb_tree name, cb_tree args, cb_tree stmt1, cb_tree stmt2, cb_tree returning, int is_system_call)
- cb_tree cb_build_goto (cb_tree target, cb_tree depending)
- cb_tree cb_build_if (cb_tree test, cb_tree stmt1, cb_tree stmt2)
- cb_tree cb_build_perform (int type)
- cb_tree cb_build_perform_varying (cb_tree name, cb_tree from, cb_tree step, cb_tree until)
- struct cb_statement * cb_build_statement (const char *name)
- cb_tree cb_build_continue (void)
- cb_tree cb_build_list (cb_tree purpose, cb_tree value, cb_tree rest)
- cb_tree cb_list_add (cb_tree l, cb_tree x)
- cb_tree cb_list_append (cb_tree l1, cb_tree l2)
- cb_tree cb_list_reverse (cb_tree l)
- int cb_list_length (cb_tree l)
- struct cb_program * cb_build_program (struct cb_program *last_program, int nest_level)
- cb_tree lookup_system_name (const char *name)
- int lookup_reserved_word (const char *name)
- void cb_list_reserved (void)
- void cb_list_intrinsics (void)
- void cb_list_mnemonics (void)
- void cb_init_reserved (void)
- void cb_list_map (cb_tree(*func)(cb_tree x), cb_tree l)
- void cb_warning_x (cb_tree x, const char *fmt,...)
- void cb_error_x (cb_tree x, const char *fmt,...)
- char * check_filler_name (char *name)
- void redefinition_error (cb_tree x)
- void redefinition_warning (cb_tree x, cb_tree y)
- void undefined_error (cb_tree x)
- void ambiguous_error (cb_tree x)
- void group_error (cb_tree x, const char *clause)
- void level_redundant_error (cb_tree x, const char *clause)
- void level_require_error (cb_tree x, const char *clause)
- void level_except_error (cb_tree x, const char *clause)
- struct cb_literal * build_literal (enum cb_category category, const unsigned char *data, size_t size)

- int cb_get_level (cb_tree x)
- cb_tree cb_build_field_tree (cb_tree level, cb_tree name, struct cb_field *last_field, enum cb_storage storage, struct cb_file *fn)
- struct cb_field * cb_resolve_redefines (struct cb_field *field, cb_tree redefines)
- void cb_validate_field (struct cb_field *p)
- void cb_validate_88_item (struct cb_field *p)
- struct cb_field * cb_validate_78_item (struct cb_field *p)
- void cb_clear_real_field (void)
- cb_tree cb_check_numeric_value (cb_tree x)
- void cb_build_registers (void)
- char * cb_encode_program_id (const char *name)
- const char * cb_build_program_id (cb_tree name, cb_tree alt_name)
- void cb_define_switch_name (cb_tree name, cb_tree sname, cb_tree flag, cb_tree ref)
- cb_tree cb_build_section_name (cb_tree name, int sect_or_para)
- cb_tree cb_build_assignment_name (struct cb_file *curfile, cb_tree name)
- cb_tree cb_build_index (cb_tree name, cb_tree values, int indexed_by, struct cb_field *qual)
- cb_tree cb_build_identifier (cb_tree x)
- cb_tree cb_build_length (cb_tree x)
- cb_tree cb_build_const_length (cb_tree x)
- cb_tree cb_build_address (cb_tree x)
- cb_tree cb_build_ppointer (cb_tree x)
- void cb_validate_program_environment (struct cb_program *prog)
- void cb_validate_program_data (struct cb_program *prog)
- void cb_validate_program_body (struct cb_program *prog)
- cb_tree cb_build_expr (cb_tree list)
- cb_tree cb_build_cond (cb_tree x)
- void cb_emit_arithmetic (cb_tree vars, int op, cb_tree val)
- cb_tree cb_build_add (cb_tree v, cb_tree n, cb_tree round_opt)
- cb_tree cb_build_sub (cb_tree v, cb_tree n, cb_tree round_opt)
- void cb_emit_corresponding (cb_tree(*func)(cb_tree f1, cb_tree f2, cb_tree f3), cb_tree x1, cb_tree x2, cb_tree opt)
- void cb_emit_move_corresponding (cb_tree x1, cb_tree x2)
- void cb_emit_accept (cb_tree var, cb_tree pos, cb_tree fgc, cb_tree bgc, cb_tree scroll, int dispattrs)
- void cb_emit_accept_line_or_col (cb_tree var, const int l_or_c)
- void cb_emit_accept_date (cb_tree var)
- void cb_emit_accept_date_yyyymmdd (cb_tree var)
- void cb_emit_accept_day (cb_tree var)
- void cb_emit_accept_day_yyyyddd (cb_tree var)
- void cb_emit_accept_day_of_week (cb_tree var)
- void cb_emit_accept_time (cb_tree var)
- void cb_emit_accept_command_line (cb_tree var)
- void cb_emit_get_environment (cb_tree envvar, cb_tree envval)
- void cb_emit_accept_environment (cb_tree var)
- void cb_emit_accept_mnemonic (cb_tree var, cb_tree mnemonic)

- void cb_emit_accept_name (cb_tree var, cb_tree name)
- void cb_emit_accept_arg_number (cb_tree var)
- void cb_emit_accept_arg_value (cb_tree var)
- void cb_emit_allocate (cb_tree target1, cb_tree target2, cb_tree size, cb_tree initialize)
- void cb_emit_free (cb_tree vars)
- void cb_emit_call (cb_tree prog, cb_tree using, cb_tree returning, cb_tree on_exception, cb_tree not_on_exception)
- void cb_emit_cancel (cb_tree prog)
- void cb_emit_close (cb_tree file, cb_tree opt)
- void cb_emit_commit (void)
- void cb_emit_continue (void)
- void cb_emit_delete (cb_tree file)
- void cb_emit_display (cb_tree values, cb_tree upon, cb_tree no_adv, cb_tree pos, cb_tree fgc, cb_tree bgc, cb_tree scroll, int dispattrs)
- cb_tree cb_build_display_upon (cb_tree x)
- cb_tree cb_build_display_upon_direct (cb_tree x)
- void cb_emit_env_name (cb_tree value)
- void cb_emit_env_value (cb_tree value)
- void cb_emit_arg_number (cb_tree value)
- void cb_emit_command_line (cb_tree value)
- void cb_emit_divide (cb_tree dividend, cb_tree divisor, cb_tree quotient, cb_tree remainder)
- void cb_emit_evaluate (cb_tree subject_list, cb_tree case_list)
- void cb_emit_goto (cb_tree target, cb_tree depending)
- void cb_emit_exit (size_t goback)
- void cb_emit_if (cb_tree cond, cb_tree stmt1, cb_tree stmt2)
- void cb_emit_initialize (cb_tree vars, cb_tree fillinit, cb_tree value, cb_tree replacing, cb_tree def)
- void cb_emit_inspect (cb_tree var, cb_tree body, cb_tree replacing, int replconv)
- void cb_init_tarrying (void)
- cb_tree cb_build_tarrying_data (cb_tree x)
- cb_tree cb_build_tarrying_characters (cb_tree l)
- cb_tree cb_build_tarrying_all (void)
- cb_tree cb_build_tarrying_leading (void)
- cb_tree cb_build_tarrying_trailing (void)
- cb_tree cb_build_tarrying_value (cb_tree x, cb_tree l)
- cb_tree cb_build_replacing_characters (cb_tree x, cb_tree l)
- cb_tree cb_build_replacing_all (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_replacing_leading (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_replacing_first (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_replacing_trailing (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_converting (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_inspect_region_start (void)
- cb_tree cb_build_inspect_region (cb_tree l, cb_tree pos, cb_tree x)
- int validate_move (cb_tree src, cb_tree dst, size_t is_value)
- cb_tree cb_build_move (cb_tree src, cb_tree dst)

- void cb_emit_move (cb_tree src, cb_tree dsts)
- void cb_emit_open (cb_tree file, cb_tree mode, cb_tree sharing)
- void cb_emit_perform (cb_tree perform, cb_tree body)
- cb_tree cb_build_perform_once (cb_tree body)
- cb_tree cb_build_perform_times (cb_tree count)
- cb_tree cb_build_perform_until (cb_tree condition, cb_tree varying)
- cb_tree cb_build_perform_forever (cb_tree body)
- cb_tree cb_build_perform_exit (struct cb_label *label)
- void cb_emit_read (cb_tree ref, cb_tree next, cb_tree into, cb_tree key, cb_tree lock_opts)
- void cb_emit_rewrite (cb_tree record, cb_tree from, cb_tree lockopt)
- void cb_emit_release (cb_tree ref, cb_tree from)
- void cb_emit_return (cb_tree ref, cb_tree into)
- void cb_emit_rollback (void)
- void cb_emit_search (cb_tree table, cb_tree varying, cb_tree at_end, cb_tree whens)
- void cb_emit_search_all (cb_tree table, cb_tree at_end, cb_tree when, cb_tree stmts)
- void cb_emit_setenv (cb_tree x, cb_tree y)
- void cb_emit_set_to (cb_tree l, cb_tree x)
- void cb_emit_set_up_down (cb_tree l, cb_tree flag, cb_tree x)
- void cb_emit_set_on_off (cb_tree l, cb_tree flag)
- void cb_emit_set_true (cb_tree l)
- void cb_emit_set_false (cb_tree l)
- void cb_emit_sort_init (cb_tree name, cb_tree keys, cb_tree col)
- void cb_emit_sort_using (cb_tree file, cb_tree l)
- void cb_emit_sort_input (cb_tree proc)
- void cb_emit_sort_giving (cb_tree file, cb_tree l)
- void cb_emit_sort_output (cb_tree proc)
- void cb_emit_sort_finish (cb_tree file)
- void cb_emit_start (cb_tree file, cb_tree op, cb_tree key)
- void cb_emit_stop_run (cb_tree x)
- void cb_emit_string (cb_tree items, cb_tree into, cb_tree pointer)
- void cb_emit_unlock (cb_tree ref)
- void cb_emit_unstring (cb_tree name, cb_tree delimited, cb_tree into, cb_tree pointer, cb_tree tallying)
- cb_tree cb_build_unstring_delimited (cb_tree all, cb_tree value)
- cb_tree cb_build_unstring_into (cb_tree name, cb_tree delimiter, cb_tree count)
- void cb_emit_write (cb_tree record, cb_tree from, cb_tree opt, cb_tree lockopt)
- cb_tree cb_build_write_advancing_lines (cb_tree pos, cb_tree lines)
- cb_tree cb_build_write_advancing_mnemonic (cb_tree pos, cb_tree mnemonic)
- cb_tree cb_build_write_advancing_page (cb_tree pos)
- void cobc_tree_cast_error (cb_tree x, const char *filen, const int linenum, const int tagnum)
- void codegen (struct cb_program *prog, int nested)
- void cb_set_in_procedure (void)
- void cb_reset_in_procedure (void)

- void cb_add_78 (struct cb_field *f)
- void cb_reset_78 (void)
- struct cb_field * check_level_78 (const char *name)

Variables

- cb_tree cb_any
- cb_tree cb_true
- cb_tree cb_false
- cb_tree cb_null
- cb_tree cb_zero
- cb_tree cb_one
- cb_tree cb_space
- cb_tree cb_low
- cb_tree cb_high
- cb_tree cb_norm_low
- cb_tree cb_norm_high
- cb_tree cb_quote
- cb_tree cb_int0
- cb_tree cb_int1
- cb_tree cb_int2
- cb_tree cb_int3
- cb_tree cb_int4
- cb_tree cb_int5
- cb_tree cb_i [8]
- cb_tree cb_error_node
- cb_tree cb_intr_whencomp
- cb_tree cb_intr_pi
- cb_tree cb_intr_e
- cb_tree cb_standard_error_handler
- size_t gen_screen_ptr
- int non_const_word
- size_t cb_needs_01

7.16.1 Define Documentation

7.16.1.1 #define CB_AFTER cb_int1

Definition at line 27 of file tree.h.

7.16.1.2 #define CB_ALPHABET_NAME(*x*) (CB_TREE_CAST (CB_TAG_ALPHABET_NAME, struct cb_alphabet_name, x))

Definition at line 338 of file tree.h.

7.16.1.3 #define CB_ALPHABET_NAME_P(x) (CB_TREE_TAG (x) == CB_TAG_ALPHABET_NAME)

Definition at line 339 of file tree.h.

7.16.1.4 #define CB_ASSIGN(x) (CB_TREE_CAST (CB_TAG_ASSIGN, struct cb_assign, x))

Definition at line 799 of file tree.h.

7.16.1.5 #define CB_ASSIGN_P(x) (CB_TREE_TAG (x) == CB_TAG_ASSIGN)

Definition at line 800 of file tree.h.

7.16.1.6 #define CB_BEFORE cb_int0

Definition at line 26 of file tree.h.

7.16.1.7 #define CB_BINARY_OP(x) (CB_TREE_CAST (CB_TAG_BINARY_OP, struct cb_binary_op, x))

Definition at line 711 of file tree.h.

7.16.1.8 #define CB_BINARY_OP_P(x) (CB_TREE_TAG (x) == CB_TAG_BINARY_OP)

Definition at line 712 of file tree.h.

7.16.1.9 #define cb_build_cast_addr_of_addr(x) cb_build_cast (CB_CAST_ADDR_OF_ADDR, x)

Definition at line 784 of file tree.h.

7.16.1.10 #define cb_build_cast_address(x) cb_build_cast (CB_CAST_ADDRESS, x)

Definition at line 783 of file tree.h.

7.16.1.11 #define cb_build_cast_integer(x) cb_build_cast (CB_CAST_INTEGER, x)

Definition at line 782 of file tree.h.

7.16.1.12 #define cb_build_cast_length(x) cb_build_cast (CB_CAST_LENGTH, x)

Definition at line 785 of file tree.h.

7.16.1.13 #define cb_build_cast_ppointer(*x*) cb_build_cast (CB_CAST_PROGRAM_POINTER, x)

Definition at line 786 of file tree.h.

7.16.1.14 #define cb_build_funcall_0(*f*) cb_build_funcall(f, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL)

Definition at line 741 of file tree.h.

7.16.1.15 #define cb_build_funcall_1(*f, a1*) cb_build_funcall(f, 1, a1, NULL, NULL, NULL, NULL, NULL, NULL)

Definition at line 743 of file tree.h.

7.16.1.16 #define cb_build_funcall_2(*f, a1, a2*) cb_build_funcall(f, 2, a1, a2, NULL, NULL, NULL, NULL, NULL)

Definition at line 745 of file tree.h.

7.16.1.17 #define cb_build_funcall_3(*f, a1, a2, a3*) cb_build_funcall(f, 3, a1, a2, a3, NULL, NULL, NULL, NULL)

Definition at line 747 of file tree.h.

7.16.1.18 #define cb_build_funcall_4(*f, a1, a2, a3, a4*) cb_build_funcall(f, 4, a1, a2, a3, a4, NULL, NULL, NULL)

Definition at line 749 of file tree.h.

7.16.1.19 #define cb_build_funcall_5(*f, a1, a2, a3, a4, a5*) cb_build_funcall(f, 5, a1, a2, a3, a4, a5, NULL, NULL)

Definition at line 751 of file tree.h.

7.16.1.20 #define cb_build_funcall_6(*f, a1, a2, a3, a4, a5, a6*) cb_build_funcall(f, 6, a1, a2, a3, a4, a5, a6, NULL)

Definition at line 753 of file tree.h.

7.16.1.21 #define cb_build_funcall_7(*f, a1, a2, a3, a4, a5, a6, a7*) cb_build_funcall(f, 7, a1, a2, a3, a4, a5, a6, a7)

Definition at line 755 of file tree.h.

7.16.1.22 #define cb_build_negation(x) cb_build_binary_op (x, '!', NULL)

Definition at line 715 of file tree.h.

7.16.1.23 #define cb_build_pair(x, y) cb_build_list (x, y, NULL)

Definition at line 1147 of file tree.h.

7.16.1.24 #define cb_build_parenthesis(x) cb_build_binary_op (x, '@', NULL)

Definition at line 714 of file tree.h.

7.16.1.25 #define cb_build_string0(str) cb_build_string (str, strlen ((char *)str))

Definition at line 319 of file tree.h.

7.16.1.26 #define CB_CALL(x) (CB_TREE_CAST (CB_TAG_CALL, struct cb_call, x))

Definition at line 984 of file tree.h.

7.16.1.27 #define CB_CALL_BY_CONTENT 2

Definition at line 971 of file tree.h.

7.16.1.28 #define CB_CALL_BY_REFERENCE 1

Definition at line 970 of file tree.h.

7.16.1.29 #define CB_CALL_BY_VALUE 3

Definition at line 972 of file tree.h.

7.16.1.30 #define CB_CALL_P(x) (CB_TREE_TAG (x) == CB_TAG_CALL)

Definition at line 985 of file tree.h.

7.16.1.31 #define CB_CAST(x) (CB_TREE_CAST (CB_TAG_CAST, struct cb_cast, x))

Definition at line 777 of file tree.h.

7.16.1.32 #define CB_CAST_P(x) (CB_TREE_TAG (x) == CB_TAG_CAST)

Definition at line 778 of file tree.h.

7.16.1.33 #define CB_CHAIN(x) (CB_LIST (x)->chain)

Definition at line 1117 of file tree.h.

7.16.1.34 #define CB_CLASS_NAME(x) (CB_TREE_CAST (CB_TAG_CLASS_NAME, struct cb_class_name, x))

Definition at line 355 of file tree.h.

7.16.1.35 #define CB_CLASS_NAME_P(x) (CB_TREE_TAG (x) == CB_TAG_CLASS_NAME)

Definition at line 356 of file tree.h.

7.16.1.36 #define cb_cons(x, l) cb_build_list (NULL, x, l)

Definition at line 1139 of file tree.h.

7.16.1.37 #define CB_CONST(x) (CB_TREE_CAST (CB_TAG_CONST, struct cb_const, x))

Definition at line 285 of file tree.h.

7.16.1.38 #define CB_CONST_P(x) (CB_TREE_TAG (x) == CB_TAG_CONST)

Definition at line 286 of file tree.h.

7.16.1.39 #define CB_CONTINUE(x) (CB_TREE_CAST (CB_TAG_CONTINUE, struct cb_continue, x))

Definition at line 1094 of file tree.h.

7.16.1.40 #define CB_CONTINUE_P(x) (CB_TREE_TAG (x) == CB_TAG_CONTINUE)

Definition at line 1095 of file tree.h.

7.16.1.41 #define CB_DECIMAL(x) (CB_TREE_CAST (CB_TAG_DECIMAL, struct cb_decimal, x))

Definition at line 426 of file tree.h.

7.16.1.42 #define CB_DECIMAL_P(x) (CB_TREE_TAG (x) == CB_TAG_DECIMAL)

Definition at line 427 of file tree.h.

7.16.1.43 #define CB_FIELD(x) (CB_TREE_CAST (CB_TAG_FIELD, struct cb_field, x))

Definition at line 534 of file tree.h.

7.16.1.44 #define CB_FIELD_P(x) (CB_TREE_TAG (x) == CB_TAG_FIELD)

Definition at line 535 of file tree.h.

7.16.1.45 #define CB_FILE(x) (CB_TREE_CAST (CB_TAG_FILE, struct cb_file, x))

Definition at line 634 of file tree.h.

7.16.1.46 #define CB_FILE_P(x) (CB_TREE_TAG (x) == CB_TAG_FILE)

Definition at line 635 of file tree.h.

7.16.1.47 #define CB_FUNCALL(x) (CB_TREE_CAST (CB_TAG_FUNCALL, struct cb_funcall, x))

Definition at line 734 of file tree.h.

7.16.1.48 #define CB_FUNCALL_P(x) (CB_TREE_TAG (x) == CB_TAG_FUNCALL)

Definition at line 735 of file tree.h.

7.16.1.49 #define CB_FUNCTION_TYPE 1

Definition at line 42 of file tree.h.

7.16.1.50 #define CB_GOTO(x) (CB_TREE_CAST (CB_TAG_GOTO, struct cb_goto, x))

Definition at line 1001 of file tree.h.

7.16.1.51 #define CB_GOTO_P(x) (CB_TREE_TAG (x) == CB_TAG_GOTO)

Definition at line 1002 of file tree.h.

7.16.1.52 #define CB_IF(x) (CB_TREE_CAST (CB_TAG_IF, struct cb_if, x))

Definition at line 1018 of file tree.h.

7.16.1.53 #define CB_IF_P(x) (CB_TREE_TAG (x) == CB_TAG_IF)

Definition at line 1019 of file tree.h.

7.16.1.54 #define CB_INDEX_P(x)

Value:

```
((CB_FIELD_P (x) || CB_REFERENCE_P (x)) \
   && cb_field (x)->usage == CB_USAGE_INDEX)
```

Definition at line 553 of file tree.h.

7.16.1.55 #define CB_INITIALIZE(x) (CB_TREE_CAST (CB_TAG_INITIALIZE, struct cb_initialize, x))

Definition at line 941 of file tree.h.

7.16.1.56 #define CB_INITIALIZE_P(x) (CB_TREE_TAG (x) == CB_TAG_INITIALIZE)

Definition at line 942 of file tree.h.

7.16.1.57 #define CB_INTEGER(x) (CB_TREE_CAST (CB_TAG_INTEGER, struct cb_integer, x))

Definition at line 300 of file tree.h.

7.16.1.58 #define CB_INTEGER_P(x) (CB_TREE_TAG (x) == CB_TAG_INTEGER)

Definition at line 301 of file tree.h.

7.16.1.59 #define CB_INTRINSIC(x) (CB_TREE_CAST (CB_TAG_INTRINSIC, struct cb_intrinsic, x))

Definition at line 917 of file tree.h.

7.16.1.60 #define CB_INTRINSIC_P(x) (CB_TREE_TAG (x) == CB_TAG_INTRINSIC)

Definition at line 918 of file tree.h.

7.16.1.61 #define CB_LABEL(x) (CB_TREE_CAST (CB_TAG_LABEL, struct cb_label, x))

Definition at line 578 of file tree.h.

7.16.1.62 #define CB_LABEL_P(x) (CB_TREE_TAG (x) == CB_TAG_LABEL)

Definition at line 579 of file tree.h.

7.16.1.63 #define CB_LIST(x) (CB_TREE_CAST (CB_TAG_LIST, struct cb_list, x))

Definition at line 1112 of file tree.h.

7.16.1.64 #define cb_list_init(x) cb_build_list (NULL, x, NULL)

Definition at line 1138 of file tree.h.

7.16.1.65 #define CB_LIST_P(x) (CB_TREE_TAG (x) == CB_TAG_LIST)

Definition at line 1113 of file tree.h.

7.16.1.66 #define CB_LITERAL(x) (CB_TREE_CAST (CB_TAG_LITERAL, struct cb_literal, x))

Definition at line 407 of file tree.h.

7.16.1.67 #define CB_LITERAL_P(x) (CB_TREE_TAG (x) == CB_TAG_LITERAL)

Definition at line 408 of file tree.h.

7.16.1.68 #define CB_LOCALE_NAME(x) (CB_TREE_CAST (CB_TAG_LOCALE_NAME, struct cb_locale_name, x))

Definition at line 372 of file tree.h.

7.16.1.69 #define CB_LOCALE_NAME_P(x) (CB_TREE_TAG (x) == CB_TAG_LOCALE_NAME)

Definition at line 373 of file tree.h.

7.16.1.70 #define CB_NAME(x) (CB_REFERENCE (x)->word->name)

Definition at line 672 of file tree.h.

7.16.1.71 #define CB_NUMERIC_LITERAL_P(x) (CB_LITERAL_P (x) && CB_TREE_CATEGORY (x) == CB_CATEGORY_NUMERIC)

Definition at line 409 of file tree.h.

7.16.1.72 #define CB_PAIR_P(x) (CB_LIST_P (x) && CB_PAIR_X (x))

Definition at line 1143 of file tree.h.

7.16.1.73 #define CB_PAIR_X(x) CB_PURPOSE (x)

Definition at line 1144 of file tree.h.

7.16.1.74 #define CB_PAIR_Y(x) CB_VALUE (x)

Definition at line 1145 of file tree.h.

7.16.1.75 #define CB_PERFORM(x) (CB_TREE_CAST (CB_TAG_PERFORM, struct cb_perform, x))

Definition at line 1057 of file tree.h.

7.16.1.76 #define CB_PERFORM_P(x) (CB_TREE_TAG (x) == CB_TAG_PERFORM)

Definition at line 1058 of file tree.h.

7.16.1.77 #define CB_PERFORM_VARYING(x) (CB_TREE_CAST (CB_TAG_PERFORM_VARYING, struct cb_perform_varying, x))

Definition at line 1055 of file tree.h.

7.16.1.78 #define CB_PICTURE(x) (CB_TREE_CAST (CB_TAG_PICTURE, struct cb_picture, x))

Definition at line 449 of file tree.h.

7.16.1.79 #define CB_PICTURE_P(x) (CB_TREE_TAG (x) == CB_TAG_PICTURE)

Definition at line 450 of file tree.h.

7.16.1.80 #define CB_PREFIX_ATTR "a_"

Definition at line 31 of file tree.h.

7.16.1.81 #define CB_PREFIX_BASE "b_"

Definition at line 32 of file tree.h.

7.16.1.82 #define CB_PREFIX_CONST "c_"

Definition at line 33 of file tree.h.

7.16.1.83 #define CB_PREFIX_DECIMAL "d_"

Definition at line 34 of file tree.h.

7.16.1.84 #define CB_PREFIX_FIELD "f_"

Definition at line 35 of file tree.h.

7.16.1.85 #define CB_PREFIX_FILE "h_"

Definition at line 36 of file tree.h.

7.16.1.86 #define CB_PREFIX_KEYS "k_"

Definition at line 37 of file tree.h.

7.16.1.87 #define CB_PREFIX_LABEL "l_"

Definition at line 38 of file tree.h.

7.16.1.88 #define CB_PREFIX_SEQUENCE "s_"

Definition at line 39 of file tree.h.

7.16.1.89 #define CB_PROGRAM_TYPE 0

Definition at line 41 of file tree.h.

7.16.1.90 #define CB_PURPOSE(*x*) (CB_LIST (x)->purpose)

Definition at line 1115 of file tree.h.

7.16.1.91 #define CB_PURPOSE_INT(x) (CB_INTEGER (CB_PURPOSE (x))->val)

Definition at line 1120 of file tree.h.

7.16.1.92 #define CB_REF_OR_FIELD_P(x) ((CB_FIELD_P (x) || CB_REFERENCE_P (x)))

Definition at line 549 of file tree.h.

7.16.1.93 #define CB_REFERENCE(x) (CB_TREE_CAST (CB_TAG_REFERENCE, struct cb_reference, x))

Definition at line 669 of file tree.h.

7.16.1.94 #define CB_REFERENCE_P(x) (CB_TREE_TAG (x) == CB_TAG_REFERENCE)

Definition at line 670 of file tree.h.

7.16.1.95 #define CB_SEARCH(x) (CB_TREE_CAST (CB_TAG_SEARCH, struct cb_search, x))

Definition at line 960 of file tree.h.

7.16.1.96 #define CB_SEARCH_P(x) (CB_TREE_TAG (x) == CB_TAG_SEARCH)

Definition at line 961 of file tree.h.

7.16.1.97 #define CB_SIZE_1 1

Definition at line 1123 of file tree.h.

7.16.1.98 #define CB_SIZE_2 2

Definition at line 1124 of file tree.h.

7.16.1.99 #define CB_SIZE_4 3

Definition at line 1125 of file tree.h.

7.16.1.100 #define CB_SIZE_8 4

Definition at line 1126 of file tree.h.

7.16.1.101 #define CB_SIZE_AUTO 0

Definition at line 1122 of file tree.h.

7.16.1.102 #define CB_SIZE_UNSIGNED 8

Definition at line 1127 of file tree.h.

7.16.1.103 #define CB_SIZES(*x*) (CB_LIST (x)->sizes)

Definition at line 1118 of file tree.h.

7.16.1.104 #define CB_SIZES_INT(*x*) ((CB_LIST (x)->sizes) & 0x07)

Definition at line 1129 of file tree.h.

7.16.1.105 #define CB_SIZES_INT_UNSIGNED(*x*) ((CB_LIST (x)->sizes) & CB_SIZE_UNSIGNED)

Definition at line 1130 of file tree.h.

7.16.1.106 #define CB_STATEMENT(*x*) (CB_TREE_CAST (CB_TAG_STATEMENT, struct cb_statement, x))

Definition at line 1080 of file tree.h.

7.16.1.107 #define CB_STATEMENT_P(*x*) (CB_TREE_TAG (x) == CB_TAG_STATEMENT)

Definition at line 1081 of file tree.h.

7.16.1.108 #define CB_STRING(*x*) (CB_TREE_CAST (CB_TAG_STRING, struct cb_string, x))

Definition at line 316 of file tree.h.

7.16.1.109 #define CB_STRING_P(*x*) (CB_TREE_TAG (x) == CB_TAG_STRING)

Definition at line 317 of file tree.h.

7.16.1.110 #define CB_SYSTEM_NAME(*x*) (CB_TREE_CAST (CB_TAG_SYSTEM_NAME, struct cb_system_name, x))

Definition at line 387 of file tree.h.

7.16.1.111 #define CB_SYSTEM_NAME_P(x) (CB_TREE_TAG (x) == CB_TAG_SYSTEM_NAME)

Definition at line 388 of file tree.h.

7.16.1.112 #define CB_TREE(x) ((struct cb_tree_common *) (x))

Definition at line 219 of file tree.h.

7.16.1.113 #define CB_TREE_CAST(tg, ty, x) ((ty *) (x))

Definition at line 235 of file tree.h.

7.16.1.114 #define CB_TREE_CATEGORY(x) cb_tree_category (CB_TREE (x))

Definition at line 222 of file tree.h.

7.16.1.115 #define CB_TREE_CLASS(x) cb_tree_class (CB_TREE (x))

Definition at line 221 of file tree.h.

7.16.1.116 #define CB_TREE_TAG(x) (CB_TREE (x)->tag)

Definition at line 220 of file tree.h.

7.16.1.117 #define CB_VALUE(x) (CB_LIST (x)->value)

Definition at line 1116 of file tree.h.

7.16.1.118 #define CB_WORD_HASH_SIZE 133

Definition at line 646 of file tree.h.

7.16.1.119 #define COB_MAX_SUBSCRIPTS 16

Definition at line 29 of file tree.h.

7.16.1.120 #define YYSTYPE cb_tree

Definition at line 24 of file tree.h.

7.16.2 Typedef Documentation

7.16.2.1 typedef struct cb_tree_common* cb_tree

Definition at line 217 of file tree.h.

7.16.3 Enumeration Type Documentation

7.16.3.1 enum cb_alphabet_name_type

Enumerator:

CB_ALPHABET_NATIVE

CB_ALPHABET_STANDARD_1

CB_ALPHABET_STANDARD_2

CB_ALPHABET_EBCDIC

CB_ALPHABET_CUSTOM

Definition at line 80 of file tree.h.

```
                        {
        CB_ALPHABET_NATIVE,
        CB_ALPHABET_STANDARD_1,
        CB_ALPHABET_STANDARD_2,
        CB_ALPHABET_EBCDIC,
        CB_ALPHABET_CUSTOM
};
```

7.16.3.2 enum cb_cast_type

Enumerator:

CB_CAST_INTEGER

CB_CAST_ADDRESS

CB_CAST_ADDR_OF_ADDR

CB_CAST_LENGTH

CB_CAST_PROGRAM_POINTER

Definition at line 763 of file tree.h.

```
                {
        CB_CAST_INTEGER,
        CB_CAST_ADDRESS,
        CB_CAST_ADDR_OF_ADDR,
        CB_CAST_LENGTH,
        CB_CAST_PROGRAM_POINTER
};
```

7.16.3.3 enum cb_category

Enumerator:

CB_CATEGORY_UNKNOWN

CB_CATEGORY_ALPHABETIC

CB_CATEGORY_ALPHANUMERIC

CB_CATEGORY_ALPHANUMERIC_EDITED

CB_CATEGORY_BOOLEAN

CB_CATEGORY_INDEX

CB_CATEGORY_NATIONAL

CB_CATEGORY_NATIONAL_EDITED

CB_CATEGORY_NUMERIC

CB_CATEGORY_NUMERIC_EDITED

CB_CATEGORY_OBJECT_REFERENCE

CB_CATEGORY_DATA_POINTER

CB_CATEGORY_PROGRAM_POINTER

Definition at line 147 of file tree.h.

```
                {
        CB_CATEGORY_UNKNOWN,                    /* 0 */
        CB_CATEGORY_ALPHABETIC,                 /* 1 */
        CB_CATEGORY_ALPHANUMERIC,               /* 2 */
        CB_CATEGORY_ALPHANUMERIC_EDITED,        /* 3 */
        CB_CATEGORY_BOOLEAN,                    /* 4 */
        CB_CATEGORY_INDEX,                      /* 5 */
        CB_CATEGORY_NATIONAL,                   /* 6 */
        CB_CATEGORY_NATIONAL_EDITED,            /* 7 */
        CB_CATEGORY_NUMERIC,                    /* 8 */
        CB_CATEGORY_NUMERIC_EDITED,             /* 9 */
        CB_CATEGORY_OBJECT_REFERENCE,           /* 10 */
        CB_CATEGORY_DATA_POINTER,               /* 11 */
        CB_CATEGORY_PROGRAM_POINTER             /* 12 */
};
```

7.16.3.4 enum cb_class

Enumerator:

CB_CLASS_UNKNOWN

CB_CLASS_ALPHABETIC

CB_CLASS_ALPHANUMERIC

CB_CLASS_BOOLEAN

CB_CLASS_INDEX

CB_CLASS_NATIONAL

CB_CLASS_NUMERIC

CB_CLASS_OBJECT

CB_CLASS_POINTER

Definition at line 135 of file tree.h.

```
        {
        CB_CLASS_UNKNOWN,               /* 0 */
        CB_CLASS_ALPHABETIC,            /* 1 */
        CB_CLASS_ALPHANUMERIC,          /* 2 */
        CB_CLASS_BOOLEAN,               /* 3 */
        CB_CLASS_INDEX,                 /* 4 */
        CB_CLASS_NATIONAL,              /* 5 */
        CB_CLASS_NUMERIC,               /* 6 */
        CB_CLASS_OBJECT,                /* 7 */
        CB_CLASS_POINTER                /* 8 */
};
```

7.16.3.5 enum cb_device_name

Enumerator:

CB_DEVICE_SYSIN

CB_DEVICE_SYSOUT

CB_DEVICE_SYSERR

CB_DEVICE_CONSOLE

Definition at line 101 of file tree.h.

```
                {
        CB_DEVICE_SYSIN,
        CB_DEVICE_SYSOUT,
        CB_DEVICE_SYSERR,
        CB_DEVICE_CONSOLE
};
```

7.16.3.6 enum cb_feature_name

Enumerator:

CB_FEATURE_FORMFEED

CB_FEATURE_C01

CB_FEATURE_C02

CB_FEATURE_C03

CB_FEATURE_C04

CB_FEATURE_C05

CB_FEATURE_C06

CB_FEATURE_C07

CB_FEATURE_C08

CB_FEATURE_C09

CB_FEATURE_C10

CB_FEATURE_C11

CB_FEATURE_C12

Definition at line 108 of file tree.h.

```
                {
        CB_FEATURE_FORMFEED,
        CB_FEATURE_C01,
        CB_FEATURE_C02,
        CB_FEATURE_C03,
        CB_FEATURE_C04,
        CB_FEATURE_C05,
        CB_FEATURE_C06,
        CB_FEATURE_C07,
        CB_FEATURE_C08,
        CB_FEATURE_C09,
        CB_FEATURE_C10,
        CB_FEATURE_C11,
        CB_FEATURE_C12
};
```

7.16.3.7 enum cb_intr_enum

Enumerator:

CB_INTR_ABS

CB_INTR_ACOS

CB_INTR_ANNUITY

CB_INTR_ASIN

CB_INTR_ATAN

CB_INTR_BOOLEAN_OF_INTEGER

CB_INTR_BYTE_LENGTH

CB_INTR_CHAR

CB_INTR_CHAR_NATIONAL

CB_INTR_COMBINED_DATETIME

CB_INTR_CONCATENATE

CB_INTR_COS

CB_INTR_CURRENT_DATE

CB_INTR_DATE_OF_INTEGER

CB_INTR_DATE_TO_YYYYMMDD

CB_INTR_DAY_OF_INTEGER

CB_INTR_DAY_TO_YYYYDDD

CB_INTR_DISPLAY_OF

CB_INTR_E

CB_INTR_EXCEPTION_FILE

CB_INTR_EXCEPTION_FILE_N

CB_INTR_EXCEPTION_LOCATION

CB_INTR_EXCEPTION_LOCATION_N

CB_INTR_EXCEPTION_STATEMENT

CB_INTR_EXCEPTION_STATUS

CB_INTR_EXP

CB_INTR_EXP10

CB_INTR_FACTORIAL

CB_INTR_FRACTION_PART

CB_INTR_HIGHEST_ALGEBRAIC

CB_INTR_INTEGER

CB_INTR_INTEGER_OF_BOOLEAN

CB_INTR_INTEGER_OF_DATE

CB_INTR_INTEGER_OF_DAY

CB_INTR_INTEGER_PART

CB_INTR_LENGTH

CB_INTR_LOCALE_COMPARE

CB_INTR_LOCALE_DATE

CB_INTR_LOCALE_TIME

CB_INTR_LOCALE_TIME_FROM_SECS

CB_INTR_LOG

CB_INTR_LOG10

CB_INTR_LOWER_CASE

CB_INTR_LOWEST_ALGEBRAIC

CB_INTR_MAX

CB_INTR_MEAN

CB_INTR_MEDIAN

CB_INTR_MIDRANGE

CB_INTR_MIN

CB_INTR_MOD

CB_INTR_NATIONAL_OF

CB_INTR_NUMVAL

CB_INTR_NUMVAL_C

CB_INTR_NUMVAL_F

CB_INTR_ORD

CB_INTR_ORD_MAX

CB_INTR_ORD_MIN

CB_INTR_PI

CB_INTR_PRESENT_VALUE

CB_INTR_RANDOM

CB_INTR_RANGE

CB_INTR_REM

CB_INTR_REVERSE

CB_INTR_SECONDS_FROM_FORMATTED_TIME

CB_INTR_SECONDS_PAST_MIDNIGHT

CB_INTR_SIGN

CB_INTR_SIN

CB_INTR_SQRT

CB_INTR_STANDARD_COMPARE

CB_INTR_STANDARD_DEVIATION

CB_INTR_STORED_CHAR_LENGTH

CB_INTR_SUBSTITUTE

CB_INTR_SUBSTITUTE_CASE

CB_INTR_SUM

CB_INTR_TAN

CB_INTR_TEST_DATE_YYYYMMDD

CB_INTR_TEST_DAY_YYYYDDD

CB_INTR_TEST_NUMVAL

CB_INTR_TEST_NUMVAL_C

CB_INTR_TEST_NUMVAL_F

CB_INTR_TRIM

CB_INTR_UPPER_CASE

CB_INTR_VARIANCE

CB_INTR_WHEN_COMPILED

CB_INTR_YEAR_TO_YYYY

Definition at line 809 of file tree.h.

```
          {
CB_INTR_ABS = 1,
CB_INTR_ACOS,
CB_INTR_ANNUITY,
CB_INTR_ASIN,
CB_INTR_ATAN,
CB_INTR_BOOLEAN_OF_INTEGER,
CB_INTR_BYTE_LENGTH,
CB_INTR_CHAR,
```

```
CB_INTR_CHAR_NATIONAL,
CB_INTR_COMBINED_DATETIME,
CB_INTR_CONCATENATE,
CB_INTR_COS,
CB_INTR_CURRENT_DATE,
CB_INTR_DATE_OF_INTEGER,
CB_INTR_DATE_TO_YYYYMMDD,
CB_INTR_DAY_OF_INTEGER,
CB_INTR_DAY_TO_YYYYDDD,
CB_INTR_DISPLAY_OF,
CB_INTR_E,
CB_INTR_EXCEPTION_FILE,
CB_INTR_EXCEPTION_FILE_N,
CB_INTR_EXCEPTION_LOCATION,
CB_INTR_EXCEPTION_LOCATION_N,
CB_INTR_EXCEPTION_STATEMENT,
CB_INTR_EXCEPTION_STATUS,
CB_INTR_EXP,
CB_INTR_EXP10,
CB_INTR_FACTORIAL,
CB_INTR_FRACTION_PART,
CB_INTR_HIGHEST_ALGEBRAIC,
CB_INTR_INTEGER,
CB_INTR_INTEGER_OF_BOOLEAN,
CB_INTR_INTEGER_OF_DATE,
CB_INTR_INTEGER_OF_DAY,
CB_INTR_INTEGER_PART,
CB_INTR_LENGTH,
CB_INTR_LOCALE_COMPARE,
CB_INTR_LOCALE_DATE,
CB_INTR_LOCALE_TIME,
CB_INTR_LOCALE_TIME_FROM_SECS,
CB_INTR_LOG,
CB_INTR_LOG10,
CB_INTR_LOWER_CASE,
CB_INTR_LOWEST_ALGEBRAIC,
CB_INTR_MAX,
CB_INTR_MEAN,
CB_INTR_MEDIAN,
CB_INTR_MIDRANGE,
CB_INTR_MIN,
CB_INTR_MOD,
CB_INTR_NATIONAL_OF,
CB_INTR_NUMVAL,
CB_INTR_NUMVAL_C,
CB_INTR_NUMVAL_F,
CB_INTR_ORD,
CB_INTR_ORD_MAX,
CB_INTR_ORD_MIN,
CB_INTR_PI,
CB_INTR_PRESENT_VALUE,
CB_INTR_RANDOM,
CB_INTR_RANGE,
CB_INTR_REM,
CB_INTR_REVERSE,
CB_INTR_SECONDS_FROM_FORMATTED_TIME,
CB_INTR_SECONDS_PAST_MIDNIGHT,
CB_INTR_SIGN,
CB_INTR_SIN,
CB_INTR_SQRT,
CB_INTR_STANDARD_COMPARE,
CB_INTR_STANDARD_DEVIATION,
```

```
        CB_INTR_STORED_CHAR_LENGTH,
        CB_INTR_SUBSTITUTE,
        CB_INTR_SUBSTITUTE_CASE,
        CB_INTR_SUM,
        CB_INTR_TAN,
        CB_INTR_TEST_DATE_YYYYMMDD,
        CB_INTR_TEST_DAY_YYYYDDD,
        CB_INTR_TEST_NUMVAL,
        CB_INTR_TEST_NUMVAL_C,
        CB_INTR_TEST_NUMVAL_F,
        CB_INTR_TRIM,
        CB_INTR_UPPER_CASE,
        CB_INTR_VARIANCE,
        CB_INTR_WHEN_COMPILED,
        CB_INTR_YEAR_TO_YYYY
};
```

7.16.3.8 enum cb_operand_type

Enumerator:

CB_SENDING_OPERAND

CB_RECEIVING_OPERAND

Definition at line 200 of file tree.h.

```
                {
        CB_SENDING_OPERAND,
        CB_RECEIVING_OPERAND
};
```

7.16.3.9 enum cb_perform_type

Enumerator:

CB_PERFORM_EXIT

CB_PERFORM_ONCE

CB_PERFORM_TIMES

CB_PERFORM_UNTIL

CB_PERFORM_FOREVER

Definition at line 1028 of file tree.h.

```
                {
        CB_PERFORM_EXIT,
        CB_PERFORM_ONCE,
        CB_PERFORM_TIMES,
        CB_PERFORM_UNTIL,
        CB_PERFORM_FOREVER
};
```

7.16.3.10 enum cb_storage

Enumerator:

CB_STORAGE_CONSTANT

CB_STORAGE_FILE

CB_STORAGE_WORKING

CB_STORAGE_LOCAL

CB_STORAGE_LINKAGE

CB_STORAGE_SCREEN

CB_STORAGE_REPORT

CB_STORAGE_COMMUNICATION

Definition at line 163 of file tree.h.

```
{
        CB_STORAGE_CONSTANT,            /* Constants */
        CB_STORAGE_FILE,                /* FILE SECTION */
        CB_STORAGE_WORKING,             /* WORKING-STORAGE SECTION */
        CB_STORAGE_LOCAL,               /* LOCAL-STORAGE SECTION */
        CB_STORAGE_LINKAGE,             /* LINKAGE SECTION */
        CB_STORAGE_SCREEN,              /* SCREEN SECTION */
        CB_STORAGE_REPORT,              /* REPORT SECTION */
        CB_STORAGE_COMMUNICATION        /* COMMUNICATION SECTION */
};
```

7.16.3.11 enum cb_switch_name

Enumerator:

CB_SWITCH_1

CB_SWITCH_2

CB_SWITCH_3

CB_SWITCH_4

CB_SWITCH_5

CB_SWITCH_6

CB_SWITCH_7

CB_SWITCH_8

Definition at line 124 of file tree.h.

```
{
        CB_SWITCH_1,
        CB_SWITCH_2,
        CB_SWITCH_3,
        CB_SWITCH_4,
        CB_SWITCH_5,
        CB_SWITCH_6,
        CB_SWITCH_7,
        CB_SWITCH_8
};
```

7.16.3.12 enum cb_system_name_category

Enumerator:

CB_CALL_CONVENTION_NAME

CB_CODE_NAME

CB_COMPUTER_NAME

CB_DEVICE_NAME

CB_ENTRY_CONVENTION_NAME

CB_EXTERNAL_LOCALE_NAME

CB_FEATURE_NAME

CB_LIBRARY_NAME

CB_SWITCH_NAME

CB_TEXT_NAME

Definition at line 88 of file tree.h.

```
                        {
        CB_CALL_CONVENTION_NAME,
        CB_CODE_NAME,
        CB_COMPUTER_NAME,
        CB_DEVICE_NAME,
        CB_ENTRY_CONVENTION_NAME,
        CB_EXTERNAL_LOCALE_NAME,
        CB_FEATURE_NAME,
        CB_LIBRARY_NAME,
        CB_SWITCH_NAME,
        CB_TEXT_NAME
};
```

7.16.3.13 enum cb_tag

Enumerator:

CB_TAG_CONST

CB_TAG_INTEGER

CB_TAG_STRING

CB_TAG_ALPHABET_NAME

CB_TAG_CLASS_NAME

CB_TAG_LOCALE_NAME

CB_TAG_SYSTEM_NAME

CB_TAG_LITERAL

CB_TAG_DECIMAL

CB_TAG_FIELD

CB_TAG_FILE

CB_TAG_REFERENCE

CB_TAG_BINARY_OP

CB_TAG_FUNCALL

CB_TAG_CAST

CB_TAG_INTRINSIC

CB_TAG_LABEL

CB_TAG_ASSIGN

CB_TAG_INITIALIZE

CB_TAG_SEARCH

CB_TAG_CALL

CB_TAG_GOTO

CB_TAG_IF

CB_TAG_PERFORM

CB_TAG_STATEMENT

CB_TAG_CONTINUE

CB_TAG_PERFORM_VARYING

CB_TAG_PICTURE

CB_TAG_LIST

Definition at line 44 of file tree.h.

```
    {
/* primitives */
CB_TAG_CONST,           /* 0 constant value */
CB_TAG_INTEGER,         /* 1 integer constant */
CB_TAG_STRING,          /* 2 string constant */
CB_TAG_ALPHABET_NAME,   /* 3 alphabet-name */
CB_TAG_CLASS_NAME,      /* 4 class-name */
CB_TAG_LOCALE_NAME,     /* 5 locale-name */
CB_TAG_SYSTEM_NAME,     /* 6 system-name */
CB_TAG_LITERAL,         /* 7 numeric/alphanumeric literal */
CB_TAG_DECIMAL,         /* 8 decimal number */
CB_TAG_FIELD,           /* 9 user-defined variable */
CB_TAG_FILE,            /* 10 file description */
/* expressions */
CB_TAG_REFERENCE,       /* 11 reference to a field, file, or label */
CB_TAG_BINARY_OP,       /* 12 binary operation */
CB_TAG_FUNCALL,         /* 13 run-time function call */
CB_TAG_CAST,            /* 14 type cast */
CB_TAG_INTRINSIC,       /* 15 intrinsic function */
/* statements */
CB_TAG_LABEL,           /* 16 label statement */
CB_TAG_ASSIGN,          /* 17 assignment statement */
CB_TAG_INITIALIZE,      /* 18 INITIALIZE statement */
CB_TAG_SEARCH,          /* 19 SEARCH statement */
CB_TAG_CALL,            /* 20 CALL statement */
CB_TAG_GOTO,            /* 21 GO TO statement */
CB_TAG_IF,              /* 22 IF statement */
CB_TAG_PERFORM,         /* 23 PERFORM statement */
CB_TAG_STATEMENT,       /* 24 general statement */
```

```
        CB_TAG_CONTINUE,        /* 25 CONTINUE statement */
        /* miscellaneous */
        CB_TAG_PERFORM_VARYING, /* 26 PERFORM VARYING parameter */
        CB_TAG_PICTURE,         /* 27 PICTURE clause */
        CB_TAG_LIST             /* 28 list */
};
```

7.16.3.14 enum cb_usage

Enumerator:

CB_USAGE_BINARY

CB_USAGE_BIT

CB_USAGE_COMP_5

CB_USAGE_COMP_X

CB_USAGE_DISPLAY

CB_USAGE_FLOAT

CB_USAGE_DOUBLE

CB_USAGE_INDEX

CB_USAGE_NATIONAL

CB_USAGE_OBJECT

CB_USAGE_PACKED

CB_USAGE_POINTER

CB_USAGE_PROGRAM

CB_USAGE_LENGTH

CB_USAGE_PROGRAM_POINTER

CB_USAGE_UNSIGNED_CHAR

CB_USAGE_SIGNED_CHAR

CB_USAGE_UNSIGNED_SHORT

CB_USAGE_SIGNED_SHORT

CB_USAGE_UNSIGNED_INT

CB_USAGE_SIGNED_INT

CB_USAGE_UNSIGNED_LONG

CB_USAGE_SIGNED_LONG

Definition at line 174 of file tree.h.

```
        {
        CB_USAGE_BINARY,                /* 0 */
        CB_USAGE_BIT,                   /* 1 */
        CB_USAGE_COMP_5,                /* 2 */
        CB_USAGE_COMP_X,                /* 3 */
        CB_USAGE_DISPLAY,               /* 4 */
        CB_USAGE_FLOAT,                 /* 5 */
```

```
        CB_USAGE_DOUBLE,                /* 6 */
        CB_USAGE_INDEX,                 /* 7 */
        CB_USAGE_NATIONAL,              /* 8 */
        CB_USAGE_OBJECT,                /* 9 */
        CB_USAGE_PACKED,                /* 10 */
        CB_USAGE_POINTER,               /* 11 */
        CB_USAGE_PROGRAM,               /* 12 */
        CB_USAGE_LENGTH,                /* 13 */
        CB_USAGE_PROGRAM_POINTER,       /* 14 */
        CB_USAGE_UNSIGNED_CHAR,         /* 15 */
        CB_USAGE_SIGNED_CHAR,           /* 16 */
        CB_USAGE_UNSIGNED_SHORT,        /* 17 */
        CB_USAGE_SIGNED_SHORT,          /* 18 */
        CB_USAGE_UNSIGNED_INT,          /* 19 */
        CB_USAGE_SIGNED_INT,            /* 20 */
        CB_USAGE_UNSIGNED_LONG,         /* 21 */
        CB_USAGE_SIGNED_LONG            /* 22 */
};
```

7.16.4 Function Documentation

7.16.4.1 void ambiguous_error (cb_tree *x*)

Definition at line 197 of file error.c.

```
{
        struct cb_word  *w;
        struct cb_field *p;
        struct cb_label *l2;
        cb_tree         l;
        cb_tree         y;

        w = CB_REFERENCE (x)->word;
        if (w->error == 0) {
                if (!errnamebuff) {
                        errnamebuff = cobc_malloc (COB_NORMAL_BUFF);
                }
                /* display error on the first time */
                snprintf (errnamebuff, COB_NORMAL_MAX, "'%s'", CB_NAME (x));
                for (l = CB_REFERENCE (x)->chain; l; l = CB_REFERENCE (l)->chain)
       {
                        strcat (errnamebuff, " in '");
                        strcat (errnamebuff, CB_NAME (l));
                        strcat (errnamebuff, "'");
                }
                cb_error_x (x, _("%s ambiguous; need qualification"), errnamebuff
      );
                w->error = 1;

                /* display all fields with the same name */
                for (l = w->items; l; l = CB_CHAIN (l)) {
                        y = CB_VALUE (l);
                        snprintf (errnamebuff, COB_NORMAL_MAX, "'%s' ", w->name);

                        switch (CB_TREE_TAG (y)) {
                        case CB_TAG_FIELD:
                                for (p = CB_FIELD (y)->parent; p; p = p->parent)
      {
                                        strcat (errnamebuff, "in '");
```

```
                                        strcat (errnamebuff, p->name);
                                        strcat (errnamebuff, "' ");
                                }
                                break;
                        case CB_TAG_LABEL:
                                l2 = CB_LABEL (y);
                                if (l2->section) {
                                        strcat (errnamebuff, "in '");
                                        strcat (errnamebuff, (const char *)(l2->
      section->name));
                                        strcat (errnamebuff, "' ");
                                }
                                break;
                        default:
                                break;
                        }
                        strcat (errnamebuff, _("defined here"));
                        cb_error_x (y, errnamebuff);
                }
        }
}
```

7.16.4.2 struct cb_file* build_file (cb_tree *name*) [read]

Definition at line 1577 of file tree.c.

```
{
        struct cb_file *p;

        p = make_tree (CB_TAG_FILE, CB_CATEGORY_UNKNOWN, sizeof (struct cb_file))
      ;
        p->name = cb_define (name, CB_TREE (p));
        p->cname = to_cname (p->name);

        p->organization = COB_ORG_SEQUENTIAL;
        p->access_mode = COB_ACCESS_SEQUENTIAL;
        p->handler = CB_LABEL (cb_standard_error_handler);
        p->handler_prog = current_program;
        return p;
}
```

7.16.4.3 struct cb_literal* build_literal (enum cb_category *category,* const unsigned char * *data,* size_t *size*) [read]

Definition at line 426 of file tree.c.

```
{
        struct cb_literal *p;

        p = make_tree (CB_TAG_LITERAL, category, sizeof (struct cb_literal));
        p->data = cobc_malloc ((size_t) (size + 1));
        p->size = size;
        memcpy (p->data, data, (size_t) size);
        /* RXW - malloc zeroes
        p->data[size] = 0;
```

```
        */
        return p;
}
```

7.16.4.4 void cb_add_78 (struct cb_field * f)

7.16.4.5 cb_tree cb_build_add (cb_tree v, cb_tree n, cb_tree round_opt)

Definition at line 2591 of file typeck.c.

```
{
        cb_tree         opt;
        struct cb_field *f;

#ifdef  COB_NON_ALIGNED
        if (CB_INDEX_P (v)) {
                return cb_build_move (cb_build_binary_op (v, '+', n), v);
        }
        if (CB_TREE_CLASS (v) == CB_CLASS_POINTER) {
                current_program->gen_ptrmanip = 1;
                return cb_build_funcall_3 ("cob_pointer_manip", v, n, cb_int0);
        }
#else
        if (CB_INDEX_P (v) || CB_TREE_CLASS (v) == CB_CLASS_POINTER) {
                return cb_build_move (cb_build_binary_op (v, '+', n), v);
        }
#endif

        if (CB_REF_OR_FIELD_P (v)) {
                f = cb_field (v);
                f->count++;
        }
        if (CB_REF_OR_FIELD_P (n)) {
                f = cb_field (n);
                f->count++;
        }
        if (round_opt == cb_high) {
                if (cb_fits_int (n)) {
                        return cb_build_optim_add (v, n);
                } else {
                        return cb_build_funcall_3 ("cob_add", v, n, cb_int0);
                }
        }
        opt = build_store_option (v, round_opt);
        if (opt == cb_int0 && cb_fits_int (n)) {
                return cb_build_optim_add (v, n);
        }
        return cb_build_funcall_3 ("cob_add", v, n, opt);
}
```

7.16.4.6 cb_tree cb_build_address (cb_tree x)

Definition at line 1027 of file typeck.c.

```
{
```

```
        if (x == cb_error_node ||
            (CB_REFERENCE_P (x) && cb_ref (x) == cb_error_node)) {
                return cb_error_node;
        }

        return cb_build_cast_address (x);
}
```

7.16.4.7 cb_tree cb_build_alphabet_name (cb_tree *name*, enum cb_alphabet_name_type *type*)

Definition at line 923 of file tree.c.

```
{
        struct cb_alphabet_name *p;

        p = make_tree (CB_TAG_ALPHABET_NAME, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_alphabet_name));
        p->name = cb_define (name, CB_TREE (p));
        p->cname = to_cname (p->name);
        p->type = type;
        return CB_TREE (p);
}
```

7.16.4.8 cb_tree cb_build_alphanumeric_literal (const unsigned char * *data*, size_t *size*)

Definition at line 999 of file tree.c.

```
{
        return CB_TREE (build_literal (CB_CATEGORY_ALPHANUMERIC, data, size));
}
```

7.16.4.9 cb_tree cb_build_any_intrinsic (cb_tree *args*)

Definition at line 2253 of file tree.c.

```
{
        struct cb_intrinsic_table       *cbp;

        cbp = lookup_intrinsic ("LENGTH", 0);
        return make_intrinsic (NULL, cbp, args, NULL, NULL);
}
```

7.16.4.10 cb_tree cb_build_assign (cb_tree *var*, cb_tree *val*)

Definition at line 2098 of file tree.c.

```
{
        struct cb_assign *p;

        p = make_tree (CB_TAG_ASSIGN, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_assign));
        p->var = var;
        p->val = val;
        return CB_TREE (p);
}
```

7.16.4.11 cb_tree cb_build_assignment_name (struct cb_file * *curfile,* cb_tree *name*)

Definition at line 677 of file typeck.c.

```
{
        const char      *s;
        const char      *p;

        if (name == cb_error_node) {
                return cb_error_node;
        }

        switch (CB_TREE_TAG (name)) {
        case CB_TAG_LITERAL:
                if (strcmp ((char *)(CB_LITERAL(name)->data), "$#@DUMMY@#$") == 0
      ) {
                        cfile->special = 2;
                }
                return name;

        case CB_TAG_REFERENCE:
                s = CB_REFERENCE (name)->word->name;
                if (strcasecmp (s, "KEYBOARD") == 0) {
                        s = "#DUMMY#";
                        cfile->special = 1;
                        return cb_build_alphanumeric_literal ((ucharptr)s, strlen
       (s));
                }
                switch (cb_assign_clause) {
                case CB_ASSIGN_COBOL2002:
                        /* TODO */
                        return cb_error_node;

                case CB_ASSIGN_MF:
                        if (cfile->external_assign) {
                                p = strrchr (s, '-');
                                if (p) {
                                        s = p + 1;
                                }
                                return cb_build_alphanumeric_literal ((ucharptr)s
      , strlen (s));
                        }
                        current_program->reference_list =
                            cb_list_add (current_program->reference_list, name);
                        return name;

                case CB_ASSIGN_IBM:
                        /* check organization */
                        if (strncmp (s, "S-", 2) == 0 ||
```

```
                            strncmp (s, "AS-", 3) == 0) {
                                goto org;
                        }
                        /* skip the device label if exists */
                        if ((p = strchr (s, '-')) != NULL) {
                                s = p + 1;
                        }
                        /* check organization again */
                        if (strncmp (s, "S-", 2) == 0 ||
                            strncmp (s, "AS-", 3) == 0) {
org:
                                /* skip it for now */
                                s = strchr (s, '-') + 1;
                        }
                        /* convert the name into literal */
                        return cb_build_alphanumeric_literal ((ucharptr)s, strlen
       (s));
                }

        default:
                return cb_error_node;
        }
}
```

7.16.4.12 cb_tree cb_build_binary_list (cb_tree *l*, int *op*)

Definition at line 2019 of file tree.c.

```
{
        cb_tree e;

        e = CB_VALUE (l);
        for (l = CB_CHAIN (l); l; l = CB_CHAIN (l)) {
                e = cb_build_binary_op (e, op, CB_VALUE (l));
        }
        return e;
}
```

7.16.4.13 cb_tree cb_build_binary_op (cb_tree *x*, int *op*, cb_tree *y*)

Definition at line 1954 of file tree.c.

```
{
        struct cb_binary_op     *p;
        enum cb_category        category = CB_CATEGORY_UNKNOWN;

        switch (op) {
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
                /* arithmetic operators */
                if (CB_TREE_CLASS (x) == CB_CLASS_POINTER ||
                    CB_TREE_CLASS (y) == CB_CLASS_POINTER) {
```

```
                        category = CB_CATEGORY_DATA_POINTER;
                        break;
                }
                x = cb_check_numeric_value (x);
                y = cb_check_numeric_value (y);
                if (x == cb_error_node || y == cb_error_node) {
                        return cb_error_node;
                }
                category = CB_CATEGORY_NUMERIC;
                break;

        case '=':
        case '~':
        case '<':
        case '>':
        case '[':
        case ']':
                /* relational operators */
                category = CB_CATEGORY_BOOLEAN;
                break;

        case '!':
        case '&':
        case '|':
                /* logical operators */
                if (CB_TREE_CLASS (x) != CB_CLASS_BOOLEAN ||
                    (y && CB_TREE_CLASS (y) != CB_CLASS_BOOLEAN)) {
                        cb_error (_("Invalid expression"));
                        return cb_error_node;
                }
                category = CB_CATEGORY_BOOLEAN;
                break;

        case '@':
                /* parentheses */
                category = CB_TREE_CATEGORY (x);
                break;

        default:
                fprintf (stderr, "Unexpected operator -> %d\n", op);
                ABORT ();
        }

        p = make_tree (CB_TAG_BINARY_OP, category, sizeof (struct cb_binary_op));

        p->op = op;
        p->x = x;
        p->y = y;
        return CB_TREE (p);
}
```

7.16.4.14 cb_tree cb_build_call (cb_tree *name*, cb_tree *args*, cb_tree *stmt1*, cb_tree *stmt2*, cb_tree *returning*, int *is_system_call*)

Definition at line 2149 of file tree.c.

```
{
        struct cb_call *p;
```

```
        p = make_tree (CB_TAG_CALL, CB_CATEGORY_UNKNOWN, sizeof (struct cb_call))
      ;
        p->name = name;
        p->args = args;
        p->stmt1 = stmt1;
        p->stmt2 = stmt2;
        p->returning = returning;
        p->is_system = is_system_call;
        return CB_TREE (p);
}
```

7.16.4.15 cb_tree cb_build_cast (enum cb_cast_type *type*, cb_tree *val*)

Definition at line 2060 of file tree.c.

```
{
        struct cb_cast          *p;
        enum cb_category        category;

        if (type == CB_CAST_INTEGER) {
                category = CB_CATEGORY_NUMERIC;
        } else {
                category = CB_CATEGORY_UNKNOWN;
        }
        p = make_tree (CB_TAG_CAST, category, sizeof (struct cb_cast));
        p->type = type;
        p->val = val;
        return CB_TREE (p);
}
```

7.16.4.16 cb_tree cb_build_class_name (cb_tree *name*, cb_tree *list*)

Definition at line 939 of file tree.c.

```
{
        struct cb_class_name    *p;
        char                    buff[COB_MINI_BUFF];

        p = make_tree (CB_TAG_CLASS_NAME, CB_CATEGORY_BOOLEAN, sizeof (struct
      cb_class_name));
        p->name = cb_define (name, CB_TREE (p));
        snprintf (buff, COB_MINI_MAX, "is_%s", to_cname (p->name));
        p->cname = strdup (buff);
        p->list = list;
        return CB_TREE (p);
}
```

7.16.4.17 cb_tree cb_build_cond (cb_tree *x*)

Definition at line 2354 of file typeck.c.

```
{
        int                     size1;
        int                     size2;
        struct cb_field         *f;
        struct cb_binary_op     *p;
        cb_tree                 d1;
        cb_tree                 d2;

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_CONST:
        case CB_TAG_FUNCALL:
                return x;
        case CB_TAG_REFERENCE:
                if (!CB_FIELD_P (cb_ref (x))) {
                        return cb_build_cond (cb_ref (x));
                }

                f = cb_field (x);

                /* level 88 condition */
                if (f->level == 88) {
                        /* We need to build a 88 condition at every occurrence
                           instead of once at the beginning because a 88 item
                           may be subscripted (i.e., it is not a constant tree).
      */
                        return cb_build_cond (build_cond_88 (x));
                }

                cb_error_x (x, _("Invalid expression"));
                return cb_error_node;
          case CB_TAG_BINARY_OP:
                p = CB_BINARY_OP (x);
                switch (p->op) {
                case '!':
                        return cb_build_negation (cb_build_cond (p->x));
                case '&':
                case '|':
                        return cb_build_binary_op (cb_build_cond (p->x), p->op,
      cb_build_cond (p->y));
                default:
                        if (CB_INDEX_P (p->x) || CB_INDEX_P (p->y)
                            || CB_TREE_CLASS (p->x) == CB_CLASS_POINTER
                            || CB_TREE_CLASS (p->y) == CB_CLASS_POINTER) {
                                x = cb_build_binary_op (p->x, '-', p->y);
                        } else if (CB_BINARY_OP_P (p->x) || CB_BINARY_OP_P (p->y)
      ) {
                                /* decimal comparison */
                                d1 = decimal_alloc ();
                                d2 = decimal_alloc ();

                                decimal_expand (d1, p->x);
                                decimal_expand (d2, p->y);
                                dpush (cb_build_funcall_2 ("cob_decimal_cmp", d1,
       d2));
                                decimal_free ();
                                decimal_free ();
                                x = cb_list_reverse (decimal_stack);
                                decimal_stack = NULL;
                        } else {
                                if (cb_chk_num_cond (p->x, p->y)) {
                                        size1 = cb_field_size (p->x);
                                        x = cb_build_funcall_3 ("memcmp",
```

```
                                        cb_build_cast_address (p->x),
                                        cb_build_cast_address (p->y),
                                        cb_int (size1));
                                break;
                        }
                        if (CB_TREE_CLASS (p->x) == CB_CLASS_NUMERIC
                            && CB_TREE_CLASS (p->y) == CB_CLASS_NUMERIC
                            && cb_fits_int (p->y)) {
                                x = cb_build_optim_cond (p);
                                break;
                        }

                        /* field comparison */
                        if ((CB_REF_OR_FIELD_P (p->x))
                           && (CB_TREE_CATEGORY (p->x) ==
      CB_CATEGORY_ALPHANUMERIC ||
                               CB_TREE_CATEGORY (p->x) ==
      CB_CATEGORY_ALPHABETIC)
                           && (cb_field_size (p->x) == 1)
                           && (!current_program->alphabet_name_list)
                           && (p->y == cb_space || p->y == cb_low ||
                               p->y == cb_high || p->y == cb_zero)) {
                                x = cb_build_funcall_2 ("$G", p->x, p->y)
      ;
                                break;
                        }
                        if (cb_chk_alpha_cond (p->x) && cb_chk_alpha_cond
       (p->y)) {
                                size1 = cb_field_size (p->x);
                                size2 = cb_field_size (p->y);
                        } else {
                                size1 = 0;
                                size2 = 0;
                        }
                        if (size1 == 1 && size2 == 1) {
                                x = cb_build_funcall_2 ("$G", p->x, p->y)
      ;
                        } else if (size1 != 0 && size1 == size2) {
                                x = cb_build_funcall_3 ("memcmp",
                                        cb_build_cast_address (p->x),
                                        cb_build_cast_address (p->y),
                                        cb_int (size1));
                        } else {
                                if (CB_TREE_CLASS (p->x) ==
      CB_CLASS_NUMERIC && p->y == cb_zero) {
                                        x = cb_build_optim_cond (p);
                                } else {
                                        x = cb_build_funcall_2 ("cob_cmp"
      , p->x, p->y);
                                }
                        }
                    }
                }
                return cb_build_binary_op (x, p->op, p->y);
        default:
                cb_error_x (x, _("Invalid expression"));
                return cb_error_node;
        }
/* NOT REACHED */
        return x;
}
```

7.16.4.18 cb_tree cb_build_const_length (cb_tree *x*)

Definition at line 956 of file typeck.c.

```
{
        struct cb_field         *f;
        char                    buff[64];

        if (x == cb_error_node) {
                return cb_error_node;
        }
        if (CB_REFERENCE_P (x) && cb_ref (x) == cb_error_node) {
                return cb_error_node;
        }

        memset (buff, 0, sizeof (buff));
        f = CB_FIELD (cb_ref (x));
        if (f->flag_any_length) {
                cb_error (_("ANY LENGTH item not allowed here"));
                return cb_error_node;
        }
        if (f->level == 88) {
                cb_error (_("88 level item not allowed here"));
                return cb_error_node;
        }
        if (!f->flag_is_verified) {
                cb_validate_field (f);
        }
        sprintf (buff, "%d", f->memory_size);
        return cb_build_numeric_literal (0, (ucharptr)buff, 0);
}
```

7.16.4.19 cb_tree cb_build_constant (cb_tree *name*, cb_tree *value*)

Definition at line 1446 of file tree.c.

```
{
        cb_tree x;

        x = cb_build_field (name);
        x->category = cb_tree_category (value);
        CB_FIELD (x)->storage = CB_STORAGE_CONSTANT;
        CB_FIELD (x)->values = cb_list_init (value);
        return x;
}
```

7.16.4.20 cb_tree cb_build_continue (void)

Definition at line 2240 of file tree.c.

```
{
        struct cb_continue *p;

        p = make_tree (CB_TAG_CONTINUE, CB_CATEGORY_UNKNOWN, sizeof (struct
```

```
      cb_continue));
        return CB_TREE (p);
}
```

7.16.4.21 cb_tree cb_build_converting (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3847 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_converting", x, y
      ));
}
```

7.16.4.22 cb_tree cb_build_decimal (int *id*)

Definition at line 1076 of file tree.c.

```
{
        struct cb_decimal *p;

        p = make_tree (CB_TAG_DECIMAL, CB_CATEGORY_NUMERIC, sizeof (struct
      cb_decimal));
        p->id = id;
        return CB_TREE (p);
}
```

7.16.4.23 cb_tree cb_build_display_upon (cb_tree *x*)

Definition at line 3406 of file typeck.c.

```
{
        if (x == cb_error_node) {
                return cb_error_node;
        }

        switch (CB_SYSTEM_NAME (cb_ref (x))->token) {
        case CB_DEVICE_CONSOLE:
        case CB_DEVICE_SYSOUT:
                return cb_int0;
        case CB_DEVICE_SYSERR:
                return cb_int1;
        default:
                cb_error_x (x, _("Invalid output stream"));
                return cb_error_node;
        }
}
```

7.16.4.24 cb_tree cb_build_display_upon_direct (cb_tree *x*)

Definition at line 3425 of file typeck.c.

```
{
        const char      *name;
        cb_tree         sys;

        if (x == cb_error_node) {
                return cb_error_node;
        }
        name = CB_NAME (x);
        if (CB_REFERENCE (x)->word->count == 0) {
                sys = lookup_system_name (CB_NAME (x));
                if (sys != cb_error_node) {
                        switch (CB_SYSTEM_NAME (sys)->token) {
                        case CB_DEVICE_CONSOLE:
                        case CB_DEVICE_SYSOUT:
                                cb_warning_x (x, _("'%s' undefined in SPECIAL-NAM
      ES"), name);
                                return cb_int0;
                        case CB_DEVICE_SYSERR:
                                cb_warning_x (x, _("'%s' undefined in SPECIAL-NAM
      ES"), name);
                                return cb_int1;
                        default:
                                break;
                        }
                }
        }

        cb_error_x (x, _("'%s' undefined in SPECIAL-NAMES"), name);
        return cb_error_node;
}
```

7.16.4.25 cb_tree cb_build_expr (cb_tree *list*)

Definition at line 1820 of file typeck.c.

```
{
        cb_tree l;
/* RXW
        cb_tree x;
*/
        int     op;

        cb_expr_init ();

        for (l = list; l; l = CB_CHAIN (l)) {
                op = CB_PURPOSE_INT (l);
                switch (op) {
                case '9': /* NUMERIC */
                        cb_expr_shift_class ("cob_is_numeric");
                        break;
                case 'A': /* ALPHABETIC */
                        cb_expr_shift_class ("cob_is_alpha");
                        break;
                case 'L': /* ALPHABETIC_LOWER */
```

```
                        cb_expr_shift_class ("cob_is_lower");
                        break;
                case 'U': /* ALPHABETIC_UPPER */
                        cb_expr_shift_class ("cob_is_upper");
                        break;
                case 'P': /* POSITIVE */
                        cb_expr_shift_sign ('>');
                        break;
                case 'N': /* NEGATIVE */
                        cb_expr_shift_sign ('<');
                        break;
                case 'O': /* OMITTED */
                        current_statement->null_check = NULL;
                        cb_expr_shift_class ("cob_is_omitted");
                        break;
/* RXW
                case 'x':
                        if (CB_VALUE (l) && CB_REFERENCE_P (CB_VALUE (l))) {
                                x = CB_CHAIN (l);
                                if (x && cb_field (CB_VALUE (l))->level == 88) {
                                        switch (CB_PURPOSE_INT (x)) {
                                        case '&':
                                        case '|':
                                        case '(':
                                        case ')':
                                                break;
                                        default:
                                                cb_error (_("Invalid condition"))
      ;
                                                break;
                                        }
                                }
                        }
                        cb_expr_shift (op, CB_VALUE (l));
                        break;
*/
                default:
                        cb_expr_shift (op, CB_VALUE (l));
                        break;
                }
        }

        return cb_expr_finish ();
}
```

7.16.4.26 cb_tree cb_build_field (cb_tree *name*)

Definition at line 1417 of file tree.c.

```
{
        struct cb_field *p;

        p = make_tree (CB_TAG_FIELD, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_field));
        p->id = cb_field_id++;
        p->name = cb_define (name, CB_TREE (p));
        p->ename = NULL;
        p->usage = CB_USAGE_DISPLAY;
        p->storage = CB_STORAGE_WORKING;
```

```
        p->occurs_max = 1;
        return CB_TREE (p);
}
```

7.16.4.27 cb_tree cb_build_field_reference (struct cb_field * *f,* cb_tree *ref*)

Definition at line 1752 of file tree.c.

```
{
        cb_tree         x;
        struct cb_word  *word;

        x = cb_build_reference (f->name);
        word = CB_REFERENCE (x)->word;
        if (ref) {
                memcpy (x, ref, sizeof (struct cb_reference));
        }
        x->category = CB_CATEGORY_UNKNOWN;
        CB_REFERENCE (x)->word = word;
        CB_REFERENCE (x)->value = CB_TREE (f);
        return x;
}
```

7.16.4.28 cb_tree cb_build_field_tree (cb_tree *level,* cb_tree *name,* struct cb_field * *last_field,* enum cb_storage *storage,* struct cb_file * *fn*)

Definition at line 78 of file field.c.

```
{
        struct cb_reference     *r;
        struct cb_field         *f;
        struct cb_field         *p;
        struct cb_field         *field_fill;
        cb_tree                 dummy_fill;
        cb_tree                 l;
        cb_tree                 x;
        int                     lv;

        if (level == cb_error_node || name == cb_error_node) {
                return cb_error_node;
        }

        /* check the level number */
        lv = cb_get_level (level);
        if (!lv) {
                return cb_error_node;
        }

        /* build the field */
        r = CB_REFERENCE (name);
        f = CB_FIELD (cb_build_field (name));
        f->storage = storage;
        last_real_field = last_field;
        if (lv == 78) {
                f->level = 01;
```

```
                f->flag_item_78 = 1;
                return CB_TREE (f);
        } else {
                f->level = lv;
        }
        if (f->level == 01 && storage == CB_STORAGE_FILE) {
                if (fn->external) {
                        f->flag_external = 1;
                        has_external = 1;
                } else if (fn->global) {
                        f->flag_is_global = 1;
                }
        }
        if (last_field) {
                if (last_field->level == 77 && f->level != 01 &&
                    f->level != 77 && f->level != 66 && f->level != 88) {
                        cb_error_x (name, _("Level number must begin with 01 or 7
      7"));
                        return cb_error_node;
                }
        }

        /* checks for redefinition */
        if (cb_warn_redefinition) {
                if (r->word->count > 1) {
                        if (f->level == 01 || f->level == 77) {
                                redefinition_warning (name, NULL);
                        } else {
                                for (l = r->word->items; l; l = CB_CHAIN (l)) {
                                        x = CB_VALUE (l);
                                        if (!CB_FIELD_P (x)
                                            || CB_FIELD (x)->level == 01
                                            || CB_FIELD (x)->level == 77
                                            || (f->level == last_field->level
                                                && CB_FIELD (x)->parent == last_f
      ield->parent)) {
                                                redefinition_warning (name, x);
                                                break;
                                        }
                                }
                        }
                }
        }

        if (last_field && last_field->level == 88) {
                last_field = last_field->parent;
        }

        /* link the field into the tree */
        if (f->level == 01 || f->level == 77) {
                /* top level */
                cb_needs_01 = 0;
                if (last_field) {
/*
                        cb_field_add (cb_field_founder (last_field), f);
*/
                        cb_field_founder (last_field)->sister = f;
                }
        } else if (!last_field || cb_needs_01) {
                /* invalid top level */
                cb_error_x (name, _("Level number must begin with 01 or 77"));
                return cb_error_node;
```

```
        } else if (f->level == 66) {
                /* level 66 */
                f->parent = cb_field_founder (last_field);
                for (p = f->parent->children; p && p->sister; p = p->sister) ;
                if (p) {
                        p->sister = f;
                }
        } else if (f->level == 88) {
                /* level 88 */
                f->parent = last_field;
        } else if (f->level > last_field->level) {
                /* lower level */
                last_field->children = f;
                f->parent = last_field;
        } else if (f->level == last_field->level) {
                /* same level */
same_level:
                last_field->sister = f;
                f->parent = last_field->parent;
        } else {
                /* upper level */
                for (p = last_field->parent; p; p = p->parent) {
                        if (p->level == f->level) {
                                last_field = p;
                                goto same_level;
                        }
                        if (cb_relax_level_hierarchy && p->level < f->level) {
                                break;
                        }
                }
                if (cb_relax_level_hierarchy) {
                        dummy_fill = cb_build_filler ();
                        field_fill = CB_FIELD (cb_build_field (dummy_fill));
                        cb_warning_x (name, _("No previous data item of level %02
      d"), f->level);
                        field_fill->level = f->level;
                        field_fill->storage = storage;
                        field_fill->children = p->children;
                        field_fill->parent = p;
                        for (p = p->children; p != NULL; p = p->sister) {
                                p->parent = field_fill;
                        }
                        field_fill->parent->children = field_fill;
                        field_fill->sister = f;
                        f->parent = field_fill->parent;
                        last_field = field_fill;
                } else {
                        cb_error_x (name, _("No previous data item of level %02d"
      ), f->level);
                        return cb_error_node;
                }
        }

        /* inherit parent's properties */
        if (f->parent) {
                f->usage = f->parent->usage;
                f->indexes = f->parent->indexes;
                f->flag_sign_leading = f->parent->flag_sign_leading;
                f->flag_sign_separate = f->parent->flag_sign_separate;
                f->flag_is_global = f->parent->flag_is_global;
        }
        return CB_TREE (f);
```

```
}
```

7.16.4.29 cb_tree cb_build_filler (void)

Definition at line 1740 of file tree.c.

```
{
        cb_tree         x;
        char            name[16];

        sprintf (name, "WORK$%d", filler_id++);
        x = cb_build_reference (name);
        x->source_line = cb_source_line;
        return x;
}
```

7.16.4.30 cb_tree cb_build_funcall (const char * *name*, int *argc*, cb_tree *a1*, cb_tree *a2*, cb_tree *a3*, cb_tree *a4*, cb_tree *a5*, cb_tree *a6*, cb_tree *a7*)

Definition at line 2035 of file tree.c.

```
{
        struct cb_funcall *p;

        p = make_tree (CB_TAG_FUNCALL, CB_CATEGORY_BOOLEAN, sizeof (struct
      cb_funcall));
        p->name = name;
        p->argc = argc;
        p->varcnt = 0;
        p->screenptr = gen_screen_ptr;
        p->argv[0] = a1;
        p->argv[1] = a2;
        p->argv[2] = a3;
        p->argv[3] = a4;
        p->argv[4] = a5;
        p->argv[5] = a6;
        p->argv[6] = a7;
        return CB_TREE (p);
}
```

7.16.4.31 cb_tree cb_build_goto (cb_tree *target*, cb_tree *depending*)

Definition at line 2168 of file tree.c.

```
{
        struct cb_goto *p;

        p = make_tree (CB_TAG_GOTO, CB_CATEGORY_UNKNOWN, sizeof (struct cb_goto))
      ;
        p->target = target;
        p->depending = depending;
        return CB_TREE (p);
}
```

7.16.4.32 cb_tree cb_build_identifier (cb_tree *x*)

Definition at line 763 of file typeck.c.

```
{
        struct cb_reference     *r;
        struct cb_field         *f;
        struct cb_field         *p;
        const char              *name;
        cb_tree                 v;
        cb_tree                 e1;
        cb_tree                 e2;
        cb_tree                 l;
        cb_tree                 sub;
        int                     offset;
        int                     length;
        int                     n;

        if (x == cb_error_node) {
                return cb_error_node;
        }

        r = CB_REFERENCE (x);
        name = r->word->name;

        /* resolve reference */
        v = cb_ref (x);
        if (v == cb_error_node) {
                return cb_error_node;
        }

        /* check if it is a data name */
        if (!CB_FIELD_P (v)) {
                if (r->subs) {
                        cb_error_x (x, _("'%s' cannot be subscripted"), name);
                        return cb_error_node;
                }
                if (r->offset) {
                        cb_error_x (x, _("'%s' cannot be reference modified"), na
      me);
                        return cb_error_node;
                }
                return x;
        }
        f = CB_FIELD (v);

        /* BASED check */
        if (CB_EXCEPTION_ENABLE (COB_EC_BOUND_PTR)) {
                for (p = f; p->parent; p = p->parent) {
                        ;
                }
                if (current_statement) {
                        if (p->flag_item_based ||
                           (f->storage == CB_STORAGE_LINKAGE &&
                            !p->flag_is_pdiv_parm)) {
                                current_statement->null_check =
      cb_build_funcall_2 (
                                        "cob_check_based",
                                        cb_build_address (
      cb_build_field_reference (p, NULL)),
                                        cb_build_string0 ((ucharptr)name));
                        }
```

```
        }
  }

  /* check the number of subscripts */
  if (!r->all && cb_list_length (r->subs) != f->indexes) {
        switch (f->indexes) {
        case 0:
                cb_error_x (x, _("'%s' cannot be subscripted"), name);
                return cb_error_node;
        case 1:
                cb_error_x (x, _("'%s' requires 1 subscript"), name);
                return cb_error_node;
        default:
                cb_error_x (x, _("'%s' requires %d subscripts"), name, f-
>indexes);
                return cb_error_node;
        }
  }

  /* subscript check */
  if (!r->all && r->subs) {
        l = r->subs;
        for (p = f; p; p = p->parent) {
                if (p->flag_occurs) {
                        sub = cb_check_integer_value (CB_VALUE (l));

                        l = CB_CHAIN (l);

                        if (sub == cb_error_node) {
                                continue;
                        }

                        /* compile-time check */
                        if (CB_LITERAL_P (sub)) {
                                n = cb_get_int (sub);
                                if (n < 1 || n > p->occurs_max) {
                                        cb_error_x (x, _("Subscript of '%
s' out of bounds: %d"),
                                                    name, n);
                                }
                        }

                        /* run-time check */
                        if (CB_EXCEPTION_ENABLE (COB_EC_BOUND_SUBSCRIPT))
 {
                                if (p->occurs_depending) {
                                        e1 = cb_build_funcall_4 ("cob_che
ck_odo",
                                                 cb_build_cast_integer (p
->occurs_depending),
                                                 cb_int (p->occurs_min),
                                                 cb_int (p->occurs_max),
                                                 cb_build_string0
                                                 ((ucharptr)(cb_field (p-
>occurs_depending)->name)));
                                        e2 = cb_build_funcall_4 ("cob_che
ck_subscript",
                                                 cb_build_cast_integer (s
ub),
                                                 cb_int1,
                                                 cb_build_cast_integer (p
->occurs_depending),
```

```
                                                        cb_build_string0 ((
ucharptr)name));
                                                r->check = cb_list_add (r->check,
 e1);
                                                r->check = cb_list_add (r->check,
 e2);
                                        } else {
                                                if (!CB_LITERAL_P (sub)) {
                                                        e1 = cb_build_funcall_4 (
"cob_check_subscript",
                                                                                 
cb_build_cast_integer (sub),
                                                                                 
                                                                cb_int1,
                                                                cb_int (p->
occurs_max),
                                                                cb_build_string0
((ucharptr)name));
                                                        r->check = cb_list_add (r
->check, e1);
                                                }
                                        }
                                }
                        }
                }
        }

        /* reference modification check */
        if (r->offset) {
                /* compile-time check */
                if (CB_LITERAL_P (r->offset)) {
                        offset = cb_get_int (r->offset);
                        if (offset < 1 || offset > f->size) {
                                cb_error_x (x, _("Offset of '%s' out of bounds: %
d"), name, offset);
                        } else if (r->length && CB_LITERAL_P (r->length)) {
                                length = cb_get_int (r->length);
                                if (length < 1 || length > f->size - offset + 1)
{
                                        cb_error_x (x, _("Length of '%s' out of b
ounds: %d"),
                                                    name, length);
                                }
                        }
                }

                /* run-time check */
                if (CB_EXCEPTION_ENABLE (COB_EC_BOUND_REF_MOD)) {
                        if (!CB_LITERAL_P (r->offset)
                            || (r->length && !CB_LITERAL_P (r->length))) {
                                e1 = cb_build_funcall_4 ("cob_check_ref_mod",
                                                         cb_build_cast_integer (r
->offset),
                                                         r->length ?
cb_build_cast_integer (r->length) :
                                                         cb_int1, cb_int (f->
size),
                                                         cb_build_string0 ((
ucharptr)f->name));
                                r->check = cb_list_add (r->check, e1);
                        }
                }
        }
```

```
        if (f->storage == CB_STORAGE_CONSTANT) {
                return CB_VALUE (f->values);
        }

        return x;
}
```

7.16.4.33 cb_tree cb_build_if (cb_tree *test*, cb_tree *stmt1*, cb_tree *stmt2*)

Definition at line 2183 of file tree.c.

```
{
        struct cb_if *p;

        p = make_tree (CB_TAG_IF, CB_CATEGORY_UNKNOWN, sizeof (struct cb_if));
        p->test = test;
        p->stmt1 = stmt1;
        p->stmt2 = stmt2;
        return CB_TREE (p);
}
```

7.16.4.34 cb_tree cb_build_implicit_field (cb_tree *name*, int *len*)

Definition at line 1432 of file tree.c.

```
{
        cb_tree x;
        char    pic[32];

        x = cb_build_field (name);
        memset (pic, 0, sizeof(pic));
        sprintf (pic, "X(%d)", len);
        CB_FIELD (x)->pic = CB_PICTURE (cb_build_picture (pic));
        cb_validate_field (CB_FIELD (x));
        return x;
}
```

7.16.4.35 cb_tree cb_build_index (cb_tree *name*, cb_tree *values*, int *indexed_by*, struct cb_field * *qual*)

Definition at line 744 of file typeck.c.

```
{
        struct cb_field *f;

        f = CB_FIELD (cb_build_field (x));
        f->usage = CB_USAGE_INDEX;
        cb_validate_field (f);
        if (values) {
                f->values = cb_list_init (values);
```

```
        }
        if (qual) {
                f->index_qual = qual;
        }
        f->flag_indexed_by = indexed_by;
        current_program->working_storage = cb_field_add (current_program->
      working_storage, f);
        return x;
}
```

7.16.4.36 cb_tree cb_build_initialize (cb_tree *var*, cb_tree *val*, cb_tree *rep*, cb_tree *def*, int *flag*)

Definition at line 2113 of file tree.c.

```
{
        struct cb_initialize *p;

        p = make_tree (CB_TAG_INITIALIZE, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_initialize));
        p->var = var;
        p->val = val;
        p->rep = rep;
        p->def = def;
        p->flag_statement = flag;
        return CB_TREE (p);
}
```

7.16.4.37 cb_tree cb_build_inspect_region (cb_tree *l*, cb_tree *pos*, cb_tree *x*)

Definition at line 3859 of file typeck.c.

```
{
        if (pos == CB_BEFORE) {
                return cb_list_add (l, cb_build_funcall_1 ("cob_inspect_before",
      x));
        } else {
                return cb_list_add (l, cb_build_funcall_1 ("cob_inspect_after", x
      ));
        }
}
```

7.16.4.38 cb_tree cb_build_inspect_region_start (void)

Definition at line 3853 of file typeck.c.

```
{
        return cb_list_init (cb_build_funcall_0 ("cob_inspect_start"));
}
```

7.16.4.39 cb_tree cb_build_intrinsic (cb_tree *name*, cb_tree *args*, cb_tree *refmod*)

Definition at line 2262 of file tree.c.

```
{
        struct cb_intrinsic_table       *cbp;
        cb_tree                         x;
        int                             numargs;

        numargs = cb_list_length (args);

        cbp = lookup_intrinsic (CB_NAME (name), 0);
        if (cbp) {
                if ((cbp->args != -1 && numargs != cbp->args) ||
                    (cbp->args == -1 && cbp->intr_enum != CB_INTR_RANDOM && numar
      gs < 1)) {
                        cb_error_x (name, _("FUNCTION %s has wrong number of argu
      ments"), cbp->name);
                        return cb_error_node;
                }
                if (refmod) {
                        if (!cbp->refmod) {
                                cb_error_x (name, _("FUNCTION %s can not have ref
      erence modification"), cbp->name);
                                return cb_error_node;
                        }
                        if (CB_LITERAL_P(CB_PAIR_X(refmod)) &&
                            cb_get_int (CB_PAIR_X(refmod))< 1) {
                                cb_error_x (name, _("FUNCTION %s has invalid refe
      rence modification"), cbp->name);
                                return cb_error_node;
                        }
                        if (CB_PAIR_Y(refmod) && CB_LITERAL_P(CB_PAIR_Y(refmod))
      &&
                            cb_get_int (CB_PAIR_Y(refmod))< 1) {
                                cb_error_x (name, _("FUNCTION %s has invalid refe
      rence modification"), cbp->name);
                                return cb_error_node;
                        }
                }
                /* cb_tree      x; */
                switch (cbp->intr_enum) {
                case CB_INTR_LENGTH:
                case CB_INTR_BYTE_LENGTH:
                        x = CB_VALUE (args);
                        if (CB_INTRINSIC_P (x)) {
                                return make_intrinsic (name, cbp, args, NULL,
      NULL);
                        } else if ((CB_FIELD_P (x) || CB_REFERENCE_P (x)) &&
                                    cb_field(x)->flag_any_length) {
                                return make_intrinsic (name, cbp, args, NULL,
      NULL);
                        } else {
                                return cb_build_length (CB_VALUE (args));
                        }

                case CB_INTR_WHEN_COMPILED:
                        if (refmod) {
                                return make_intrinsic (name, cbp,
                                        cb_list_init (cb_intr_whencomp), NULL, re
      fmod);
                        } else {
```

```
                                return cb_intr_whencomp;
                        }
                case CB_INTR_PI:
                        return cb_intr_pi;
                case CB_INTR_E:
                        return cb_intr_e;

                case CB_INTR_LOWER_CASE:
                case CB_INTR_UPPER_CASE:
                case CB_INTR_REVERSE:
/* RXW Why did I do this ? - still do not know
                        if (CB_INTRINSIC_P (CB_VALUE (args))) {
                                return make_intrinsic (name, cbp, args, cb_int0);

                        } else {
                                return make_intrinsic (name, cbp, args,
                                                       cb_build_length (CB_VALUE
      (args)));
                        }
RXW */

                case CB_INTR_ABS:
                case CB_INTR_ACOS:
                case CB_INTR_ANNUITY:
                case CB_INTR_ASIN:
                case CB_INTR_ATAN:
                case CB_INTR_CHAR:
                case CB_INTR_COMBINED_DATETIME:
                case CB_INTR_COS:
                case CB_INTR_CURRENT_DATE:
                case CB_INTR_DATE_OF_INTEGER:
                case CB_INTR_DAY_OF_INTEGER:
                case CB_INTR_EXCEPTION_FILE:
                case CB_INTR_EXCEPTION_LOCATION:
                case CB_INTR_EXCEPTION_STATUS:
                case CB_INTR_EXCEPTION_STATEMENT:
                case CB_INTR_EXP:
                case CB_INTR_EXP10:
                case CB_INTR_FACTORIAL:
                case CB_INTR_FRACTION_PART:
                case CB_INTR_INTEGER:
                case CB_INTR_INTEGER_OF_DATE:
                case CB_INTR_INTEGER_OF_DAY:
                case CB_INTR_INTEGER_PART:
                case CB_INTR_LOCALE_DATE:
                case CB_INTR_LOCALE_TIME:
                case CB_INTR_LOCALE_TIME_FROM_SECS:
                case CB_INTR_LOG:
                case CB_INTR_LOG10:
                case CB_INTR_MOD:
                case CB_INTR_NUMVAL:
                case CB_INTR_NUMVAL_C:
                case CB_INTR_ORD:
                case CB_INTR_REM:
                case CB_INTR_SECONDS_FROM_FORMATTED_TIME:
                case CB_INTR_SECONDS_PAST_MIDNIGHT:
                case CB_INTR_SIGN:
                case CB_INTR_SIN:
                case CB_INTR_SQRT:
                case CB_INTR_STORED_CHAR_LENGTH:
                case CB_INTR_TAN:
                case CB_INTR_TEST_DATE_YYYYMMDD:
```

```
                case CB_INTR_TEST_DAY_YYYYDDD:
                case CB_INTR_TRIM:
                        return make_intrinsic (name, cbp, args, NULL, refmod);

                case CB_INTR_CONCATENATE:
                        return make_intrinsic (name, cbp, args, cb_int1, refmod);

                case CB_INTR_DATE_TO_YYYYMMDD:
                case CB_INTR_DAY_TO_YYYYDDD:
                case CB_INTR_MAX:
                case CB_INTR_MEAN:
                case CB_INTR_MEDIAN:
                case CB_INTR_MIDRANGE:
                case CB_INTR_MIN:
                case CB_INTR_ORD_MAX:
                case CB_INTR_ORD_MIN:
                case CB_INTR_PRESENT_VALUE:
                case CB_INTR_RANDOM:
                case CB_INTR_RANGE:
                case CB_INTR_STANDARD_DEVIATION:
                case CB_INTR_SUM:
                case CB_INTR_VARIANCE:
                case CB_INTR_YEAR_TO_YYYY:
                        return make_intrinsic (name, cbp, args, cb_int1, NULL);
                case CB_INTR_SUBSTITUTE:
                case CB_INTR_SUBSTITUTE_CASE:
                        if (numargs < 3 || (numargs % 2) == 0) {
                                cb_error_x (name, _("FUNCTION %s has wrong number
       of arguments"), cbp->name);
                                return cb_error_node;
                        }
                        return make_intrinsic (name, cbp, args, cb_int1, refmod);


                default:
                        break;
                }
        }
        cb_error_x (name, _("FUNCTION %s not implemented"), CB_NAME (name));
        return cb_error_node;
}
```

7.16.4.40 cb_tree cb_build_label (cb_tree *name*, struct cb_label * *section*)

Definition at line 2081 of file tree.c.

```
{
        struct cb_label *p;

        p = make_tree (CB_TAG_LABEL, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_label));
        p->id = cb_id++;
        p->name = (const unsigned char *)cb_define (name, CB_TREE (p));
        p->orig_name = p->name;
        p->section = section;
        return CB_TREE (p);
}
```

7.16.4.41 cb_tree cb_build_length (cb_tree *x*)

Definition at line 986 of file typeck.c.

```
{
        struct cb_field         *f;
        struct cb_literal       *l;
        cb_tree                 temp;
        char                    buff[64];

        if (x == cb_error_node) {
                return cb_error_node;
        }
        if (CB_REFERENCE_P (x) && cb_ref (x) == cb_error_node) {
                return cb_error_node;
        }

        memset (buff, 0, sizeof (buff));
        if (CB_LITERAL_P (x)) {
                l = CB_LITERAL (x);
                sprintf (buff, "%d", (int)l->size);
                return cb_build_numeric_literal (0, (ucharptr)buff, 0);
        }
        if (CB_REF_OR_FIELD_P (x)) {
                f = CB_FIELD (cb_ref (x));
                if (f->flag_any_length) {
                        return cb_build_any_intrinsic (cb_list_init (x));
                }
                if (cb_field_variable_size (f) == NULL) {
                        sprintf (buff, "%d", cb_field_size (x));
                        return cb_build_numeric_literal (0, (ucharptr)buff, 0);
                }
        }
        if (CB_INTRINSIC_P (x)) {
                return cb_build_any_intrinsic (cb_list_init (x));
        }
        temp = cb_build_index (cb_build_filler (), NULL, 0, NULL);
        CB_FIELD (cb_ref (temp))->usage = CB_USAGE_LENGTH;
        CB_FIELD (cb_ref (temp))->count++;
        cb_emit (cb_build_assign (temp, cb_build_length_1 (x)));
        return temp;
}
```

7.16.4.42 cb_tree cb_build_list (cb_tree *purpose*, cb_tree *value*, cb_tree *rest*)

Definition at line 769 of file tree.c.

```
{
        struct cb_list *p;

        p = make_tree (CB_TAG_LIST, CB_CATEGORY_UNKNOWN, sizeof (struct cb_list))
      ;
        p->purpose = purpose;
        p->value = value;
        p->chain = rest;
        return CB_TREE (p);
}
```

7.16.4.43 cb_tree cb_build_locale_name (cb_tree *name*, cb_tree *list*)

Definition at line 957 of file tree.c.

```
{
        struct cb_class_name    *p;

        p = make_tree (CB_TAG_LOCALE_NAME, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_locale_name));
        p->name = cb_define (name, CB_TREE (p));
        p->cname = to_cname (p->name);
        p->list = list;
        return CB_TREE (p);
}
```

7.16.4.44 cb_tree cb_build_move (cb_tree *src*, cb_tree *dst*)

Definition at line 4964 of file typeck.c.

```
{
        struct cb_field *f;
        struct cb_field *p;

        if (src == cb_error_node || dst == cb_error_node) {
                return cb_error_node;
        }

        if (validate_move (src, dst, 0) < 0) {
                return cb_error_node;
        }

        if (CB_REFERENCE_P (src)) {
                CB_REFERENCE (src)->type = CB_SENDING_OPERAND;
        }
        if (CB_REFERENCE_P (dst)) {
                CB_REFERENCE (dst)->type = CB_RECEIVING_OPERAND;
        }

        if (CB_TREE_CLASS (dst) == CB_CLASS_POINTER) {
                return cb_build_assign (dst, src);
        }

        if (CB_REFERENCE_P (src) && CB_ALPHABET_NAME_P(CB_REFERENCE(src)->value))
       {
                return cb_build_move_call (src, dst);
        }
        if (CB_INDEX_P (dst)) {
                if (src == cb_null) {
                        return cb_build_assign (dst, cb_zero);
                }
                return cb_build_assign (dst, src);
        }

        if (CB_INDEX_P (src)) {
                return cb_build_funcall_2 ("cob_set_int", dst,
      cb_build_cast_integer (src));
        }
```

```
        if (CB_INTRINSIC_P (src) || CB_INTRINSIC_P (dst)) {
                return cb_build_move_call (src, dst);
        }

        f = cb_field (dst);

        if (CB_EXCEPTION_ENABLE (COB_EC_BOUND_SUBSCRIPT)) {
                for (p = f; p; p = p->parent) {
                        if (p->flag_occurs) {
                                return cb_build_move_call (src, dst);
                        }
                }
                if (CB_REF_OR_FIELD_P (src)) {
                        for (p = cb_field (src); p; p = p->parent) {
                                if (p->flag_occurs) {
                                        return cb_build_move_call (src, dst);
                                }
                        }
                }
        }

        /* output optimal code */
        if (src == cb_zero) {
                return cb_build_move_zero (dst);
        } else if (src == cb_space) {
                return cb_build_move_space (dst);
        } else if (src == cb_high) {
                return cb_build_move_high (dst);
        } else if (src == cb_low) {
                return cb_build_move_low (dst);
        } else if (src == cb_quote) {
                return cb_build_move_quote (dst);
        } else if (CB_LITERAL_P (src)) {
                return cb_build_move_literal (src, dst);
        }
        return cb_build_move_field (src, dst);
}
```

7.16.4.45 cb_tree cb_build_numeric_literal (int *sign,* const unsigned char * *data,* int *scale*)

Definition at line 988 of file tree.c.

```
{
        struct cb_literal *p;

        p = build_literal (CB_CATEGORY_NUMERIC, data, strlen ((char *)data));
        p->sign = (char)sign;
        p->scale = (char)scale;
        return CB_TREE (p);
}
```

7.16.4.46 cb_tree cb_build_perform (int *type*)

Definition at line 2199 of file tree.c.

```
{
```

```
        struct cb_perform *p;

        p = make_tree (CB_TAG_PERFORM, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_perform));
        p->type = type;
        return CB_TREE (p);
}
```

7.16.4.47 cb_tree cb_build_perform_exit (struct cb_label * *label*)

Definition at line 5156 of file typeck.c.

```
{
        cb_tree x;

        x = cb_build_perform (CB_PERFORM_EXIT);
        CB_PERFORM (x)->data = CB_TREE (label);
        return x;
}
```

7.16.4.48 cb_tree cb_build_perform_forever (cb_tree *body*)

Definition at line 5143 of file typeck.c.

```
{
        cb_tree x;

        if (body == cb_error_node) {
                return cb_error_node;
        }
        x = cb_build_perform (CB_PERFORM_FOREVER);
        CB_PERFORM (x)->body = body;
        return x;
}
```

7.16.4.49 cb_tree cb_build_perform_once (cb_tree *body*)

Definition at line 5105 of file typeck.c.

```
{
        cb_tree x;

        if (body == cb_error_node) {
                return cb_error_node;
        }
        x = cb_build_perform (CB_PERFORM_ONCE);
        CB_PERFORM (x)->body = body;
        return x;
}
```

7.16.4.50 cb_tree cb_build_perform_times (cb_tree *count*)

Definition at line 5118 of file typeck.c.

```
{
        cb_tree x;

        if (cb_check_integer_value (times) == cb_error_node) {
                return cb_error_node;
        }

        x = cb_build_perform (CB_PERFORM_TIMES);
        CB_PERFORM (x)->data = times;
        return x;
}
```

7.16.4.51 cb_tree cb_build_perform_until (cb_tree *condition,* cb_tree *varying*)

Definition at line 5132 of file typeck.c.

```
{
        cb_tree x;

        x = cb_build_perform (CB_PERFORM_UNTIL);
        CB_PERFORM (x)->test = condition;
        CB_PERFORM (x)->varying = varying;
        return x;
}
```

7.16.4.52 cb_tree cb_build_perform_varying (cb_tree *name,* cb_tree *from,* cb_tree *step,* cb_tree *until*)

Definition at line 2209 of file tree.c.

```
{
        struct cb_perform_varying *p;

        p = make_tree (CB_TAG_PERFORM_VARYING, CB_CATEGORY_UNKNOWN, sizeof (struc
      t cb_perform_varying));
        p->name = name;
        p->from = from;
        p->step = name ? cb_build_add (name, by, cb_high) : NULL;
        p->until = until;
        return CB_TREE (p);
}
```

7.16.4.53 cb_tree cb_build_picture (const char * *str*)

Definition at line 1090 of file tree.c.

```
{
        struct cb_picture       *pic;
        const char              *p;
        size_t                  idx = 0;
        size_t                  buffcnt = 0;
        size_t                  at_beginning;
        size_t                  at_end;
        size_t                  p_char_seen;
        size_t                  s_char_seen;
        int                     category = 0;
        int                     size = 0;
        int                     allocated = 0;
        int                     digits = 0;
        int                     scale = 0;
        int                     s_count = 0;
        int                     v_count = 0;
        int                     i;
        int                     n;
        unsigned char           c;
        unsigned char           lastonechar = 0;
        unsigned char           lasttwochar = 0;
        unsigned char           buff[COB_SMALL_BUFF];

        pic = make_tree (CB_TAG_PICTURE, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_picture));
        if (strlen (str) > 50) {
                goto error;
        }
        memset (buff, 0, sizeof (buff));
        p_char_seen = 0;
        s_char_seen = 0;
        for (p = str; *p; p++) {
                n = 1;
                c = *p;
repeat:
                /* count the number of repeated chars */
                while (p[1] == c) {
                        p++, n++;
                }

                /* add parenthesized numbers */
                if (p[1] == '(') {
                        i = 0;
                        p += 2;
                        for (; *p == '0'; p++) {
                                ;
                        }
                        for (; *p != ')'; p++) {
                                if (!isdigit (*p)) {
                                        goto error;
                                } else {
                                        allocated++;
                                        if (allocated > 9) {
                                                goto error;
                                        }
                                        i = i * 10 + (*p - '0');
                                }
                        }
                        if (i == 0) {
                                goto error;
                        }
                        n += i - 1;
```

```
        goto repeat;
}

/* check grammar and category */
/* FIXME: need more error check */
switch (c) {
case 'A':
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        category |= PIC_ALPHABETIC;
        break;

case 'X':
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        category |= PIC_ALPHANUMERIC;
        break;

case '9':
        category |= PIC_NUMERIC;
        digits += n;
        if (v_count) {
                scale += n;
        }
        break;

case 'N':
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        category |= PIC_NATIONAL;
        break;

case 'S':
        category |= PIC_NUMERIC;
        if (category & PIC_ALPHABETIC) {
                goto error;
        }
        s_count += n;
        if (s_count > 1 || idx != 0) {
                goto error;
        }
        s_char_seen = 1;
        continue;

case ',':
case '.':
        category |= PIC_NUMERIC_EDITED;
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        if (c != current_program->decimal_point) {
                break;
        }
        /* fall through */
case 'V':
        category |= PIC_NUMERIC;
        if (category & PIC_ALPHABETIC) {
                goto error;
        }
```

```
                v_count += n;
                if (v_count > 1) {
                        goto error;
                }
                break;

        case 'P':
                category |= PIC_NUMERIC;
                if (category & PIC_ALPHABETIC) {
                        goto error;
                }
                if (p_char_seen) {
                        goto error;
                }
                at_beginning = 0;
                at_end = 0;
                switch (buffcnt) {
                case 0:
                        /* P..... */
                        at_beginning = 1;
                        break;
                case 1:
                        /* VP.... */
                        /* SP.... */
                        if (lastonechar == 'V' || lastonechar == 'S') {
                                at_beginning = 1;
                        }
                        break;
                case 2:
                        /* SVP... */
                        if (lasttwochar == 'S' && lastonechar == 'V') {
                                at_beginning = 1;
                        }
                        break;
                }
                if (p[1] == 0 || (p[1] == 'V' && p[2] == 0)) {
                        /* .....P */
                        /* ....PV */
                        at_end = 1;
                }
                if (!at_beginning && !at_end) {
                        goto error;
                }
                p_char_seen = 1;
                if (at_beginning) {
                        v_count++;      /* implicit V */
                }
                digits += n;
                if (v_count) {
                        scale += n;
                } else {
                        scale -= n;
                }
                break;

        case '0':
        case 'B':
        case '/':
                category |= PIC_EDITED;
                if (s_char_seen || p_char_seen) {
                        goto error;
                }
```

```
        break;

case '*':
case 'Z':
        category |= PIC_NUMERIC_EDITED;
        if (category & PIC_ALPHABETIC) {
                goto error;
        }
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        digits += n;
        if (v_count) {
                scale += n;
        }
        break;

case '+':
case '-':
        category |= PIC_NUMERIC_EDITED;
        if (category & PIC_ALPHABETIC) {
                goto error;
        }
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        digits += n - 1;
        s_count++;
        /* FIXME: need more check */
        break;

case 'C':
        category |= PIC_NUMERIC_EDITED;
        if (!(p[1] == 'R' && p[2] == 0)) {
                goto error;
        }
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        p++;
        s_count++;
        break;

case 'D':
        category |= PIC_NUMERIC_EDITED;
        if (!(p[1] == 'B' && p[2] == 0)) {
                goto error;
        }
        if (s_char_seen || p_char_seen) {
                goto error;
        }
        p++;
        s_count++;
        break;

default:
        if (c == current_program->currency_symbol) {
                category |= PIC_NUMERIC_EDITED;
                digits += n - 1;
                /* FIXME: need more check */
                break;
        }
```

```
                goto error;
        }

        /* calculate size */
        if (c != 'V' && c != 'P') {
                size += n;
        }
        if (c == 'C' || c == 'D' || c == 'N') {
                size += n;
        }

        /* store in the buffer */
        buff[idx++] = c;
        lasttwochar = lastonechar;
        lastonechar = c;
        memcpy (&buff[idx], (unsigned char *)&n, sizeof(int));
        idx += sizeof(int);
        ++buffcnt;
  }
  buff[idx] = 0;

  if (size == 0 && v_count) {
        goto error;
  }
  /* set picture */
  pic->orig = strdup (str);
  pic->size = size;
  pic->digits = (unsigned char)digits;
  pic->scale = (signed char)scale;
  pic->have_sign = (unsigned char)s_count;

  /* set picture category */
  switch (category) {
  case PIC_ALPHABETIC:
        pic->category = CB_CATEGORY_ALPHABETIC;
        break;
  case PIC_NUMERIC:
        pic->category = CB_CATEGORY_NUMERIC;
        if (digits > 36) {
                cb_error (_("Numeric field cannot be larger than 36 digit
s"));
        }
        break;
  case PIC_ALPHANUMERIC:
  case PIC_NATIONAL:
        pic->category = CB_CATEGORY_ALPHANUMERIC;
        break;
  case PIC_NUMERIC_EDITED:
        pic->str = cobc_malloc (idx + 1);
        memcpy (pic->str, buff, idx);
        pic->category = CB_CATEGORY_NUMERIC_EDITED;
        pic->lenstr = idx;
        break;
  case PIC_EDITED:
  case PIC_ALPHABETIC_EDITED:
  case PIC_ALPHANUMERIC_EDITED:
  case PIC_NATIONAL_EDITED:
        pic->str = cobc_malloc (idx + 1);
        memcpy (pic->str, buff, idx);
        pic->category = CB_CATEGORY_ALPHANUMERIC_EDITED;
        pic->lenstr = idx;
```

```
                break;
        default:
                goto error;
        }
        goto end;

error:
        cb_error (_("Invalid picture string - '%s'"), str);

end:
        return CB_TREE (pic);
}
```

7.16.4.54 cb_tree cb_build_ppointer (cb_tree *x*)

Definition at line 1038 of file typeck.c.

```
{
        struct cb_field *f;

        if (x == cb_error_node ||
            (CB_REFERENCE_P (x) && cb_ref (x) == cb_error_node)) {
                return cb_error_node;
        }

        if (CB_REFERENCE_P (x)) {
                f = cb_field (cb_ref(x));
                f->count++;
        }
        return cb_build_cast_ppointer (x);
}
```

7.16.4.55 struct cb_program* cb_build_program (struct cb_program * *last_program*, int *nest_level*) [read]

Definition at line 841 of file tree.c.

```
{
        struct cb_program *p;

        cb_reset_78 ();
        cb_reset_in_procedure ();
        cb_clear_real_field ();
        p = cobc_malloc (sizeof (struct cb_program));
        p->next_program = last_program;
        p->nested_level = nest_level;
        p->decimal_point = '.';
        p->currency_symbol = '$';
        p->numeric_separator = ',';
        if (nest_level) {
                p->global_file_list = last_program->global_file_list;
                p->collating_sequence = last_program->collating_sequence;
                p->function_spec_list = last_program->function_spec_list;
                p->class_spec_list = last_program->class_spec_list;
                p->interface_spec_list = last_program->interface_spec_list;
```

```
                p->program_spec_list = last_program->program_spec_list;
                p->property_spec_list = last_program->property_spec_list;
                p->alphabet_name_list = last_program->alphabet_name_list;
                p->class_name_list = last_program->class_name_list;
                p->locale_list = last_program->locale_list;
                p->symbolic_list = last_program->symbolic_list;
                p->decimal_point = last_program->decimal_point;
                p->numeric_separator = last_program->numeric_separator;
                p->currency_symbol = last_program->currency_symbol;
                p->cb_return_code = last_program->cb_return_code;
        } else {
                functions_are_all = cb_flag_functions_all;
        }
        return p;
}
```

7.16.4.56 const char* cb_build_program_id (cb_tree *name*, cb_tree *alt_name*)

Definition at line 591 of file typeck.c.

```
{
        const char      *s;

/* This needs some more thought, should we generate an entry
        point per program source name ?
        if (alt_name) {
                s = (char *)CB_LITERAL (alt_name)->data;
        } else if (CB_LITERAL_P (name)) {
                s = (char *)CB_LITERAL (name)->data;
        } else {
                s = (char *)CB_NAME (name);
        }

        if (!cb_flag_main && strcmp (s, source_name)) {
                cb_warning (_("Source name '%s' differs from PROGRAM-ID '%s'"),
                                source_name, s);
                current_program->source_name = strdup (source_name);
        }
 End comment out */

        if (alt_name) {
                current_program->orig_source_name = strdup ((char *)CB_LITERAL (a
      lt_name)->data);
                s = (char *)CB_LITERAL (alt_name)->data;
        } else if (CB_LITERAL_P (name)) {
                current_program->orig_source_name = strdup ((char *)CB_LITERAL (n
      ame)->data);
                s = cb_encode_program_id ((char *)CB_LITERAL (name)->data);
        } else {
                current_program->orig_source_name = strdup (CB_NAME (name));
                s = cb_encode_program_id (CB_NAME (name));
        }
        if (cobc_check_valid_name (current_program->orig_source_name)) {
                cb_error (_("PROGRAM-ID '%s' invalid"), current_program->
      orig_source_name);
        }
        return s;
}
```

7.16.4.57 cb_tree cb_build_reference (const char * *name*)

Definition at line 1730 of file tree.c.

```
{
        struct cb_reference *p;

        p = make_tree (CB_TAG_REFERENCE, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_reference));
        p->word = lookup_word (name);
        return CB_TREE (p);
}
```

7.16.4.58 void cb_build_registers (void)

Definition at line 494 of file typeck.c.

```
{
#if !defined(__linux__) && !defined(__CYGWIN__) && defined(HAVE_TIMEZONE)
        long    contz;
#endif
        time_t  t;
        char    buff[48];

        /* RETURN-CODE */
        if (!current_program->nested_level) {
                current_program->cb_return_code =
                        cb_build_index (cb_build_reference ("RETURN-CODE"),
                                        cb_zero, 0, NULL);
                cb_field (current_program->cb_return_code)->flag_is_global = 1;
        }

        /* SORT-RETURN */
        current_program->cb_sort_return =
                cb_build_index (cb_build_reference ("SORT-RETURN"), cb_zero, 0,
      NULL);
        cb_field (current_program->cb_sort_return)->flag_no_init = 1;

        /* NUMBER-OF-CALL-PARAMETERS */
        current_program->cb_call_params =
                cb_build_index (cb_build_reference ("NUMBER-OF-CALL-PARAMETERS"),
       cb_zero, 0, NULL);
        cb_field (current_program->cb_call_params)->flag_no_init = 1;

        /* TALLY */
        /* 01 TALLY GLOBAL PICTURE 9(9) USAGE COMP-5 VALUE ZERO. */
        /* TALLY/EXAMINE  not standard/supported */

        t = time (NULL);

        /* WHEN-COMPILED */
        memset (buff, 0, sizeof (buff));
        strftime (buff, 17, "%m/%d/%y%H.%M.%S", localtime (&t));
        cb_build_constant (cb_build_reference ("WHEN-COMPILED"),
                           cb_build_alphanumeric_literal ((ucharptr)buff, 16));

        /* FUNCTION WHEN-COMPILED */
        memset (buff, 0, sizeof (buff));
```

```
#if defined(__linux__) || defined(__CYGWIN__)
        strftime (buff, 22, "%Y%m%d%H%M%S00%z", localtime (&t));
#elif defined(HAVE_TIMEZONE)
        strftime (buff, 17, "%Y%m%d%H%M%S00", localtime (&t));
        if (timezone <= 0) {
                contz = -timezone;
                buff[16] = '+';
        } else {
                contz = timezone;
                buff[16] = '-';
        }
        sprintf (&buff[17], "%2.2ld%2.2ld", contz / 3600, contz % 60);
#else
        strftime (buff, 22, "%Y%m%d%H%M%S0000000", localtime (&t));
#endif
        cb_intr_whencomp = cb_build_alphanumeric_literal ((ucharptr)buff, 21);

        /* FUNCTION PI */
        memset (buff, 0, sizeof (buff));
        strcpy (buff, "31415926535897932384626433832795029");
        cb_intr_pi = cb_build_numeric_literal (0, (ucharptr)buff, 34);

        /* FUNCTION E */
        memset (buff, 0, sizeof (buff));
        strcpy (buff, "27182818284590452353602874713526625");
        cb_intr_e = cb_build_numeric_literal (0, (ucharptr)buff, 34);
}
```

7.16.4.59 cb_tree cb_build_replacing_all (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3823 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_all", y, x));
}
```

7.16.4.60 cb_tree cb_build_replacing_characters (cb_tree *x*, cb_tree *l*)

Definition at line 3817 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_1 ("cob_inspect_characters", x));

}
```

7.16.4.61 cb_tree cb_build_replacing_first (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3835 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_first", y, x));
}
```

7.16.4.62 cb_tree cb_build_replacing_leading (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3829 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_leading", y, x));

}
```

7.16.4.63 cb_tree cb_build_replacing_trailing (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3841 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_trailing", y, x))
      ;
}
```

7.16.4.64 cb_tree cb_build_search (int *flag_all*, cb_tree *table*, cb_tree *var*, cb_tree *end_stmt*, cb_tree *whens*)

Definition at line 2131 of file tree.c.

```
{
        struct cb_search *p;

        p = make_tree (CB_TAG_SEARCH, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_search));
        p->flag_all = flag_all;
        p->table = table;
        p->var = var;
        p->end_stmt = end_stmt;
        p->whens = whens;
        return CB_TREE (p);
}
```

7.16.4.65 cb_tree cb_build_section_name (cb_tree *name*, int *sect_or_para*)

Definition at line 653 of file typeck.c.

```
{
        cb_tree x;

        if (name == cb_error_node) {
                return cb_error_node;
        }

        if (CB_REFERENCE (name)->word->count > 0) {
                x = CB_VALUE (CB_REFERENCE (name)->word->items);
                /* Used as a non-label name or used as a section name.
```

```
                   Duplicate paragraphs are allowed if not referenced;
                   Checked in typeck.c */
                if (!CB_LABEL_P (x) || sect_or_para == 0
                    || (sect_or_para && CB_LABEL_P (x) && CB_LABEL (x)->is_sectio
      n)) {
                        redefinition_error (name);
                        return cb_error_node;
                }
        }

        return name;
}
```

7.16.4.66 struct cb_statement* cb_build_statement (const char * *name*) [read]

Definition at line 2226 of file tree.c.

```
{
        struct cb_statement *p;

        p = make_tree (CB_TAG_STATEMENT, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_statement));
        p->name = name;
        return p;
}
```

7.16.4.67 cb_tree cb_build_string (const unsigned char * *data*, size_t *size*)

Definition at line 908 of file tree.c.

```
{
        struct cb_string *p;

        p = make_tree (CB_TAG_STRING, CB_CATEGORY_ALPHANUMERIC, sizeof (struct
      cb_string));
        p->size = size;
        p->data = data;
        return CB_TREE (p);
}
```

7.16.4.68 cb_tree cb_build_sub (cb_tree *v*, cb_tree *n*, cb_tree *round_opt*)

Definition at line 2633 of file typeck.c.

```
{
        cb_tree         opt;
        struct cb_field *f;

#ifdef  COB_NON_ALIGNED
        if (CB_INDEX_P (v)) {
                return cb_build_move (cb_build_binary_op (v, '-', n), v);
        }
```

```
        if (CB_TREE_CLASS (v) == CB_CLASS_POINTER) {
                current_program->gen_ptrmanip = 1;
                return cb_build_funcall_3 ("cob_pointer_manip", v, n, cb_int1);
        }
#else
        if (CB_INDEX_P (v) || CB_TREE_CLASS (v) == CB_CLASS_POINTER) {
                return cb_build_move (cb_build_binary_op (v, '-', n), v);
        }
#endif

        if (CB_REF_OR_FIELD_P (v)) {
                f = cb_field (v);
                f->count++;
        }
        if (CB_REF_OR_FIELD_P (n)) {
                f = cb_field (n);
                f->count++;
        }
        opt = build_store_option (v, round_opt);
        if (opt == cb_int0 && cb_fits_int (n)) {
                return cb_build_optim_sub (v, n);
        }
        return cb_build_funcall_3 ("cob_sub", v, n, opt);
}
```

7.16.4.69 cb_tree cb_build_system_name (enum cb_system_name_category *category*, int *token*)

Definition at line 973 of file tree.c.

```
{
        struct cb_system_name *p;

        p = make_tree (CB_TAG_SYSTEM_NAME, CB_CATEGORY_UNKNOWN, sizeof (struct
      cb_system_name));
        p->category = category;
        p->token = token;
        return CB_TREE (p);
}
```

7.16.4.70 cb_tree cb_build_tarrying_all (void)

Definition at line 3778 of file typeck.c.

```
{
        if (inspect_data == NULL) {
                cb_error (_("Data name expected before ALL"));
        }
        inspect_func = "cob_inspect_all";
        return NULL;
}
```

7.16.4.71 cb_tree cb_build_tarrying_characters (cb_tree *l*)

Definition at line 3768 of file typeck.c.

```
{
        if (inspect_data == NULL) {
                cb_error (_("Data name expected before CHARACTERS"));
        }
        inspect_func = NULL;
        return cb_list_add (l, cb_build_funcall_1 ("cob_inspect_characters", insp
      ect_data));
}
```

7.16.4.72 cb_tree cb_build_tarrying_data (cb_tree *x*)

Definition at line 3761 of file typeck.c.

```
{
        inspect_data = x;
        return NULL;
}
```

7.16.4.73 cb_tree cb_build_tarrying_leading (void)

Definition at line 3788 of file typeck.c.

```
{
        if (inspect_data == NULL) {
                cb_error (_("Data name expected before LEADING"));
        }
        inspect_func = "cob_inspect_leading";
        return NULL;
}
```

7.16.4.74 cb_tree cb_build_tarrying_trailing (void)

Definition at line 3798 of file typeck.c.

```
{
        if (inspect_data == NULL) {
                cb_error (_("Data name expected before TRAILING"));
        }
        inspect_func = "cob_inspect_trailing";
        return NULL;
}
```

7.16.4.75 cb_tree cb_build_tarrying_value (cb_tree *x,* cb_tree *l*)

Definition at line 3808 of file typeck.c.

```
{
        if (inspect_func == NULL) {
                cb_error_x (x, _("ALL, LEADING or TRAILING expected before '%s'")
      , cb_name (x));
        }
        return cb_list_add (l, cb_build_funcall_2 (inspect_func, inspect_data, x)
      );
}
```

7.16.4.76 cb_tree cb_build_unstring_delimited (cb_tree *all,* cb_tree *value*)

Definition at line 5866 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return cb_error_node;
        }
        return cb_build_funcall_2 ("cob_unstring_delimited", value, all);
}
```

7.16.4.77 cb_tree cb_build_unstring_into (cb_tree *name,* cb_tree *delimiter,* cb_tree *count*)

Definition at line 5875 of file typeck.c.

```
{
        if (cb_validate_one (name)) {
                return cb_error_node;
        }
        if (delimiter == NULL) {
                delimiter = cb_int0;
        }
        if (count == NULL) {
                count = cb_int0;
        }
        return cb_build_funcall_3 ("cob_unstring_into", name, delimiter, count);
}
```

7.16.4.78 cb_tree cb_build_write_advancing_lines (cb_tree *pos,* cb_tree *lines*)

Definition at line 5957 of file typeck.c.

```
{
        cb_tree e;
        int     opt;
```

```
        opt = (pos == CB_BEFORE) ? COB_WRITE_BEFORE : COB_WRITE_AFTER;
        e = cb_build_binary_op (cb_int (opt | COB_WRITE_LINES), '+', lines);
        return cb_build_cast_integer (e);
}
```

7.16.4.79 cb_tree cb_build_write_advancing_mnemonic (cb_tree *pos*, cb_tree *mnemonic*)

Definition at line 5968 of file typeck.c.

```
{
        int     opt;
        int     token;

        token = CB_SYSTEM_NAME (cb_ref (mnemonic))->token;
        switch (token) {
        case CB_FEATURE_FORMFEED:
                opt = (pos == CB_BEFORE) ? COB_WRITE_BEFORE : COB_WRITE_AFTER;
                return cb_int (opt | COB_WRITE_PAGE);
        case CB_FEATURE_C01:
        case CB_FEATURE_C02:
        case CB_FEATURE_C03:
        case CB_FEATURE_C04:
        case CB_FEATURE_C05:
        case CB_FEATURE_C06:
        case CB_FEATURE_C07:
        case CB_FEATURE_C08:
        case CB_FEATURE_C09:
        case CB_FEATURE_C10:
        case CB_FEATURE_C11:
        case CB_FEATURE_C12:
                opt = (pos == CB_BEFORE) ? COB_WRITE_BEFORE : COB_WRITE_AFTER;
                return cb_int (opt | COB_WRITE_CHANNEL | COB_WRITE_PAGE | token);

        default:
                cb_error_x (mnemonic, _("Invalid mnemonic name"));
                return cb_error_node;
        }
}
```

7.16.4.80 cb_tree cb_build_write_advancing_page (cb_tree *pos*)

Definition at line 5999 of file typeck.c.

```
{
        int opt = (pos == CB_BEFORE) ? COB_WRITE_BEFORE : COB_WRITE_AFTER;

        return cb_int (opt | COB_WRITE_PAGE);
}
```

7.16.4.81 cb_tree cb_check_numeric_value (cb_tree *x*)

Definition at line 426 of file typeck.c.

```
{
        if (x == cb_error_node) {
                return cb_error_node;
        }

        if (CB_TREE_CATEGORY (x) == CB_CATEGORY_NUMERIC) {
                return x;
        }

        cb_error_x (x, _("'%s' is not a numeric value"), cb_name (x));
        return cb_error_node;
}
```

7.16.4.82 void cb_clear_real_field (void)

Definition at line 1050 of file field.c.

```
{
        last_real_field = NULL;
}
```

7.16.4.83 cb_tree cb_concat_literals (cb_tree *x1*, cb_tree *x2*)

Definition at line 1005 of file tree.c.

```
{
        unsigned char           *buff;
        cb_tree                 x;
        unsigned char           *data1;
        unsigned char           *data2;
        size_t                  size1;
        size_t                  size2;

        if (x1 == cb_error_node || x2 == cb_error_node) {
                return cb_error_node;
        }
        if (CB_LITERAL_P (x1)) {
                data1 = CB_LITERAL (x1)->data;
                size1 = CB_LITERAL (x1)->size;
        } else if (CB_CONST_P (x1)) {
                size1 = 1;
                if (x1 == cb_space) {
                        data1 = (unsigned char *)" ";
                } else if (x1 == cb_zero) {
                        data1 = (unsigned char *)"0";
                } else if (x1 == cb_quote) {
                        data1 = (unsigned char *)"\"";
                } else if (x1 == cb_norm_low) {
                        data1 = (unsigned char *)"\0";
                } else if (x1 == cb_norm_high) {
                        data1 = (unsigned char *)"\255";
                } else if (x1 == cb_null) {
                        data1 = (unsigned char *)"\0";
                } else {
                        return cb_error_node;
```

```
                }
        } else {
                return cb_error_node;
        }
        if (CB_LITERAL_P (x2)) {
                data2 = CB_LITERAL (x2)->data;
                size2 = CB_LITERAL (x2)->size;
        } else if (CB_CONST_P (x2)) {
                size2 = 1;
                if (x2 == cb_space) {
                        data2 = (unsigned char *)" ";
                } else if (x2 == cb_zero) {
                        data2 = (unsigned char *)"0";
                } else if (x2 == cb_quote) {
                        data2 = (unsigned char *)"\"";
                } else if (x2 == cb_norm_low) {
                        data2 = (unsigned char *)"\0";
                } else if (x2 == cb_norm_high) {
                        data2 = (unsigned char *)"\255";
                } else if (x2 == cb_null) {
                        data2 = (unsigned char *)"\0";
                } else {
                        return cb_error_node;
                }
        } else {
                return cb_error_node;
        }
        buff = cobc_malloc (size1 + size2 + 3);
        memcpy (buff, data1, size1);
        memcpy (buff + size1, data2, size2);
        x = cb_build_alphanumeric_literal (buff, size1 + size2);
        free (buff);
        return x;
}
```

7.16.4.84 const char* cb_define (cb_tree *name*, cb_tree *val*)

Definition at line 1769 of file tree.c.

```
{
        struct cb_word *w;

        w = CB_REFERENCE (name)->word;
        w->items = cb_list_add (w->items, val);
        w->count++;
        val->source_file = name->source_file;
        val->source_line = name->source_line;
        CB_REFERENCE (name)->value = val;
        return w->name;
}
```

7.16.4.85 void cb_define_switch_name (cb_tree *name*, cb_tree *sname*, cb_tree *flag*, cb_tree *ref*)

Definition at line 629 of file typeck.c.

```
{
        cb_tree switch_id;
        cb_tree value;

        if (name == cb_error_node) {
                return;
        }
        if (sname == cb_error_node) {
                return;
        }
        if (CB_SYSTEM_NAME (sname)->category != CB_SWITCH_NAME) {
                cb_error_x (ref, _("Switch-name is expected '%s'"), CB_NAME (ref)
      );
        } else {
                switch_id = cb_int (CB_SYSTEM_NAME (sname)->token);
                value = cb_build_funcall_1 ("cob_get_switch", switch_id);
                if (flag == cb_int0) {
                        value = cb_build_negation (value);
                }
                cb_build_constant (name, value);
        }
}
```

7.16.4.86 void cb_define_system_name (const char * *name*)

Definition at line 1783 of file tree.c.

```
{
        cb_tree x;

        x = cb_build_reference (name);
        if (CB_REFERENCE (x)->word->count == 0) {
                cb_define (x, lookup_system_name (name));
        }
}
```

7.16.4.87 void cb_emit_accept (cb_tree *var*, cb_tree *pos*, cb_tree *fgc*, cb_tree *bgc*, cb_tree *scroll*, int *dispattrs*)

Definition at line 2802 of file typeck.c.

```
{
        cb_tree line;
        cb_tree column;

        if (cb_validate_one (var)) {
                return;
        }
        if (cb_validate_one (pos)) {
                return;
        }
        if (cb_validate_one (fgc)) {
                return;
        }
        if (cb_validate_one (bgc)) {
```

```
        return;
    }
    if (cb_validate_one (scroll)) {
        return;
    }
    if (current_program->flag_screen) {
        /* Bump ref count to force CRT STATUS field generation */
        cb_field (current_program->crt_status)->count++;
        if ((CB_REF_OR_FIELD_P (var)) &&
            CB_FIELD (cb_ref (var))->storage == CB_STORAGE_SCREEN) {
            output_screen_from (CB_FIELD (cb_ref (var)), 0);
            gen_screen_ptr = 1;
            if (pos) {
                if (CB_PAIR_P (pos)) {
                    line = CB_PAIR_X (pos);
                    column = CB_PAIR_Y (pos);
                    cb_emit (cb_build_funcall_3 ("cob_screen_
accept",
                        var, line, column));
                } else {
                    cb_emit (cb_build_funcall_3 ("cob_screen_
accept",
                        var, pos, NULL));
                }
            } else {
                cb_emit (cb_build_funcall_3 ("cob_screen_accept",

                    var, NULL, NULL));
            }
            gen_screen_ptr = 0;
            output_screen_to (CB_FIELD (cb_ref (var)), 0);
        } else {
            if (pos || fgc || bgc) {
                if (!pos) {
                    cb_emit (cb_build_funcall_7 ("cob_field_a
ccept",
                        var, NULL, NULL, fgc, bgc,
                        scroll, cb_int (dispattrs)));
                } else if (CB_PAIR_P (pos)) {
                    line = CB_PAIR_X (pos);
                    column = CB_PAIR_Y (pos);
                    cb_emit (cb_build_funcall_7 ("cob_field_a
ccept",
                        var, line, column, fgc, bgc,
                        scroll, cb_int (dispattrs)));
                } else {
                    cb_emit (cb_build_funcall_7 ("cob_field_a
ccept",
                        var, pos, NULL, fgc, bgc,
                        scroll, cb_int (dispattrs)));
                }
            } else {
                cb_emit (cb_build_funcall_7 ("cob_field_accept",
                    var, NULL, NULL, fgc, bgc,
                    scroll, cb_int (dispattrs)));
            }
        }
    } else if (pos || fgc || bgc || scroll) {
        /* Bump ref count to force CRT STATUS field generation */
        cb_field (current_program->crt_status)->count++;
        if (!pos) {
            cb_emit (cb_build_funcall_7 ("cob_field_accept",
```

```
                                var, NULL, NULL, fgc, bgc, scroll,
                                cb_int (dispattrs)));
                } else if (CB_PAIR_P (pos)) {
                        line = CB_PAIR_X (pos);
                        column = CB_PAIR_Y (pos);
                        cb_emit (cb_build_funcall_7 ("cob_field_accept",
                                var, line, column, fgc, bgc, scroll,
                                cb_int (dispattrs)));
                } else {
                        cb_emit (cb_build_funcall_7 ("cob_field_accept",
                                var, pos, NULL, fgc, bgc, scroll,
                                cb_int (dispattrs)));
                }
        } else {
                cb_emit (cb_build_funcall_1 ("cob_accept", var));
        }
}
```

7.16.4.88 void cb_emit_accept_arg_number (cb_tree *var*)

Definition at line 2986 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_arg_number", var));
}
```

7.16.4.89 void cb_emit_accept_arg_value (cb_tree *var*)

Definition at line 2995 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_arg_value", var));
}
```

7.16.4.90 void cb_emit_accept_command_line (cb_tree *var*)

Definition at line 2956 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_command_line", var));
}
```

7.16.4.91 void cb_emit_accept_date (cb_tree *var*)

Definition at line 2902 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_date", var));
}
```

7.16.4.92 void cb_emit_accept_date_yyyymmdd (cb_tree *var*)

Definition at line 2911 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_date_yyyymmdd", var));
}
```

7.16.4.93 void cb_emit_accept_day (cb_tree *var*)

Definition at line 2920 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_day", var));
}
```

7.16.4.94 void cb_emit_accept_day_of_week (cb_tree *var*)

Definition at line 2938 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_day_of_week", var));
}
```

7.16.4.95 void cb_emit_accept_day_yyyyddd (cb_tree *var*)

Definition at line 2929 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_day_yyyyddd", var));
}
```

7.16.4.96 void cb_emit_accept_environment (cb_tree *var*)

Definition at line 2977 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_environment", var));
}
```

7.16.4.97 void cb_emit_accept_line_or_col (cb_tree *var,* const int *l_or_c*)

Definition at line 2893 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_2 ("cob_screen_line_col", var, cb_int (l_or_c))
      );
}
```

7.16.4.98 void cb_emit_accept_mnemonic (cb_tree *var,* cb_tree *mnemonic*)

Definition at line 3004 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        switch (CB_SYSTEM_NAME (cb_ref (mnemonic))->token) {
        case CB_DEVICE_CONSOLE:
        case CB_DEVICE_SYSIN:
                cb_emit (cb_build_funcall_1 ("cob_accept", var));
                break;
        default:
                cb_error_x (mnemonic, _("Invalid input stream '%s'"),
                            cb_name (mnemonic));
                break;
        }
}
```

7.16.4.99 void cb_emit_accept_name (cb_tree *var,* cb_tree *name*)

Definition at line 3022 of file typeck.c.

```
{
        cb_tree sys;

        if (cb_validate_one (var)) {
                return;
        }
        if (CB_REFERENCE (name)->word->count == 0) {
                sys = lookup_system_name (CB_NAME (name));

                if (sys != cb_error_node) {
                        switch (CB_SYSTEM_NAME (sys)->token) {
                        case CB_DEVICE_CONSOLE:
                        case CB_DEVICE_SYSIN:
                                cb_warning_x (name, _("'%s' undefined in SPECIAL-
      NAMES"), CB_NAME (name));
                                cb_emit (cb_build_funcall_1 ("cob_accept", var));

                                return;
                        default:
                                break;
                        }
                }
        }

        cb_error_x (name, _("'%s' undefined in SPECIAL-NAMES"), CB_NAME (name));
}
```

7.16.4.100 void cb_emit_accept_time (cb_tree *var*)

Definition at line 2947 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_time", var));
}
```

7.16.4.101 void cb_emit_allocate (cb_tree *target1,* cb_tree *target2,* cb_tree *size,* cb_tree *initialize*)

Definition at line 3053 of file typeck.c.

```
{
        cb_tree x;
        char    buff[32];

        if (cb_validate_one (target1)) {
                return;
        }
```

```
        if (cb_validate_one (target2)) {
                return;
        }
        if (cb_validate_one (size)) {
                return;
        }
        if (target1) {
                if (!(CB_REFERENCE_P(target1) &&
                      cb_field (target1)->flag_item_based)) {
                        cb_error_x (CB_TREE(current_statement),
                                _("Target of ALLOCATE is not a BASED item"));
                }
        }
        if (target2) {
                if (!(CB_REFERENCE_P(target2) &&
                      CB_TREE_CLASS (target2) == CB_CLASS_POINTER)) {
                        cb_error_x (CB_TREE(current_statement),
                                _("Target of RETURNING is not a data pointer"));
                }
        }
        if (size) {
                if (CB_TREE_CLASS (size) != CB_CLASS_NUMERIC) {
                        cb_error_x (CB_TREE(current_statement),
                                _("The CHARACTERS field of ALLOCATE must be numer
      ic"));
                }
        }
        if (target1) {
                sprintf (buff, "%d", cb_field (target1)->memory_size);
                x = cb_build_numeric_literal (0, (ucharptr)buff, 0);
                cb_emit (cb_build_funcall_3 ("cob_allocate",
                         cb_build_cast_addr_of_addr (target1), target2, x));
        } else {
                cb_emit (cb_build_funcall_3 ("cob_allocate",
                         NULL, target2, size));
        }
        if (initialize && target1) {
                current_statement->handler2 =
                        cb_build_initialize (target1, cb_true, NULL, cb_true, 0);

        }
}
```

7.16.4.102 void cb_emit_arg_number (cb_tree *value*)

Definition at line 3273 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_display_arg_number", value));
}
```

7.16.4.103 void cb_emit_arithmetic (cb_tree *vars*, int *op*, cb_tree *val*)

Definition at line 2094 of file typeck.c.

```
{
        cb_tree         l;
        struct cb_field *f;

        val = cb_check_numeric_value (val);
        if (op) {
                cb_list_map (cb_check_numeric_name, vars);
        } else {
                cb_list_map (cb_check_numeric_edited_name, vars);
        }

        if (cb_validate_one (val)) {
                return;
        }
        if (cb_validate_list (vars)) {
                return;
        }

        if (!CB_BINARY_OP_P (val)) {
                if (op == '+' || op == '-') {
                        if (CB_EXCEPTION_ENABLE (COB_EC_DATA_INCOMPATIBLE) &&
                           (CB_REF_OR_FIELD_P (val))) {
                                f = cb_field (val);
                                if (f->usage == CB_USAGE_DISPLAY ||
                                    f->usage == CB_USAGE_PACKED) {
                                        cb_emit (cb_build_funcall_2 ("cob_check_n
      umeric",
                                                                     val,
                                                                     cb_build_string0 ((
      ucharptr)(f->name))));
                                }
                        }
                        for (l = vars; l; l = CB_CHAIN (l)) {
                                if (CB_EXCEPTION_ENABLE (COB_EC_DATA_INCOMPATIBLE
      ) &&
                                   (CB_REF_OR_FIELD_P (CB_VALUE(l)))) {
                                        f = cb_field (CB_VALUE(l));
                                        if (f->usage == CB_USAGE_DISPLAY ||
                                            f->usage == CB_USAGE_PACKED) {
                                                cb_emit (cb_build_funcall_2 ("cob
      _check_numeric",
                                                                     CB_VALUE(l),
                                                                     cb_build_string0 ((
      ucharptr)(f->name))));
                                        }
                                }
                                if (op == '+') {
                                        CB_VALUE (l) = cb_build_add (CB_VALUE (l)
      , val, CB_PURPOSE (l));
                                } else {
                                        CB_VALUE (l) = cb_build_sub (CB_VALUE (l)
      , val, CB_PURPOSE (l));
                                }
                        }
                        cb_emit_list (vars);
                        return;
                }
        }

        cb_emit (build_decimal_assign (vars, op, val));
}
```

7.16.4.104 void cb_emit_call (cb_tree *prog*, cb_tree *using*, cb_tree *returning*, cb_tree *on_exception*, cb_tree *not_on_exception*)

Definition at line 3108 of file typeck.c.

```
{
        cb_tree                         l;
        cb_tree                         x;
        const struct system_table       *psyst;
        int                             is_sys_call = 0;

        if (CB_INTRINSIC_P (prog)) {
                if (CB_INTRINSIC(prog)->intr_tab->category !=
      CB_CATEGORY_ALPHANUMERIC) {
                        cb_error (_("Only alphanumeric FUNCTION types are allowed
       here"));
                        return;
                }
        }
        if (returning) {
                if (CB_TREE_CLASS(returning) != CB_CLASS_NUMERIC &&
                    CB_TREE_CLASS(returning) != CB_CLASS_POINTER) {
                        cb_error (_("Invalid RETURNING field"));
                        return;
                }
        }
        for (l = using; l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
                if (x == cb_error_node) {
                        continue;
                }
                if (CB_CONST_P (x) && x != cb_null) {
                        cb_error_x (x, _("Figurative constant invalid here"));
                }
                if ((CB_REFERENCE_P (x) && CB_FIELD_P(CB_REFERENCE(x)->value))
                     || CB_FIELD_P (x)) {
                        if (cb_field (x)->level == 88) {
                                cb_error_x (x, _("'%s' Not a data name"),
      CB_NAME (x));
                                return;
                        }
                        if (cb_warn_call_params &&
                            CB_PURPOSE_INT (l) == CB_CALL_BY_REFERENCE) {
                                if (cb_field (x)->level != 01 &&
                                    cb_field (x)->level != 77) {
                                        cb_warning_x (x, _("'%s' is not 01 or 77
      level item"), CB_NAME (x));
                                }
                        }
                }
        }

        if (CB_LITERAL_P(prog)) {
                for (psyst = (const struct system_table *)&system_tab[0]; psyst->
      syst_name; psyst++) {
                        if (!strcmp((const char *)CB_LITERAL(prog)->data,
                             (const char *)psyst->syst_name)) {
                                if (psyst->syst_params > cb_list_length (using))
      {
                                        cb_error (_("Wrong number of CALL paramet
      ers for '%s'"),
```

```
                                                        (char *)psyst->syst_name);
                                        return;
                                }
                                is_sys_call = 1;
                                break;
                        }
                }
        }

        cb_emit (cb_build_call (prog, using, on_exception, not_on_exception,
                 returning, is_sys_call));
}
```

7.16.4.105 void cb_emit_cancel (cb_tree *prog*)

Definition at line 3177 of file typeck.c.

```
{
        if (cb_validate_one (prog)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_field_cancel", prog));
}
```

7.16.4.106 void cb_emit_close (cb_tree *file*, cb_tree *opt*)

Definition at line 3190 of file typeck.c.

```
{
        if (file == cb_error_node) {
                return;
        }
        file = cb_ref (file);
        if (file == cb_error_node) {
                return;
        }
        current_statement->file = file;
        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        }
        cb_emit (cb_build_funcall_3 ("cob_close", file, opt,
                CB_FILE(file)->file_status));
}
```

7.16.4.107 void cb_emit_command_line (cb_tree *value*)

Definition at line 3282 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return;
```

```
        }
        cb_emit (cb_build_funcall_1 ("cob_display_command_line", value));
}
```

7.16.4.108 void cb_emit_commit (void)

Definition at line 3213 of file typeck.c.

```
{
        cb_emit (cb_build_funcall_0 ("cob_commit"));
}
```

7.16.4.109 void cb_emit_continue (void)

Definition at line 3223 of file typeck.c.

```
{
        cb_emit (cb_build_continue ());
}
```

7.16.4.110 void cb_emit_corresponding (cb_tree(*)(cb_tree f1, cb_tree f2, cb_tree f3) *func,* cb_tree *x1,* cb_tree *x2,* cb_tree *opt*)

Definition at line 2695 of file typeck.c.

```
{
        x1 = cb_check_group_name (x1);
        x2 = cb_check_group_name (x2);

        if (cb_validate_one (x1)) {
                return;
        }
        if (cb_validate_one (x2)) {
                return;
        }

        emit_corresponding (func, x1, x2, opt);
}
```

7.16.4.111 void cb_emit_delete (cb_tree *file*)

Definition at line 3233 of file typeck.c.

```
{
        if (file == cb_error_node) {
                return;
        }
        file = cb_ref (file);
```

```
        if (file == cb_error_node) {
                return;
        }
        current_statement->file = file;
        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        }
        cb_emit (cb_build_funcall_2 ("cob_delete", file, CB_FILE(file)->file_stat
      us));
}
```

7.16.4.112 void cb_emit_display (cb_tree *values*, cb_tree *upon*, cb_tree *no_adv*, cb_tree *pos*, cb_tree *fgc*, cb_tree *bgc*, cb_tree *scroll*, int *dispattrs*)

Definition at line 3291 of file typeck.c.

```
{
        cb_tree l;
        cb_tree x;
        cb_tree line;
        cb_tree column;
        cb_tree p;

        if (cb_validate_list (values)) {
                return;
        }
        if (cb_validate_one (pos)) {
                return;
        }
        if (cb_validate_one (fgc)) {
                return;
        }
        if (cb_validate_one (bgc)) {
                return;
        }
        if (cb_validate_one (scroll)) {
                return;
        }
        for (l = values; l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
                if (x == cb_error_node) {
                        return;
                }

                switch (CB_TREE_TAG (x)) {
                case CB_TAG_LITERAL:
                case CB_TAG_INTRINSIC:
                case CB_TAG_CONST:
                case CB_TAG_STRING:
                case CB_TAG_INTEGER:
                        break;
                case CB_TAG_REFERENCE:
                        if (!CB_FIELD_P(CB_REFERENCE(x)->value)) {
                                cb_error_x (x, _("'%s' is an invalid type for DIS
      PLAY operand"), cb_name (x));
                                return;
                        }
                        break;
```

```
		default:
			cb_error_x (x, _("Invalid type for DISPLAY operand"));
			return;
		}
	}
	if (upon == cb_error_node) {
		return;
	}

	x = CB_VALUE (values);
	if ((CB_REF_OR_FIELD_P (x)) &&
	    CB_FIELD (cb_ref (x))->storage == CB_STORAGE_SCREEN) {
		output_screen_from (CB_FIELD (cb_ref (x)), 0);
		gen_screen_ptr = 1;
		if (pos) {
			if (CB_PAIR_P (pos)) {
				line = CB_PAIR_X (pos);
				column = CB_PAIR_Y (pos);
				if (line == NULL) {
					line = cb_one;
				}
				if (column == NULL) {
					column = cb_one;
				}
				cb_emit (cb_build_funcall_3 ("cob_screen_display"
, x,
					line, column));
			} else {
				cb_emit (cb_build_funcall_3 ("cob_screen_display"
, x,
					pos, NULL));
			}
		} else {
			cb_emit (cb_build_funcall_3 ("cob_screen_display", x,
				NULL, NULL));
		}
		gen_screen_ptr = 0;
	} else if (pos || fgc || bgc || scroll || dispattrs) {
		if (!pos) {
			cb_emit (cb_build_funcall_7 ("cob_field_display",
				CB_VALUE (values), NULL, NULL, fgc, bgc,
				scroll, cb_int (dispattrs)));
		} else if (CB_PAIR_P (pos)) {
			line = CB_PAIR_X (pos);
			column = CB_PAIR_Y (pos);
			if (line == NULL) {
				line = cb_one;
			}
			if (column == NULL) {
				column = cb_one;
			}
			cb_emit (cb_build_funcall_7 ("cob_field_display",
				CB_VALUE (values), line, column, fgc, bgc,
				scroll, cb_int (dispattrs)));
		} else {
			cb_emit (cb_build_funcall_7 ("cob_field_display",
				CB_VALUE (values), pos, NULL, fgc, bgc,
				scroll, cb_int (dispattrs)));
		}
	} else {
		/* DISPLAY x ... [UPON device-name] */
		p = cb_build_funcall_3 ("cob_display", upon, no_adv, values);
```

```
                CB_FUNCALL(p)->varcnt = cb_list_length (values);
                cb_emit (p);
                for (l = values; l; l = CB_CHAIN (l)) {
                        x = CB_VALUE (l);
                        if (CB_FIELD_P (x)) {
                                CB_FIELD (cb_ref (x))->count++;
                        }
                }
        }
}
```

7.16.4.113 void cb_emit_divide (cb_tree *dividend,* cb_tree *divisor,* cb_tree *quotient,* cb_tree *remainder*)

Definition at line 3460 of file typeck.c.

```
{
        if (cb_validate_one (dividend)) {
                return;
        }
        if (cb_validate_one (divisor)) {
                return;
        }
        CB_VALUE (quotient) = cb_check_numeric_edited_name (CB_VALUE (quotient));

        CB_VALUE (remainder) = cb_check_numeric_edited_name (CB_VALUE (remainder)
      );

        if (cb_validate_one (CB_VALUE (quotient))) {
                return;
        }
        if (cb_validate_one (CB_VALUE (remainder))) {
                return;
        }

        cb_emit (cb_build_funcall_4 ("cob_div_quotient", dividend, divisor,
                                     CB_VALUE (quotient),
                                     build_store_option (CB_VALUE (quotient),
      CB_PURPOSE (quotient))));
        cb_emit (cb_build_funcall_2 ("cob_div_remainder", CB_VALUE (remainder),
                                     build_store_option (CB_VALUE (remainder),
      cb_int0)));
}
```

7.16.4.114 void cb_emit_env_name (cb_tree *value*)

Definition at line 3255 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_display_environment", value));
}
```

7.16.4.115 void cb_emit_env_value (cb_tree *value*)

Definition at line 3264 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_display_env_value", value));
}
```

7.16.4.116 void cb_emit_evaluate (cb_tree *subject_list*, cb_tree *case_list*)

Definition at line 3601 of file typeck.c.

```
{
        cb_emit (build_evaluate (subject_list, case_list));
}
```

7.16.4.117 void cb_emit_exit (size_t *goback*)

Definition at line 3675 of file typeck.c.

```
{
        if (goback) {
                cb_emit (cb_build_goto (cb_int1, NULL));
        } else {
                cb_emit (cb_build_goto (NULL, NULL));
        }
}
```

7.16.4.118 void cb_emit_free (cb_tree *vars*)

Definition at line 3611 of file typeck.c.

```
{
        cb_tree         l;
        struct cb_field *f;
        int             i;

        if (cb_validate_list (vars)) {
                return;
        }
        for (l = vars, i = 1; l; l = CB_CHAIN (l), i++) {
                if (CB_TREE_CLASS (CB_VALUE (l)) == CB_CLASS_POINTER) {
                        if (CB_CAST_P (CB_VALUE (l))) {
                                f = cb_field (CB_CAST (CB_VALUE(l))->val);
                                if (!f->flag_item_based) {
                                        cb_error_x (CB_TREE (current_statement),
                                                _("Target %d of FREE, a data addr
```

```
      ess identifier, must address a BASED data item"), i);
                                }
                                cb_emit (cb_build_funcall_2 ("cob_free_alloc",
                                        cb_build_cast_address (CB_VALUE (l)),
      NULL));
                        } else {
                                cb_emit (cb_build_funcall_2 ("cob_free_alloc",
                                        NULL, cb_build_cast_address (CB_VALUE (l)
      )));
                        }
                } else if (CB_REF_OR_FIELD_P (CB_VALUE (l))) {
                                f = cb_field (CB_VALUE (l));
                                if (!f->flag_item_based) {
                                        cb_error_x (CB_TREE (current_statement),
                                                _("Target %d of FREE, a data addr
      ess identifier, must address a BASED data item"), i);
                                }
                                cb_emit (cb_build_funcall_2 ("cob_free_alloc",
                                        cb_build_cast_addr_of_addr (CB_VALUE (l))
      , NULL));
                } else {
                        cb_error_x (CB_TREE (current_statement),
                                _("Target %d of FREE must be a data pointer"), i)
      ;
                }
          }
}
```

7.16.4.119 void cb_emit_get_environment (cb_tree *envvar,* cb_tree *envval*)

Definition at line 2965 of file typeck.c.

```
{
        if (cb_validate_one (envvar)) {
                return;
        }
        if (cb_validate_one (envval)) {
                return;
        }
        cb_emit (cb_build_funcall_2 ("cob_get_environment", envvar, envval));
}
```

7.16.4.120 void cb_emit_goto (cb_tree *target,* cb_tree *depending*)

Definition at line 3654 of file typeck.c.

```
{
        if (target == cb_error_node) {
                return;
        }
        if (depending) {
                /* GO TO procedure-name ... DEPENDING ON identifier */
                cb_emit (cb_build_goto (target, depending));
        } else {
                /* GO TO procedure-name */
```

```
                if (target == NULL) {
                        cb_verify (cb_goto_statement_without_name, "GO TO without
       procedure-name");
                } else if (CB_CHAIN (target)) {
                        cb_error (_("GO TO with multiple procedure-names"));
                } else {
                        cb_emit (cb_build_goto (CB_VALUE (target), NULL));
                }
        }
}
```

7.16.4.121 void cb_emit_if (cb_tree *cond*, cb_tree *stmt1*, cb_tree *stmt2*)

Definition at line 3689 of file typeck.c.

```
{
        cb_emit (cb_build_if (cond, stmt1, stmt2));
}
```

7.16.4.122 void cb_emit_initialize (cb_tree *vars*, cb_tree *fillinit*, cb_tree *value*, cb_tree *replacing*, cb_tree *def*)

Definition at line 3699 of file typeck.c.

```
{
        cb_tree l;
        int fill_init = 1;

        if (cb_validate_list (vars)) {
                return;
        }
        if (value == NULL && replacing == NULL) {
                def = cb_true;
        }
        if (fillinit == cb_true) {
                fill_init = 0;
        }
        for (l = vars; l; l = CB_CHAIN (l)) {
                cb_emit (cb_build_initialize (CB_VALUE (l), value, replacing, def
      , fill_init));
        }
}
```

7.16.4.123 void cb_emit_inspect (cb_tree *var*, cb_tree *body*, cb_tree *replacing*, int *replconv*)

Definition at line 3723 of file typeck.c.

```
{
        switch (CB_TREE_TAG(var)) {
        case CB_TAG_REFERENCE:
```

```
                break;
        case CB_TAG_INTRINSIC:
                switch (CB_TREE_CATEGORY(var)) {
                case CB_CATEGORY_ALPHABETIC:
                case CB_CATEGORY_ALPHANUMERIC:
                case CB_CATEGORY_NATIONAL:
                        break;
                default:
                        cb_error (_("Invalid target for INSPECT"));
                        return;
                }
                break;
        case CB_TAG_LITERAL:
                break;
        default:
                cb_error (_("Invalid target for REPLACING/CONVERTING"));
                return;
        }
        if (replconv && sending_id) {
                cb_error (_("Invalid target for REPLACING/CONVERTING"));
        }
        cb_emit (cb_build_funcall_2 ("cob_inspect_init", var, replacing));
        cb_emit_list (body);
        cb_emit (cb_build_funcall_0 ("cob_inspect_finish"));
}
```

7.16.4.124 void cb_emit_move (cb_tree *src*, cb_tree *dsts*)

Definition at line 5041 of file typeck.c.

```
{
        cb_tree l;

        if (cb_validate_one (src)) {
                return;
        }
        if (cb_validate_list (dsts)) {
                return;
        }

        for (l = dsts; l; l = CB_CHAIN (l)) {
                cb_emit (cb_build_move (src, CB_VALUE (l)));
        }
}
```

7.16.4.125 void cb_emit_move_corresponding (cb_tree *x1*, cb_tree *x2*)

Definition at line 2738 of file typeck.c.

```
{
        cb_tree         l;
        cb_tree         v;

        x1 = cb_check_group_name (x1);
        if (cb_validate_one (x1)) {
```

```
                return;
        }
        for (l = x2; l; l = CB_CHAIN(l)) {
                v = CB_VALUE(l);
                v = cb_check_group_name (v);
                if (cb_validate_one (v)) {
                        return;
                }
                emit_move_corresponding (x1, v);
        }
}
```

7.16.4.126 void cb_emit_open (cb_tree *file*, cb_tree *mode*, cb_tree *sharing*)

Definition at line 5062 of file typeck.c.

```
{
        if (file == cb_error_node) {
                return;
        }
        file = cb_ref (file);
        if (file == cb_error_node) {
                return;
        }
        current_statement->file = file;

        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        }
        if (sharing == NULL) {
                sharing = CB_FILE (file)->sharing ? CB_FILE (file)->sharing :
      cb_int0;
        }

        /* READ ONLY */
        if (sharing == cb_int0 && CB_INTEGER (mode)->val != COB_OPEN_INPUT) {
                sharing = cb_int1;
        }

        cb_emit (cb_build_funcall_4 ("cob_open", file, mode,
                 sharing, CB_FILE(file)->file_status));
}
```

7.16.4.127 void cb_emit_perform (cb_tree *perform*, cb_tree *body*)

Definition at line 5095 of file typeck.c.

```
{
        if (perform == cb_error_node) {
                return;
        }
        CB_PERFORM (perform)->body = body;
        cb_emit (perform);
}
```

7.16.4.128 void cb_emit_read (cb_tree *ref*, cb_tree *next*, cb_tree *into*, cb_tree *key*, cb_tree *lock_opts*)

Definition at line 5170 of file typeck.c.

```
{
        int     read_opts = 0;
        cb_tree file;
        cb_tree rec;

        if (lock_opts == cb_int1) {
                read_opts = COB_READ_LOCK;
        } else if (lock_opts == cb_int2) {
                read_opts = COB_READ_NO_LOCK;
        } else if (lock_opts == cb_int3) {
                read_opts = COB_READ_IGNORE_LOCK;
        } else if (lock_opts == cb_int4) {
                read_opts = COB_READ_WAIT_LOCK;
        }
        if (ref == cb_error_node) {
                return;
        }
        file = cb_ref (ref);
        if (file == cb_error_node) {
                return;
        }
        rec = cb_build_field_reference (CB_FILE (file)->record, ref);
        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        }
        if (next == cb_int1 || next == cb_int2 ||
            CB_FILE (file)->access_mode == COB_ACCESS_SEQUENTIAL) {
                /* READ NEXT/PREVIOUS */
                if (next == cb_int2) {
                        if (CB_FILE (file)->organization != COB_ORG_INDEXED) {
                                cb_error_x (CB_TREE (current_statement),
                                _("READ PREVIOUS only allowed for INDEXED SEQUENT
      IAL files"));
                        }
                        read_opts |= COB_READ_PREVIOUS;
                } else {
                        read_opts |= COB_READ_NEXT;
                }
                if (key) {
                        cb_warning (_("KEY ignored with sequential READ"));
                }
                cb_emit (cb_build_funcall_4 ("cob_read", file, cb_int0,
                         CB_FILE(file)->file_status,
                         cb_int (read_opts)));
        } else {
                /* READ */
                cb_emit (cb_build_funcall_4 ("cob_read",
                         file, key ? key : CB_FILE (file)->key,
                         CB_FILE(file)->file_status, cb_int (read_opts)));
        }
        if (into) {
                current_statement->handler3 = cb_build_move (rec, into);
        }
        current_statement->file = file;
}
```

7.16.4.129 void cb_emit_release (cb_tree *ref*, cb_tree *from*)

Definition at line 5278 of file typeck.c.

```
{
        struct cb_field *f;
        cb_tree         file;

        if (record == cb_error_node) {
                return;
        }
        if (from == cb_error_node) {
                return;
        }
        if (cb_ref (record) == cb_error_node) {
                return;
        }
        if (!CB_REF_OR_FIELD_P (cb_ref (record))) {
                cb_error_x (CB_TREE (current_statement),
                        _("RELEASE requires a record name as subject"));
                return;
        }
        if (cb_field (record)->storage != CB_STORAGE_FILE) {
                cb_error_x (CB_TREE (current_statement),
                        _("RELEASE subject does not refer to a record name"));
                return;
        }
        f = CB_FIELD (cb_ref (record));
        file = CB_TREE (f->file);
        if (CB_FILE (file)->organization != COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                        _("RELEASE not allowed on this record item"));
                return;
        }
        current_statement->file = file;
        if (from) {
                cb_emit (cb_build_move (from, record));
        }
        cb_emit (cb_build_funcall_1 ("cob_file_release", file));
}
```

7.16.4.130 void cb_emit_return (cb_tree *ref*, cb_tree *into*)

Definition at line 5321 of file typeck.c.

```
{
        cb_tree file;
        cb_tree rec;

        if (ref == cb_error_node) {
                return;
        }
        if (into == cb_error_node) {
                return;
        }
        file = cb_ref (ref);
        if (file == cb_error_node) {
                return;
```

```
        }
        rec = cb_build_field_reference (CB_FILE (file)->record, ref);
        cb_emit (cb_build_funcall_1 ("cob_file_return", file));
        if (into) {
                current_statement->handler3 = cb_build_move (rec, into);
        }
        current_statement->file = file;
}
```

7.16.4.131 void cb_emit_rewrite (cb_tree *record,* cb_tree *from,* cb_tree *lockopt*)

Definition at line 5232 of file typeck.c.

```
{
        cb_tree file;
        int     opts = 0;

        if (record == cb_error_node || cb_ref (record) == cb_error_node) {
                return;
        }
        if (!CB_REF_OR_FIELD_P (cb_ref (record))) {
                cb_error_x (CB_TREE (current_statement),
                        _("REWRITE requires a record name as subject"));
                return;
        }
        if (cb_field (record)->storage != CB_STORAGE_FILE) {
                cb_error_x (CB_TREE (current_statement),
                        _("REWRITE subject does not refer to a record name"));
                return;
        }
        file = CB_TREE (CB_FIELD (cb_ref (record))->file);
        current_statement->file = file;
        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        } else if (current_statement->handler_id == COB_EC_I_O_INVALID_KEY &&
                   (CB_FILE(file)->organization != COB_ORG_RELATIVE &&
                    CB_FILE(file)->organization != COB_ORG_INDEXED)) {
                        cb_error_x (CB_TREE(current_statement),
                        _("INVALID KEY clause invalid with this file type"));
        } else if ((CB_FILE (file)->lock_mode & COB_LOCK_AUTOMATIC) && lockopt) {

                cb_error_x (CB_TREE (current_statement),
                _("LOCK clause invalid with file LOCK AUTOMATIC"));
        } else if (lockopt == cb_int1) {
                opts = COB_WRITE_LOCK;
        }
        if (from) {
                cb_emit (cb_build_move (from, record));
        }
        cb_emit (cb_build_funcall_4 ("cob_rewrite", file, record,
                        cb_int (opts), CB_FILE(file)->file_status));
}
```

7.16.4.132 void cb_emit_rollback (void)

Definition at line 5349 of file typeck.c.

```
{
        cb_emit (cb_build_funcall_0 ("cob_rollback"));
}
```

7.16.4.133 void cb_emit_search (cb_tree *table*, cb_tree *varying*, cb_tree *at_end*, cb_tree *whens*)

Definition at line 5427 of file typeck.c.

```
{
        if (cb_validate_one (table)) {
                return;
        }
        if (cb_validate_one (varying)) {
                return;
        }
        if (table == cb_error_node) {
                return;
        }
        cb_emit (cb_build_search (0, table, varying, at_end, whens));
}
```

7.16.4.134 void cb_emit_search_all (cb_tree *table*, cb_tree *at_end*, cb_tree *when*, cb_tree *stmts*)

Definition at line 5442 of file typeck.c.

```
{
        if (cb_validate_one (table)) {
                return;
        }
        if (table == cb_error_node) {
                return;
        }
        cb_emit (cb_build_search (1, table, NULL, at_end,
                                  cb_build_if (cb_build_search_all (table, when),
       stmts, NULL)));
}
```

7.16.4.135 void cb_emit_set_false (cb_tree *l*)

Definition at line 5604 of file typeck.c.

```
{
        cb_tree         x;
        struct cb_field *f;
        cb_tree         ref;
        cb_tree         val;

        for (; l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
```

```
                if (x == cb_error_node) {
                        return;
                }
                if (!(CB_REFERENCE_P (x) && CB_FIELD_P(CB_REFERENCE(x)->value))
                     && !CB_FIELD_P (x)) {
                        cb_error_x (x, _("Invalid SET statement"));
                        return;
                }
                f = cb_field (x);
                if (f->level != 88) {
                        cb_error_x (x, _("Invalid SET statement"));
                        return;
                }
                if (!f->false_88) {
                        cb_error_x (x, _("Field does not have FALSE clause"));
                        return;
                }
                ref = cb_build_field_reference (f->parent, x);
                val = CB_VALUE (f->false_88);
                if (CB_PAIR_P (val)) {
                        val = CB_PAIR_X (val);
                }
                cb_emit (cb_build_move (val, ref));
        }
}
```

7.16.4.136 void cb_emit_set_on_off (cb_tree *l*, cb_tree *flag*)

Definition at line 5558 of file typeck.c.

```
{
        struct cb_system_name *s;

        if (cb_validate_list (l)) {
                return;
        }
        for (; l; l = CB_CHAIN (l)) {
                s = CB_SYSTEM_NAME (cb_ref (CB_VALUE (l)));
                cb_emit (cb_build_funcall_2 ("cob_set_switch", cb_int (s->token),
       flag));
        }
}
```

7.16.4.137 void cb_emit_set_to (cb_tree *l*, cb_tree *x*)

Definition at line 5465 of file typeck.c.

```
{
        cb_tree         l;
        cb_tree         v;
        struct cb_cast  *p;
#if 0
        enum cb_class class = CB_CLASS_UNKNOWN;
#endif
```

```
        if (cb_validate_one (x)) {
                return;
        }
        if (cb_validate_list (vars)) {
                return;
        }

#if 0
        /* determine the class of targets */
        for (l = vars; l; l = CB_CHAIN (l)) {
                if (CB_TREE_CLASS (CB_VALUE (l)) != CB_CLASS_UNKNOWN) {
                        if (class == CB_CLASS_UNKNOWN) {
                                class = CB_TREE_CLASS (CB_VALUE (l));
                        } else if (class != CB_TREE_CLASS (CB_VALUE (l))) {
                                break;
                        }
                }
        }
        if (l || (class != CB_CLASS_INDEX && class != CB_CLASS_POINTER)) {
                cb_error_x (CB_TREE (current_statement),
                            _("The targets of SET must be either indexes or point
      ers"));
                return;
        }
#endif

        if (CB_CAST_P (x)) {
                p = CB_CAST (x);
                if (p->type == CB_CAST_PROGRAM_POINTER) {
                        for (l = vars; l; l = CB_CHAIN (l)) {
                                v = CB_VALUE (l);
                                if (!CB_REFERENCE_P (v)) {
                                        cb_error_x (CB_TREE (current_statement),
                                        _("SET targets must be PROGRAM-POINTER"))
      ;
                                        CB_VALUE (l) = cb_error_node;
                                } else if (CB_FIELD(cb_ref(v))->usage !=
      CB_USAGE_PROGRAM_POINTER) {
                                        cb_error_x (CB_TREE (current_statement),
                                        _("SET targets must be PROGRAM-POINTER"))
      ;
                                        CB_VALUE (l) = cb_error_node;
                                }
                        }
                }
        }
        /* validate the targets */
        for (l = vars; l; l = CB_CHAIN (l)) {
                v = CB_VALUE (l);
                if (CB_CAST_P (v)) {
                        p = CB_CAST (v);
                        if (p->type == CB_CAST_ADDRESS
                            && !CB_FIELD (cb_ref (p->val))->flag_item_based
                            && CB_FIELD (cb_ref (p->val))->storage !=
      CB_STORAGE_LINKAGE) {
                                cb_error_x (p->val, _("The address of '%s' cannot
       be changed"),
                                            cb_name (p->val));
                                CB_VALUE (l) = cb_error_node;
                        }
                }
        }
```

```
        if (cb_validate_list (vars)) {
                return;
        }

        for (l = vars; l; l = CB_CHAIN (l)) {
                cb_emit (cb_build_move (x, CB_VALUE (l)));
        }
}
```

7.16.4.138 void cb_emit_set_true (cb_tree *l*)

Definition at line 5572 of file typeck.c.

```
{
        cb_tree         x;
        struct cb_field *f;
        cb_tree         ref;
        cb_tree         val;

        for (; l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
                if (x == cb_error_node) {
                        return;
                }
                if (!(CB_REFERENCE_P (x) && CB_FIELD_P(CB_REFERENCE(x)->value))
                     && !CB_FIELD_P (x)) {
                        cb_error_x (x, _("Invalid SET statement"));
                        return;
                }
                f = cb_field (x);
                if (f->level != 88) {
                        cb_error_x (x, _("Invalid SET statement"));
                        return;
                }
                ref = cb_build_field_reference (f->parent, x);
                val = CB_VALUE (f->values);
                if (CB_PAIR_P (val)) {
                        val = CB_PAIR_X (val);
                }
                cb_emit (cb_build_move (val, ref));
        }
}
```

7.16.4.139 void cb_emit_set_up_down (cb_tree *l*, cb_tree *flag*, cb_tree *x*)

Definition at line 5540 of file typeck.c.

```
{
        if (cb_validate_one (x)) {
                return;
        }
        if (cb_validate_list (l)) {
                return;
        }
        for (; l; l = CB_CHAIN (l)) {
```

```
                if (flag == cb_int0) {
                        cb_emit (cb_build_add (CB_VALUE (l), x, cb_int0));
                } else {
                        cb_emit (cb_build_sub (CB_VALUE (l), x, cb_int0));
                }
        }
}
```

7.16.4.140 void cb_emit_setenv (cb_tree *x*, cb_tree *y*)

Definition at line 5459 of file typeck.c.

```
{
        cb_emit (cb_build_funcall_2 ("cob_set_environment", x, y));
}
```

7.16.4.141 void cb_emit_sort_finish (cb_tree *file*)

Definition at line 5741 of file typeck.c.

```
{
        if (CB_FILE_P (cb_ref (file))) {
                cb_emit (cb_build_funcall_1 ("cob_file_sort_close", cb_ref (file)
      ));
        }
}
```

7.16.4.142 void cb_emit_sort_giving (cb_tree *file*, cb_tree *l*)

Definition at line 5715 of file typeck.c.

```
{
        cb_tree         p;
        int             listlen;

        if (cb_validate_list (l)) {
                return;
        }
        for (p = l; p; p = CB_CHAIN (p)) {
                if (CB_FILE (cb_ref(CB_VALUE(p)))->organization == COB_ORG_SORT)
      {
                        cb_error (_("Invalid SORT GIVING parameter"));
                }
        }
        listlen = cb_list_length (l);
        p = cb_build_funcall_2 ("cob_file_sort_giving", cb_ref (file), l);
        CB_FUNCALL(p)->varcnt = listlen;
        cb_emit (p);
}
```

7.16.4.143 void cb_emit_sort_init (cb_tree *name*, cb_tree *keys*, cb_tree *col*)

Definition at line 5644 of file typeck.c.

```
{
        cb_tree         l;
        struct cb_field *f;

        if (cb_validate_list (keys)) {
                return;
        }
        for (l = keys; l; l = CB_CHAIN (l)) {
                if (CB_VALUE (l) == NULL) {
                        CB_VALUE (l) = name;
                }
                cb_ref (CB_VALUE (l));
        }

        if (CB_FILE_P (cb_ref (name))) {
                if (CB_FILE (cb_ref (name))->organization != COB_ORG_SORT) {
                        cb_error_x (name, _("Invalid SORT filename"));
                }
                cb_field (current_program->cb_sort_return)->count++;
                cb_emit (cb_build_funcall_5 ("cob_file_sort_init", cb_ref (name),

                                             cb_int (cb_list_length (keys)), col,

                                             cb_build_cast_address (
      current_program->cb_sort_return),
                                             CB_FILE(cb_ref (name))->file_status)
      );
                for (l = keys; l; l = CB_CHAIN (l)) {
                        cb_emit (cb_build_funcall_4 ("cob_file_sort_init_key",
      cb_ref (name),
                                        CB_PURPOSE (l),
                                        CB_VALUE (l),
                                        cb_int (cb_field (CB_VALUE(l))->offset)))
      ;
                }
        } else {
                f = CB_FIELD (cb_ref (name));
                if (keys == NULL) {
                        cb_error_x (name, _("Table sort without keys not implemen
      ted yet"));
                }
                cb_emit (cb_build_funcall_2 ("cob_table_sort_init", cb_int (
      cb_list_length (keys)), col));
                for (l = keys; l; l = CB_CHAIN (l)) {
                        cb_emit (cb_build_funcall_3 ("cob_table_sort_init_key",
                                        CB_PURPOSE (l),
                                        CB_VALUE (l),
                                        cb_int (cb_field (CB_VALUE(l))->offset)))
      ;
                }
                cb_emit (cb_build_funcall_2 ("cob_table_sort", name,
                                             (f->occurs_depending
                                              ? cb_build_cast_integer (f->
      occurs_depending)
                                              : cb_int (f->occurs_max))));
        }
}
```

7.16.4.144 void cb_emit_sort_input (cb_tree *proc*)

Definition at line 5709 of file typeck.c.

```
{
        cb_emit (cb_build_perform_once (proc));
}
```

7.16.4.145 void cb_emit_sort_output (cb_tree *proc*)

Definition at line 5735 of file typeck.c.

```
{
        cb_emit (cb_build_perform_once (proc));
}
```

7.16.4.146 void cb_emit_sort_using (cb_tree *file,* cb_tree *l*)

Definition at line 5694 of file typeck.c.

```
{
        if (cb_validate_list (l)) {
                return;
        }
        for (; l; l = CB_CHAIN (l)) {
                if (CB_FILE (cb_ref(CB_VALUE(l)))->organization == COB_ORG_SORT)
      {
                        cb_error (_("Invalid SORT USING parameter"));
                }
                cb_emit (cb_build_funcall_2 ("cob_file_sort_using",
                        cb_ref (file), cb_ref (CB_VALUE (l))));
        }
}
```

7.16.4.147 void cb_emit_start (cb_tree *file,* cb_tree *op,* cb_tree *key*)

Definition at line 5753 of file typeck.c.

```
{
        if (cb_validate_one (key)) {
                return;
        }
        if (file != cb_error_node) {
                current_statement->file = cb_ref (file);
                cb_emit (cb_build_funcall_4 ("cob_start", cb_ref (file), op,
                                             key ? key : CB_FILE (cb_ref (file))-
      >key,
                                                CB_FILE(cb_ref(file))->file_statu
      s));
        }
}
```

7.16.4.148 void cb_emit_stop_run (cb_tree *x*)

Definition at line 5771 of file typeck.c.

```
{
        cb_emit (cb_build_funcall_1 ("cob_stop_run", cb_build_cast_integer (x)));

}
```

7.16.4.149 void cb_emit_string (cb_tree *items*, cb_tree *into*, cb_tree *pointer*)

Definition at line 5781 of file typeck.c.

```
{
        cb_tree start;
        cb_tree l;
        cb_tree end;
        cb_tree dlm;

        if (cb_validate_one (into)) {
                return;
        }
        if (cb_validate_one (pointer)) {
                return;
        }
        start = items;
        cb_emit (cb_build_funcall_2 ("cob_string_init", into, pointer));
        while (start) {

                /* find DELIMITED item */
                for (end = start; end; end = CB_CHAIN (end)) {
                        if (CB_PAIR_P (CB_VALUE (end))) {
                                break;
                        }
                }

                /* cob_string_delimited */
                dlm = end ? CB_PAIR_X (CB_VALUE (end)) : cb_int0;
                cb_emit (cb_build_funcall_1 ("cob_string_delimited", dlm));

                /* cob_string_append */
                for (l = start; l != end; l = CB_CHAIN (l)) {
                        cb_emit (cb_build_funcall_1 ("cob_string_append",
      CB_VALUE (l)));
                }

                start = end ? CB_CHAIN (end) : NULL;
        }
        cb_emit (cb_build_funcall_0 ("cob_string_finish"));
}
```

7.16.4.150 void cb_emit_unlock (cb_tree *ref*)

Definition at line 5824 of file typeck.c.

```
{
        cb_tree file;

        if (ref != cb_error_node) {
                file = cb_ref (ref);
                cb_emit (cb_build_funcall_2 ("cob_unlock_file",
                         file, CB_FILE(file)->file_status));
                current_statement->file = file;
        }
}
```

7.16.4.151 void cb_emit_unstring (cb_tree *name,* cb_tree *delimited,* cb_tree *into,* cb_tree *pointer,* cb_tree *tallying*)

Definition at line 5841 of file typeck.c.

```
{
        if (cb_validate_one (name)) {
                return;
        }
        if (cb_validate_one (tallying)) {
                return;
        }
        if (cb_validate_list (delimited)) {
                return;
        }
        if (cb_validate_list (into)) {
                return;
        }
        cb_emit (cb_build_funcall_3 ("cob_unstring_init", name, pointer,
                cb_int (cb_list_length (delimited))));
        cb_emit_list (delimited);
        cb_emit_list (into);
        if (tallying) {
                cb_emit (cb_build_funcall_1 ("cob_unstring_tallying", tallying));

        }
        cb_emit (cb_build_funcall_0 ("cob_unstring_finish"));
}
```

7.16.4.152 void cb_emit_write (cb_tree *record,* cb_tree *from,* cb_tree *opt,* cb_tree *lockopt*)

Definition at line 5894 of file typeck.c.

```
{
        cb_tree         file;
        int             val;

        if (record != cb_error_node && cb_ref (record) != cb_error_node) {
                if (!CB_REF_OR_FIELD_P (cb_ref (record))) {
                        cb_error_x (CB_TREE (current_statement),
                                _("WRITE requires a record name as subject"));
                        return;
                }
```

```
                if (cb_field (record)->storage != CB_STORAGE_FILE) {
                        cb_error_x (CB_TREE (current_statement),
                                _("WRITE subject does not refer to a record name"
      ));
                        return;
                }
                file = CB_TREE (CB_FIELD (cb_ref (record))->file);
                current_statement->file = file;
                if (CB_FILE (file)->organization == COB_ORG_SORT) {
                        cb_error_x (CB_TREE (current_statement),
                        _("Operation not allowed on SORT files"));
                } else if (current_statement->handler_id == COB_EC_I_O_INVALID_KE
      Y &&
                          (CB_FILE(file)->organization != COB_ORG_RELATIVE &&
                           CB_FILE(file)->organization != COB_ORG_INDEXED)) {
                                cb_error_x (CB_TREE(current_statement),
                                _("INVALID KEY clause invalid with this file type
      "));
                } else if (lockopt) {
                        if ((CB_FILE (file)->lock_mode & COB_LOCK_AUTOMATIC)) {
                                cb_error_x (CB_TREE (current_statement),
                                _("LOCK clause invalid with file LOCK AUTOMATIC")
      );
                        } else if (opt != cb_int0) {
                                cb_error_x (CB_TREE (current_statement),
                                _("LOCK clause invalid here"));
                        } else if (lockopt == cb_int1) {
                                opt = cb_int (COB_WRITE_LOCK);
                        }
                }
                if (from) {
                        cb_emit (cb_build_move (from, record));
                }
                if (CB_FILE (file)->organization == COB_ORG_LINE_SEQUENTIAL &&
                    opt == cb_int0) {
                        opt = cb_int (COB_WRITE_BEFORE | COB_WRITE_LINES | 1);
                }
                /* RXW - This is horrible */
                if (current_statement->handler_id == COB_EC_I_O_EOP &&
                    current_statement->handler1) {
                        if (CB_CAST_P(opt)) {
                                val = CB_INTEGER(CB_BINARY_OP(CB_CAST(opt)->val)-
      >x)->val;
                                val |= COB_WRITE_EOP;
                                CB_BINARY_OP(CB_CAST(opt)->val)->x = cb_int (val)
      ;
                        } else {
                                val = CB_INTEGER(opt)->val;
                                val |= COB_WRITE_EOP;
                                opt = cb_int (val);
                        }
                }
                cb_emit (cb_build_funcall_4 ("cob_write", file, record, opt,
                                             CB_FILE(file)->file_status));
          }
}
```

7.16.4.153 char* cb_encode_program_id (const char * *name*)

Definition at line 563 of file typeck.c.

```
{
        unsigned char           *p;
        const unsigned char     *s;
        unsigned char           buff[COB_SMALL_BUFF];

        p = buff;
        s = (const unsigned char *)name;
        /* encode the initial digit */
        if (isdigit (*s)) {
                p += sprintf ((char *)p, "_%02X", *s++);
        }
        /* encode invalid letters */
        for (; *s; s++) {
                if (isalnum (*s) || *s == '_') {
                        *p++ = *s;
                } else if (*s == '-') {
                        *p++ = '_';
                        *p++ = '_';
                } else {
                        p += sprintf ((char *)p, "_%02X", *s);
                }
        }
        *p = 0;
        return strdup ((char *)buff);
}
```

7.16.4.154 void cb_error_x (cb_tree *x*, const char * *fmt*, ...)

Definition at line 110 of file error.c.

```
{
        va_list ap;

        va_start (ap, fmt);
        print_error ((char *)(x->source_file), x->source_line, "Error: ", fmt, ap
      );
        va_end (ap);

        errorcount++;
}
```

7.16.4.155 struct cb_field* cb_field (cb_tree *x*) [read]

Definition at line 1458 of file tree.c.

```
{
        if (CB_REFERENCE_P (x)) {
                return CB_FIELD (cb_ref (x));
        } else {
                return CB_FIELD (x);
        }
}
```

7.16.4.156 struct cb_field* cb_field_add (struct cb_field * f, struct cb_field * p) [read]

Definition at line 1468 of file tree.c.

```
{
        struct cb_field *t;

        if (f == NULL) {
                return p;
        }
        for (t = f; t->sister; t = t->sister) {
                ;
        }
        t->sister = p;
        return f;
}
```

7.16.4.157 struct cb_field* cb_field_founder (struct cb_field * f) [read]

Definition at line 1521 of file tree.c.

```
{
        while (f->parent) {
                f = f->parent;
        }
        return f;
}
```

7.16.4.158 int cb_field_size (cb_tree x)

Definition at line 1483 of file tree.c.

```
{
        struct cb_reference     *r;
        struct cb_field         *f;

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_LITERAL:
                return CB_LITERAL (x)->size;
        case CB_TAG_FIELD:
                return CB_FIELD (x)->size;
        case CB_TAG_REFERENCE:
                r = CB_REFERENCE (x);
                f = CB_FIELD (r->value);

                if (r->length) {
                        if (CB_LITERAL_P (r->length)) {
                                return cb_get_int (r->length);
                        } else {
                                return -1;
                        }
                } else if (r->offset) {
                        if (CB_LITERAL_P (r->offset)) {
```

```
                                return f->size - cb_get_int (r->offset) + 1;
                        } else {
                                return -1;
                        }
                } else {
                        return f->size;
                }
        default:
                fprintf (stderr, "Unexpected tree tag %d\n", CB_TREE_TAG (x));
                ABORT ();
        }
/* NOT REACHED */
        return 0;
}
```

7.16.4.159 int cb_field_subordinate (struct cb_field * p, struct cb_field * f)

Definition at line 1562 of file tree.c.

```
{
        for (p = p->parent; p; p = p->parent) {
                if (p == f) {
                        return 1;
                }
        }
        return 0;
}
```

7.16.4.160 struct cb_field* cb_field_variable_address (struct cb_field * f) [read]

Definition at line 1545 of file tree.c.

```
{
        struct cb_field *p;

        for (p = f->parent; p; f = f->parent, p = f->parent) {
                for (p = p->children; p != f; p = p->sister) {
                        if (p->occurs_depending || cb_field_variable_size (p)) {
                                return p;
                        }
                }
        }
        return NULL;
}
```

7.16.4.161 struct cb_field* cb_field_variable_size (struct cb_field * f) [read]

Definition at line 1530 of file tree.c.

```
{
        struct cb_field *p;
```

```
        for (f = f->children; f; f = f->sister) {
                if (f->occurs_depending) {
                        return f;
                } else if ((p = cb_field_variable_size (f)) != NULL) {
                        return p;
                }
        }
        return NULL;
}
```

7.16.4.162 int cb_fits_int (cb_tree *x*)

Definition at line 587 of file tree.c.

```
{
        struct cb_literal       *l;
        struct cb_field         *f;

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_LITERAL:
                l = CB_LITERAL (x);
                if (l->scale <= 0 && l->size < 10) {
                        return 1;
                }
                return 0;
        case CB_TAG_FIELD:
                f = CB_FIELD (x);
                switch (f->usage) {
                case CB_USAGE_INDEX:
                case CB_USAGE_LENGTH:
                        return 1;
                case CB_USAGE_BINARY:
                case CB_USAGE_COMP_5:
                case CB_USAGE_COMP_X:
                        if (f->pic->scale <= 0 && f->size <= (int)sizeof (int)) {

                                return 1;
                        }
                        return 0;
                case CB_USAGE_DISPLAY:
                        if (f->size < 10) {
                                if (!f->pic || f->pic->scale <= 0) {
                                        return 1;
                                }
                        }
                        return 0;
                case CB_USAGE_PACKED:
                        if (f->pic->scale <= 0 && f->pic->digits < 10) {
                                return 1;
                        }
                        return 0;
                default:
                        return 0;
                }
        case CB_TAG_REFERENCE:
                return cb_fits_int (CB_REFERENCE (x)->value);
        default:
                return 0;
        }
}
```

7.16.4.163 int cb_fits_long_long (cb_tree *x*)

Definition at line 635 of file tree.c.

```
{
        struct cb_literal       *l;
        struct cb_field         *f;

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_LITERAL:
                l = CB_LITERAL (x);
                if (l->scale <= 0 && l->size < 19) {
                        return 1;
                }
                return 0;
        case CB_TAG_FIELD:
                f = CB_FIELD (x);
                switch (f->usage) {
                case CB_USAGE_INDEX:
                case CB_USAGE_LENGTH:
                        return 1;
                case CB_USAGE_BINARY:
                case CB_USAGE_COMP_5:
                case CB_USAGE_COMP_X:
                        if (f->pic->scale <= 0 && f->size <= (int)sizeof (long lo
      ng)) {
                                return 1;
                        }
                        return 0;
                case CB_USAGE_DISPLAY:
                        if (f->pic->scale <= 0 && f->size < 19) {
                                return 1;
                        }
                        return 0;
                default:
                        return 0;
                }
        case CB_TAG_REFERENCE:
                return cb_fits_long_long (CB_REFERENCE (x)->value);
        default:
                return 0;
        }
}
```

7.16.4.164 int cb_get_int (cb_tree *x*)

Definition at line 676 of file tree.c.

```
{
        struct cb_literal       *l;
        size_t                  i;
        int                     val = 0;

        l = CB_LITERAL (x);
        for (i = 0; i < l->size; i++) {
                if (l->data[i] != '0') {
                        break;
                }
```

```
        }

/* RXWRXW
        if (l->size - i >= 10) {
                ABORT ();
        }
*/

        for (; i < l->size; i++) {
                val = val * 10 + l->data[i] - '0';
        }
        if (l->sign < 0) {
                val = -val;
        }
        return val;
}
```

7.16.4.165 int cb_get_level (cb_tree *x*)

Definition at line 41 of file field.c.

```
{
        const char      *p;
        const char      *name;
        int             level = 0;

        name = CB_NAME (x);
        /* get level */
        for (p = name; *p; p++) {
                if (!isdigit (*p)) {
                        goto level_error;
                }
                level = level * 10 + (*p - '0');
        }

        /* check level */
        switch (level) {
        case 66:
        case 77:
        case 78:
        case 88:
                break;
        default:
                if (level < 1 || level > 49) {
                        goto level_error;
                }
                break;
        }

        return level;

level_error:
        cb_error_x (x, _("Invalid level number '%s'"), name);
        return 0;
}
```

7.16.4.166 long long cb_get_long_long (cb_tree *x*)

Definition at line 705 of file tree.c.

```
{
        struct cb_literal       *l;
        size_t                  i;
        long long               val = 0;

        l = CB_LITERAL (x);
        for (i = 0; i < l->size; i++) {
                if (l->data[i] != '0') {
                        break;
                }
        }

        if (l->size - i >= 19) {
                ABORT ();
        }

        for (; i < l->size; i++) {
                val = val * 10 + l->data[i] - '0';
        }
        if (l->sign < 0) {
                val = -val;
        }
        return val;
}
```

7.16.4.167 void cb_init_constants (void)

Definition at line 732 of file tree.c.

```
{
        char    *s;
        int     i;

        cb_error_node = make_constant (CB_CATEGORY_UNKNOWN, NULL);
        cb_any = make_constant (CB_CATEGORY_UNKNOWN, NULL);
        cb_true = make_constant (CB_CATEGORY_BOOLEAN, "1");
        cb_false = make_constant (CB_CATEGORY_BOOLEAN, "0");
        cb_null = make_constant (CB_CATEGORY_DATA_POINTER, "0");
        cb_zero = make_constant (CB_CATEGORY_NUMERIC, "&cob_zero");
        cb_one = make_constant (CB_CATEGORY_NUMERIC, "&cob_one");
        cb_space = make_constant (CB_CATEGORY_ALPHANUMERIC, "&cob_space");
        cb_low = make_constant (CB_CATEGORY_ALPHANUMERIC, "&cob_low");
        cb_norm_low = cb_low;
        cb_high = make_constant (CB_CATEGORY_ALPHANUMERIC, "&cob_high");
        cb_norm_high = cb_high;
        cb_quote = make_constant (CB_CATEGORY_ALPHANUMERIC, "&cob_quote");
        cb_int0 = cb_int (0);
        cb_int1 = cb_int (1);
        cb_int2 = cb_int (2);
        cb_int3 = cb_int (3);
        cb_int4 = cb_int (4);
        cb_int5 = cb_int (5);
        for (i = 1; i < 8; i++) {
                s = cobc_malloc (4);
```

```
                sprintf (s, "i%d", i);
                cb_i[i] = make_constant (CB_CATEGORY_NUMERIC, s);
        }
        cb_standard_error_handler = make_constant_label ("Default Error Handler")
      ;
}
```

7.16.4.168 void cb_init_reserved (void)

Definition at line 1019 of file reserved.c.

```
{
        int     i;

        /* build system-name table */
        for (i = 0; system_table[i].name != NULL; ++i) {
                system_table[i].node =
                  cb_build_system_name (system_table[i].category, system_table[i]
      .token);
        }
}
```

7.16.4.169 void cb_init_tarrying (void)

Definition at line 3754 of file typeck.c.

```
{
        inspect_func = NULL;
        inspect_data = NULL;
}
```

7.16.4.170 cb_tree cb_int (int *n*)

Definition at line 881 of file tree.c.

```
{
        struct cb_integer       *x;
        struct int_node         *p;

        for (p = int_node_table; p; p = p->next) {
                if (p->n == n) {
                        return p->node;
                }
        }

        x = make_tree (CB_TAG_INTEGER, CB_CATEGORY_NUMERIC, sizeof (struct
      cb_integer));
        x->val = n;

        p = cobc_malloc (sizeof (struct int_node));
        p->n = n;
        p->node = CB_TREE (x);
```

```
        p->next = int_node_table;
        int_node_table = p;
        return p->node;
}
```

7.16.4.171 cb_tree cb_list_add (cb_tree *l*, cb_tree *x*)

Definition at line 798 of file tree.c.

```
{
        return cb_list_append (l, cb_list_init (x));
}
```

7.16.4.172 cb_tree cb_list_append (cb_tree *l1*, cb_tree *l2*)

Definition at line 781 of file tree.c.

```
{
        cb_tree l;

        if (l1 == NULL) {
                return l2;
        } else {
                l = l1;
                while (CB_CHAIN (l)) {
                        l = CB_CHAIN (l);
                }
                CB_CHAIN (l) = l2;
                return l1;
        }
}
```

7.16.4.173 void cb_list_intrinsics (void)

Definition at line 979 of file reserved.c.

```
{
        const char      *s;
        size_t          i;
        size_t          n;

        printf ("Intrinsic Function (Implemented Y/N)\n\n");
        for (i = 0; i < NUM_INTRINSICS; ++i) {
                n = strlen (function_list[i].name);
                switch (n / 8) {
                case 0:
                        s = "\t\t\t\t";
                        break;
                case 1:
                        s = "\t\t\t";
                        break;
                case 2:
```

```
                        s = "\t\t";
                        break;
                default:
                        s = "\t";
                        break;
                }
                printf ("%s%s(%s)\n", function_list[i].name, s,
                        function_list[i].implemented ? "Y" : "N");
        }
}
```

7.16.4.174 int cb_list_length (cb_tree *l*)

Definition at line 818 of file tree.c.

```
{
        int n = 0;

        for (; l; l = CB_CHAIN (l)) {
                n++;
        }
        return n;
}
```

7.16.4.175 void cb_list_map (cb_tree(*)(cb_tree x) *func*, cb_tree *l*)

Definition at line 829 of file tree.c.

```
{
        for (; l; l = CB_CHAIN (l)) {
                CB_VALUE (l) = func (CB_VALUE (l));
        }
}
```

7.16.4.176 void cb_list_mnemonics (void)

Definition at line 1008 of file reserved.c.

```
{
        size_t          i;

        printf ("Mnemonic names\n\n");
        for (i = 0; system_table[i].name != NULL; ++i) {
                printf ("%s\n", system_table[i].name);
        }
}
```

7.16.4.177 void cb_list_reserved (void)

Definition at line 950 of file reserved.c.

```
{
        const char      *s;
        size_t  i;
        size_t  n;

        printf ("Reserved Words (Parsed Y/N)\n\n");
        for (i = 0; i < NUM_RESERVED_WORDS; ++i) {
                n = strlen (reserved_words[i].name);
                switch (n / 8) {
                case 0:
                        s = "\t\t\t\t";
                        break;
                case 1:
                        s = "\t\t\t";
                        break;
                case 2:
                        s = "\t\t";
                        break;
                default:
                        s = "\t";
                        break;
                }
                printf ("%s%s(%s)\n", reserved_words[i].name, s,
                        reserved_words[i].token != -1 ? "Y" : "N");
        }
}
```

7.16.4.178 cb_tree cb_list_reverse (cb_tree *l*)

Definition at line 804 of file tree.c.

```
{
        cb_tree next;
        cb_tree last = NULL;

        for (; l; l = next) {
                next = CB_CHAIN (l);
                CB_CHAIN (l) = last;
                last = l;
        }
        return last;
}
```

7.16.4.179 char* cb_name (cb_tree *x*)

Definition at line 441 of file tree.c.

```
{
        if (!treenamebuff) {
                treenamebuff = cobc_malloc (COB_NORMAL_BUFF);
```

```
        }
        cb_name_1 (treenamebuff, x);
        return treenamebuff;
}
```

7.16.4.180 cb_tree cb_ref (cb_tree *x*)

Definition at line 1794 of file tree.c.

```
{
        struct cb_reference     *r;
        struct cb_field         *p;
        struct cb_label         *s;
        cb_tree                 candidate = NULL;
        cb_tree                 items;
        cb_tree                 cb1;
        cb_tree                 cb2;
        cb_tree                 v;
        cb_tree                 c;
        struct cb_program       *prog;
        struct cb_word          *w;
        size_t                  val;
        size_t                  ambiguous = 0;

        r = CB_REFERENCE (x);
        /* if this reference has already been resolved (and the value
           has been cached), then just return the value */
        if (r->value) {
                return r->value;
        }
        /* resolve the value */

        items = r->word->items;
        for (; items; items = CB_CHAIN (items)) {
                /* find a candidate value by resolving qualification */
                v = CB_VALUE (items);
                c = r->chain;
                switch (CB_TREE_TAG (v)) {
                case CB_TAG_FIELD:
                        /* in case the value is a field, it might be qualified
                           by its parent names and a file name */
                        if (CB_FIELD (v)->flag_indexed_by) {
                                p = CB_FIELD (v)->index_qual;
                        } else {
                                p = CB_FIELD (v)->parent;
                        }
                        /* resolve by parents */
                        for (; p; p = p->parent) {
                                if (c && strcasecmp (CB_NAME (c), p->name) == 0)
      {
                                        c = CB_REFERENCE (c)->chain;
                                }
                        }

                        /* resolve by file */
                        if (c && CB_REFERENCE (c)->chain == NULL) {
                                if (CB_REFERENCE (c)->word->count == 1 &&
      CB_FILE_P (cb_ref (c))
                                    && (CB_FILE (cb_ref (c)) == cb_field_founder
```

```
(CB_FIELD (v))->file)) {
                                        c = CB_REFERENCE (c)->chain;
                                }
                        }

                        break;
                case CB_TAG_LABEL:
                        /* in case the value is a label, it might be qualified
                           by its section name */
                        s = CB_LABEL (v)->section;

                        /* unqualified paragraph name referenced within the secti
on
                           is resolved without ambiguity check if not duplicated
*/
                        if (c == NULL && r->offset && s == CB_LABEL (r->offset))
{
                                for (cb1 = CB_CHAIN (items); cb1; cb1 = CB_CHAIN
(cb1)) {
                                        cb2 = CB_VALUE (cb1);
                                        if (s == CB_LABEL (cb2)->section) {
                                                ambiguous_error (x);
                                                goto error;
                                        }
                                }
                                candidate = v;
                                goto end;
                        }

                        /* resolve by section name */
                        if (c && s && strcasecmp (CB_NAME (c), (char *)s->name) =
= 0) {
                                c = CB_REFERENCE (c)->chain;
                        }

                        break;
                default:
                        /* other values cannot be qualified */
                        break;
                }

                /* a well qualified value is a good candidate */
                if (c == NULL) {
                        if (candidate == NULL) {
                                /* keep the first candidate */
                                candidate = v;
                        } else {
                                /* multiple candidates and possibly ambiguous */
                                ambiguous = 1;
                                /* continue search because the reference might no
t
                                   be ambiguous and exit loop by "goto end" later
 */
                        }
                }
          }

          /* there is no candidate */
          if (candidate == NULL) {
                  if (current_program->nested_level > 0) {
                          /* Nested program - check parents for GLOBAL candidate */
```

```
                        ambiguous = 0;
                        val = hash ((const unsigned char *)r->word->name);
                        prog = current_program->next_program;
                        for (; prog; prog = prog->next_program) {
                                if (prog->nested_level >= current_program->
      nested_level) {
                                        continue;
                                }
                                for (w = prog->word_table[val]; w; w = w->next) {

                                        if (strcasecmp (r->word->name, w->name) =
      = 0) {
                                                candidate = global_check (r, w->
      items, &ambiguous);
                                                if (candidate) {
                                                        if (ambiguous) {
                                                                ambiguous_error (
      x);
                                                                goto error;
                                                        }
                                                        if (CB_FILE_P(candidate))
       {
                                                                current_program->
      gen_file_error = 1;
                                                        }
                                                        goto end;
                                                }
                                        }
                                }
                                if (prog->nested_level == 0) {
                                        break;
                                }
                        }
                }
                undefined_error (x);
                goto error;
        }

        /* the reference is ambiguous */
        if (ambiguous) {
                ambiguous_error (x);
                goto error;
        }

end:
        if (CB_FIELD_P (candidate)) {
                CB_FIELD (candidate)->count++;
                if (CB_FIELD (candidate)->flag_invalid) {
                        goto error;
                }
        }

        r->value = candidate;
        return r->value;

error:
        r->value = cb_error_node;
        return cb_error_node;
}
```

7.16.4.181 void cb_reset_78 (void)

7.16.4.182 void cb_reset_in_procedure (void)

7.16.4.183 struct cb_field* cb_resolve_redefines (struct cb_field * *field,* cb_tree *redefines*) [read]

Definition at line 231 of file field.c.

```
{
        struct cb_field         *f;
        struct cb_reference     *r;
        const char              *name;
        cb_tree                 x;

        r = CB_REFERENCE (redefines);
        name = CB_NAME (redefines);
        x = CB_TREE (field);

        /* check qualification */
        if (r->chain) {
                cb_error_x (x, _("'%s' cannot be qualified here"), name);
                return NULL;
        }

        /* check subscripts */
        if (r->subs) {
                cb_error_x (x, _("'%s' cannot be subscripted here"), name);
                return NULL;
        }

        /* resolve the name in the current group (if any) */
        if (field->parent && field->parent->children) {
                for (f = field->parent->children; f; f = f->sister) {
                        if (strcasecmp (f->name, name) == 0) {
                                break;
                        }
                }
                if (f == NULL) {
                        cb_error_x (x, _("'%s' undefined in '%s'"), name, field->
      parent->name);
                        return NULL;
                }
        } else {
                if (cb_ref (redefines) == cb_error_node) {
                        return NULL;
                }
                f = cb_field (redefines);
        }

        /* check level number */
        if (f->level != field->level) {
                cb_error_x (x, _("Level number of REDEFINES entries must be ident
      ical"));
                return NULL;
        }
        if (f->level == 66 || f->level == 88) {
                cb_error_x (x, _("Level number of REDEFINES entry cannot be 66 or
       88"));
                return NULL;
```

```
        }

        if (!cb_indirect_redefines && f->redefines) {
                cb_error_x (x, _("'%s' not the original definition"), f->name);
                return NULL;
        }

        /* return the original definition */
        while (f->redefines) {
                f = f->redefines;
        }
        return f;
}
```

7.16.4.184 void cb_set_in_procedure (void)

7.16.4.185 enum cb_category cb_tree_category (cb_tree *x*)

Definition at line 458 of file tree.c.

```
{
        struct cb_cast          *p;
        struct cb_reference     *r;
        struct cb_field         *f;

        if (x == cb_error_node) {
                return 0;
        }
        if (x->category != CB_CATEGORY_UNKNOWN) {
                return x->category;
        }

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_CAST:
                p = CB_CAST (x);
                switch (p->type) {
                case CB_CAST_ADDRESS:
                case CB_CAST_ADDR_OF_ADDR:
                        x->category = CB_CATEGORY_DATA_POINTER;
                        break;
                case CB_CAST_PROGRAM_POINTER:
                        x->category = CB_CATEGORY_PROGRAM_POINTER;
                        break;
                default:
                        fprintf (stderr, "Unexpected cast type -> %d\n", p->type)
      ;
                        ABORT ();
                }
                break;
        case CB_TAG_REFERENCE:
                r = CB_REFERENCE (x);
                if (r->offset) {
                        x->category = CB_CATEGORY_ALPHANUMERIC;
                } else {
                        x->category = cb_tree_category (r->value);
                }
                break;
        case CB_TAG_FIELD:
```

```
                f = CB_FIELD (x);
                if (f->children) {
                        x->category = CB_CATEGORY_ALPHANUMERIC;
                } else if (f->usage == CB_USAGE_POINTER && f->level != 88) {
                        x->category = CB_CATEGORY_DATA_POINTER;
                } else if (f->usage == CB_USAGE_PROGRAM_POINTER && f->level != 88
      ) {
                        x->category = CB_CATEGORY_PROGRAM_POINTER;
                } else {
                        switch (f->level) {
                        case 66:
                                if (f->rename_thru) {
                                        x->category = CB_CATEGORY_ALPHANUMERIC;
                                } else {
                                        x->category = cb_tree_category (CB_TREE (
      f->redefines));
                                }
                                break;
                        case 88:
                                x->category = CB_CATEGORY_BOOLEAN;
                                break;
                        default:
                                x->category = f->pic->category;
                                break;
                        }
                }
                break;
          case CB_TAG_ALPHABET_NAME:
          case CB_TAG_LOCALE_NAME:
                x->category = CB_CATEGORY_ALPHANUMERIC;
                break;
          case CB_TAG_BINARY_OP:
                x->category = CB_CATEGORY_BOOLEAN;
                break;
          default:
                fprintf (stderr, "Unknown tree tag %d Category %d\n",
      CB_TREE_TAG (x), x->category);
                ABORT ();
          }

          return x->category;
}
```

7.16.4.186 enum cb_class cb_tree_class (cb_tree *x*)

Definition at line 451 of file tree.c.

```
{

          return category_to_class_table[CB_TREE_CATEGORY (x)];
}
```

7.16.4.187 int cb_tree_type (cb_tree *x*)

Definition at line 537 of file tree.c.

```
{
        struct cb_field *f;

        f = cb_field (x);
        if (f->children) {
                return COB_TYPE_GROUP;
        }

        switch (CB_TREE_CATEGORY (x)) {
        case CB_CATEGORY_ALPHABETIC:
        case CB_CATEGORY_ALPHANUMERIC:
                return COB_TYPE_ALPHANUMERIC;
        case CB_CATEGORY_ALPHANUMERIC_EDITED:
                return COB_TYPE_ALPHANUMERIC_EDITED;
        case CB_CATEGORY_NUMERIC:
                switch (f->usage) {
                case CB_USAGE_DISPLAY:
                        return COB_TYPE_NUMERIC_DISPLAY;
                case CB_USAGE_BINARY:
                case CB_USAGE_COMP_5:
                case CB_USAGE_COMP_X:
                case CB_USAGE_INDEX:
                case CB_USAGE_LENGTH:
                        return COB_TYPE_NUMERIC_BINARY;
                case CB_USAGE_FLOAT:
                        return COB_TYPE_NUMERIC_FLOAT;
                case CB_USAGE_DOUBLE:
                        return COB_TYPE_NUMERIC_DOUBLE;
                case CB_USAGE_PACKED:
                        return COB_TYPE_NUMERIC_PACKED;
                default:
                        fprintf (stderr, "Unexpected numeric usage -> %d\n", f->
      usage);
                        ABORT ();
                }
          case CB_CATEGORY_NUMERIC_EDITED:
                return COB_TYPE_NUMERIC_EDITED;
          case CB_CATEGORY_OBJECT_REFERENCE:
          case CB_CATEGORY_DATA_POINTER:
          case CB_CATEGORY_PROGRAM_POINTER:
                return COB_TYPE_NUMERIC_BINARY;
          default:
                fprintf (stderr, "Unexpected category -> %d\n", CB_TREE_CATEGORY
      (x));
                ABORT ();
          }
/* NOT REACHED */
        return 0;
}
```

7.16.4.188 struct cb_field* cb_validate_78_item (struct cb_field * *p*) [read]

Definition at line 1033 of file field.c.

```
{
        cb_tree x;

        x = CB_TREE (f);
        if (!f->values) {
```

```
                level_require_error (x, "VALUE");
        }

        if (f->pic || f->flag_occurs) {
                level_except_error (x, "VALUE");
        }
        cb_add_78 (f);
        return last_real_field;
}
```

7.16.4.189 void cb_validate_88_item (struct cb_field * p)

Definition at line 1018 of file field.c.

```
{
        cb_tree x;

        x = CB_TREE (f);
        if (!f->values) {
                level_require_error (x, "VALUE");
        }

        if (f->pic || f->flag_occurs) {
                level_except_error (x, "VALUE");
        }
}
```

7.16.4.190 void cb_validate_field (struct cb_field * p)

Definition at line 973 of file field.c.

```
{
        struct cb_field         *c;

        if (validate_field_1 (f) != 0) {
                f->flag_invalid = 1;
                return;
        }
        /* RXW - Remove */
        if (f->flag_item_78) {
                f->flag_is_verified = 1;
                return;
        }

        /* setup parameters */
        if (f->storage == CB_STORAGE_LOCAL ||
            f->storage == CB_STORAGE_LINKAGE ||
            f->flag_item_based) {
                f->flag_local = 1;
        }
        if (f->storage == CB_STORAGE_LINKAGE || f->flag_item_based) {
                f->flag_base = 1;
        }
        setup_parameters (f);
```

```
        /* compute size */
        compute_size (f);
        if (!f->redefines) {
                f->memory_size = f->size * f->occurs_max;
        } else if (f->redefines->memory_size < f->size * f->occurs_max) {
                f->redefines->memory_size = f->size * f->occurs_max;
        }

        validate_field_value (f);
        if (f->flag_is_global) {
                f->count++;
                for (c = f->children; c; c = c->sister) {
                        c->flag_is_global = 1;
                        c->count++;
                }
        }
        f->flag_is_verified = 1;
}
```

7.16.4.191 void cb_validate_program_body (struct cb_program * *prog*)

Definition at line 1416 of file typeck.c.

```
{
        /* resolve all labels */
        cb_tree l;
        cb_tree x;
        cb_tree v;

        for (l = cb_list_reverse (prog->label_list); l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
                v = cb_ref (x);
                if (CB_LABEL_P (v)) {
                        CB_LABEL (v)->need_begin = 1;
                        if (CB_REFERENCE (x)->length) {
                                CB_LABEL (v)->need_return = 1;
                        }
                } else if (v != cb_error_node) {
                        cb_error_x (x, _("'%s' not procedure name"), cb_name (x))
      ;
                }
        }

        prog->file_list = cb_list_reverse (prog->file_list);
        prog->exec_list = cb_list_reverse (prog->exec_list);
}
```

7.16.4.192 void cb_validate_program_data (struct cb_program * *prog*)

Definition at line 1286 of file typeck.c.

```
{
        cb_tree         l;
        cb_tree         x;
        cb_tree         assign;
```

```
  struct cb_field *p;
  struct cb_file  *f;
  unsigned char   *c;

  for (l = current_program->file_list; l; l = CB_CHAIN (l)) {
          f = CB_FILE (CB_VALUE (l));
          if (!f->finalized) {
                  finalize_file (f, NULL);
          }
  }
  /* build undeclared assignment name now */
  if (cb_assign_clause == CB_ASSIGN_MF) {
          for (l = current_program->file_list; l; l = CB_CHAIN (l)) {
                  assign = CB_FILE (CB_VALUE (l))->assign;
                  if (!assign) {
                          continue;
                  }
                  if (CB_REFERENCE_P (assign)) {
                          for (x = current_program->file_list; x; x =
CB_CHAIN (x)) {
                                  if (!strcmp (CB_FILE (CB_VALUE (x))->
name,
                                       CB_REFERENCE (assign)->word->name))
{
                                     redefinition_error (assign);
                                  }
                          }
                          p = check_level_78 (CB_REFERENCE (assign)->word->
name);
                          if (p) {
                                  c = (unsigned char *)CB_LITERAL(CB_VALUE(
p->values))->data;
                                  assign = CB_TREE (build_literal (
CB_CATEGORY_ALPHANUMERIC, c, strlen ((char *)c)));
                                  CB_FILE (CB_VALUE (l))->assign = assign;
                          }
                  }
                  if (CB_REFERENCE_P (assign) && CB_REFERENCE (assign)->wor
d->count == 0) {
                          if (cb_warn_implicit_define) {
                                  cb_warning (_("'%s' will be implicitly de
fined"), CB_NAME (assign));
                          }
                          x = cb_build_implicit_field (assign,
COB_SMALL_BUFF);
                          p = current_program->working_storage;
                          CB_FIELD (x)->count++;
                          if (p) {
                                  while (p->sister) {
                                          p = p->sister;
                                  }
                                  p->sister = CB_FIELD (x);
                          } else {
                                  current_program->working_storage =
CB_FIELD (x);
                          }
                  }
                  if (CB_REFERENCE_P (assign)) {
                          x = cb_ref (assign);
                          if (CB_FIELD_P (x) && CB_FIELD (x)->level == 88)
{
                                  cb_error_x (assign, _("ASSIGN data item '
```

```
      %s' invalid"), CB_NAME (assign));
                                }
                        }
                }
        }

        if (prog->cursor_pos) {
                x = cb_ref (prog->cursor_pos);
                if (x == cb_error_node) {
                        prog->cursor_pos = NULL;
                } else if (CB_FIELD(x)->size != 6 && CB_FIELD(x)->size != 4) {
                        cb_error_x (prog->cursor_pos, _("'%s' CURSOR is not 4 or
      6 characters long"),
                                    cb_name (prog->cursor_pos));
                        prog->cursor_pos = NULL;
                }
        }
        if (prog->crt_status) {
                x = cb_ref (prog->crt_status);
                if (x == cb_error_node) {
                        prog->crt_status = NULL;
                } else if (CB_FIELD(x)->size != 4) {
                        cb_error_x (prog->crt_status, _("'%s' CRT STATUS is not 4
       characters long"),
                                    cb_name (prog->crt_status));
                        prog->crt_status = NULL;
                }
        } else {
                l = cb_build_reference ("COB-CRT-STATUS");
                p = CB_FIELD (cb_build_field (l));
                p->usage = CB_USAGE_DISPLAY;
                p->pic = CB_PICTURE (cb_build_picture ("9(4)"));
                cb_validate_field (p);
                p->flag_no_init = 1;
                /* Do not initialize/bump ref count here
                p->values = cb_list_init (cb_zero);
                p->count++;
                */
                current_program->working_storage =
                        cb_field_add (current_program->working_storage, p);
                prog->crt_status = l;
                /* RXWRXW - Maybe better
                prog->crt_status = cb_build_index (cb_build_reference ("COB-CRT-S
      TATUS"), cb_zero, 0, NULL);
                */
        }

        /* resolve all references so far */
        for (l = cb_list_reverse (prog->reference_list); l; l = CB_CHAIN (l)) {
                cb_ref (CB_VALUE (l));
        }
        for (l = current_program->file_list; l; l = CB_CHAIN (l)) {
                f = CB_FILE (CB_VALUE (l));
                if (f->record_depending && f->record_depending != cb_error_node)
      {
                        x = f->record_depending;
                        if (cb_ref (x) != cb_error_node) {
/* RXW - This breaks old legacy programs
                                if (CB_REF_OR_FIELD_P(x)) {
                                        p = cb_field (x);
                                        switch (p->storage) {
                                        case CB_STORAGE_WORKING:
```

```
                                        case CB_STORAGE_LOCAL:
                                        case CB_STORAGE_LINKAGE:
                                                break;
                                        default:
                                                cb_error (_("RECORD DEPENDING ite
      m must be in WORKING/LOCAL/LINKAGE section"));
                                        }
                                } else {
*/
                                if (!CB_REFERENCE_P(x) && !CB_FIELD_P(x)) {
                                        cb_error (_("Invalid RECORD DEPENDING ite
      m"));
                                }
                        }
                }
          }
}
```

7.16.4.193 void cb_validate_program_environment (struct cb_program * *prog*)

Definition at line 1079 of file typeck.c.

```
{
        cb_tree                 x;
        cb_tree                 y;
        cb_tree                 l;
        cb_tree                 ls;
        struct cb_alphabet_name *ap;
        unsigned char           *data;
        size_t                  dupls;
        size_t                  unvals;
        size_t                  count;
        int                     lower;
        int                     upper;
        int                     size;
        int                     n;
        int                     i;
        int                     lastval;
        int                     values[256];

        /* Check ALPHABET clauses */
        for (l = current_program->alphabet_name_list; l; l = CB_CHAIN (l)) {
                ap = CB_ALPHABET_NAME (CB_VALUE (l));
                if (ap->type != CB_ALPHABET_CUSTOM) {
                        continue;
                }
                ap->low_val_char = 0;
                ap->high_val_char = 255;
                dupls = 0;
                unvals = 0;
                count = 0;
                lastval = 0;
                for (n = 0; n < 256; n++) {
                        values[n] = -1;
                }
                for (y = ap->custom_list; y; y = CB_CHAIN (y)) {
                        if (count > 255) {
                                unvals = 1;
                                break;
```

```
                        }
                        x = CB_VALUE (y);
                        if (CB_PAIR_P (x)) {
                                /* X THRU Y */
                                lower = get_value (CB_PAIR_X (x));
                                upper = get_value (CB_PAIR_Y (x));
                                lastval = upper;
                                if (!count) {
                                        ap->low_val_char = lower;
                                }
                                if (lower < 0 || lower > 255) {
                                        unvals = 1;
                                        continue;
                                }
                                if (upper < 0 || upper > 255) {
                                        unvals = 1;
                                        continue;
                                }
                                if (lower <= upper) {
                                        for (i = lower; i <= upper; i++) {
                                                if (values[i] != -1) {
                                                        dupls = 1;
                                                }
                                                values[i] = i;
                                                count++;
                                        }
                                } else {
                                        for (i = lower; i >= upper; i--) {
                                                if (values[i] != -1) {
                                                        dupls = 1;
                                                }
                                                values[i] = i;
                                                count++;
                                        }
                                }
                        } else if (CB_LIST_P (x)) {
                                /* X ALSO Y ... */
                                if (!count) {
                                        ap->low_val_char = get_value (CB_VALUE (x
    ));
                                }
                                for (ls = x; ls; ls = CB_CHAIN (ls)) {
                                        n = get_value (CB_VALUE (ls));
                                        if (!CB_CHAIN (ls)) {
                                                lastval = n;
                                        }
                                        if (n < 0 || n > 255) {
                                                unvals = 1;
                                                continue;
                                        }
                                        if (values[n] != -1) {
                                                dupls = 1;
                                        }
                                        values[n] = n;
                                        count++;
                                }
                        } else {
                                /* literal */
                                if (CB_TREE_CLASS (x) == CB_CLASS_NUMERIC) {
                                        n = get_value (x);
                                        lastval = n;
                                        if (!count) {
```

```
                                        ap->low_val_char = n;
                                }
                                if (n < 0 || n > 255) {
                                        unvals = 1;
                                        continue;
                                }
                                if (values[n] != -1) {
                                        dupls = 1;
                                }
                                values[n] = n;
                                count++;
                        } else if (CB_LITERAL_P (x)) {
                                size = (int)CB_LITERAL (x)->size;
                                data = CB_LITERAL (x)->data;
                                if (!count) {
                                        ap->low_val_char = data[0];
                                }
                                lastval = data[size - 1];
                                for (i = 0; i < size; i++) {
                                        n = data[i];
                                        if (values[n] != -1) {
                                                dupls = 1;
                                        }
                                        values[n] = n;
                                        count++;
                                }
                        } else {
                                n = get_value (x);
                                lastval = n;
                                if (!count) {
                                        ap->low_val_char = n;
                                }
                                if (n < 0 || n > 255) {
                                        unvals = 1;
                                        continue;
                                }
                                if (values[n] != -1) {
                                        dupls = 1;
                                }
                                values[n] = n;
                                count++;
                        }
                }
        }
        if (dupls || unvals) {
                if (dupls) {
                        cb_error_x (l, _("Duplicate character values in a
lphabet '%s'"),
                                    cb_name (CB_VALUE(l)));
                }
                if (unvals) {
                        cb_error_x (l, _("Invalid character values in alp
habet '%s'"),
                                    cb_name (CB_VALUE(l)));
                }
                ap->low_val_char = 0;
                ap->high_val_char = 255;
                continue;
        }
        /* Calculate HIGH-VALUE */
        /* If all 256 values have been specified, HIGH-VALUE is the last
one */
```

```
                /* Otherwise if HIGH-VALUE has been specified, find the highest *
      /
                /* value that has not been used */
                if (count == 256) {
                        ap->high_val_char = lastval;
                } else if (values[255] != -1) {
                        for (n = 254; n >= 0; n--) {
                                if (values[n] == -1) {
                                        ap->high_val_char = n;
                                        break;
                                }
                        }
                }
        }
        /* Rest HIGH/LOW-VALUES */
        cb_low = cb_norm_low;
        cb_high = cb_norm_high;
        /* resolve the program collating sequence */
        if (!prog->collating_sequence) {
                return;
        }
        x = cb_ref (prog->collating_sequence);
/* RXWRXW
        if (x == cb_error_node) {
                prog->collating_sequence = NULL;
                return;
        }
*/
        if (!CB_ALPHABET_NAME_P (x)) {
                cb_error_x (prog->collating_sequence, _("'%s' not alphabet name")
      ,
                            cb_name (prog->collating_sequence));
                prog->collating_sequence = NULL;
                return;
        }
        if (CB_ALPHABET_NAME (x)->type != CB_ALPHABET_CUSTOM) {
                return;
        }
        if (CB_ALPHABET_NAME (x)->low_val_char) {
                cb_low = cb_build_alphanumeric_literal ((ucharptr)"\0", 1);
                CB_LITERAL(cb_low)->data[0] = CB_ALPHABET_NAME (x)->low_val_char;

                CB_LITERAL(cb_low)->all = 1;
        }
        if (CB_ALPHABET_NAME (x)->high_val_char != 255){
                cb_high = cb_build_alphanumeric_literal ((ucharptr)"\0", 1);
                CB_LITERAL(cb_high)->data[0] = CB_ALPHABET_NAME (x)->high_val_cha
      r;
                CB_LITERAL(cb_high)->all = 1;
        }
}
```

7.16.4.194 void cb_warning_x (cb_tree *x*, const char * *fmt*, ...)

Definition at line 98 of file error.c.

```
{
        va_list ap;
```

```
        va_start (ap, fmt);
        print_error ((char *)(x->source_file), x->source_line, "Warning: ", fmt,
      ap);
        va_end (ap);

        warningcount++;
}
```

7.16.4.195 char* check_filler_name (char * *name*)

Definition at line 64 of file error.c.

```
{
        if (!memcmp (name, "WORK$", 5)) {
                name = (char *)"FILLER";
        }
        return name;
}
```

7.16.4.196 struct cb_field* check_level_78 (const char * *name*) [read]

7.16.4.197 void cobc_tree_cast_error (cb_tree *x*, const char * *filen*, const int *linenum*, const int *tagnum*)

Definition at line 317 of file cobc.c.

```
{
        fprintf (stderr, "%s:%d: Invalid type cast from '%s'\n",
                filen, linenum, x ? cb_name (x) : "null");
        fprintf (stderr, "Tag 1 %d Tag 2 %d\n", x ? CB_TREE_TAG(x) : 0,
                tagnum);
        (void)longjmp (cob_jmpbuf, 1);
}
```

7.16.4.198 void codegen (struct cb_program * *prog*, int *nested*)

Definition at line 4683 of file codegen.c.

```
{
        int                     i;
        cb_tree                 l;
        struct attr_list        *j;
        struct literal_list     *m;
        struct field_list       *k;
        struct call_list        *clp;
        struct base_list        *blp;
        unsigned char           *s;
        struct cb_program       *cp;
        cb_tree                 l1;
        cb_tree                 l2;
        const char              *prevprog;
```

```
        time_t                  loctime;
        char                    locbuff[48];

        current_prog = prog;
        param_id = 0;
        stack_id = 0;
        num_cob_fields = 0;
        progid = 0;
        loop_counter = 0;
        output_indent_level = 0;
        last_line = 0;
        needs_exit_prog = 0;
        gen_custom = 0;
        call_cache = NULL;
        label_cache = NULL;
        local_cache = NULL;
        excp_current_program_id = prog->orig_source_name;
        excp_current_section = NULL;
        excp_current_paragraph = NULL;
        memset ((char *)i_counters, 0, sizeof (i_counters));

        output_target = yyout;

        if (!nested) {
                gen_ebcdic = 0;
                gen_ebcdic_ascii = 0;
                gen_full_ebcdic = 0;
                gen_native = 0;
                attr_cache = NULL;
                base_cache = NULL;
                literal_cache = NULL;
                field_cache = NULL;

                loctime = time (NULL);
                strftime (locbuff, sizeof(locbuff) - 1, "%b %d %Y %H:%M:%S %Z",
                        localtime (&loctime));
                output_header (output_target, locbuff);
                output_header (cb_storage_file, locbuff);
                for (cp = prog; cp; cp = cp->next_program) {
                        output_header (cp->local_storage_file, locbuff);
                }

                output_storage ("/* Frame stack declaration */\n");
                output_storage ("struct cob_frame {\n");
                output_storage ("\tint\tperform_through;\n");
#ifndef __GNUC__
                output_storage ("\tint\treturn_address;\n");
#elif   COB_USE_SETJMP
                output_storage ("\tjmp_buf\treturn_address;\n");
#else
                output_storage ("\tvoid\t*return_address;\n");
#endif
                output_storage ("};\n\n");
                output_storage ("/* Union for CALL statement */\n");
                output_storage ("union cob_call_union {\n");
                output_storage ("\tvoid *(*funcptr)();\n");
                output_storage ("\tint  (*funcint)();\n");
                output_storage ("\tvoid *func_void;\n");
                output_storage ("};\n");
                output_storage ("union cob_call_union\tcob_unifunc;\n\n");

                output ("#define  __USE_STRING_INLINES 1\n");
```

```
#ifdef  _XOPEN_SOURCE_EXTENDED
                output ("#ifndef        _XOPEN_SOURCE_EXTENDED\n");
                output ("#define  _XOPEN_SOURCE_EXTENDED 1\n");
                output ("#endif\n");
#endif
                output ("#include <stdio.h>\n");
                output ("#include <stdlib.h>\n");
                output ("#include <string.h>\n");
                output ("#include <math.h>\n");
#if     COB_USE_SETJMP
                output ("#include <setjmp.h>\n");
#endif
#ifdef  WORDS_BIGENDIAN
                output ("#define WORDS_BIGENDIAN 1\n");
#endif
#ifdef  HAVE_BUILTIN_EXPECT
                output ("#define HAVE_BUILTIN_EXPECT\n");
#endif
                if (optimize_flag) {
                        output ("#define COB_LOCAL_INLINE\n");
                }
                output ("#include <libcob.h>\n\n");

                output ("#define COB_SOURCE_FILE               \"%s\"\n",
      cb_source_file);
                output ("#define COB_PACKAGE_VERSION    \"%s\"\n",
      PACKAGE_VERSION);
                output ("#define COB_PATCH_LEVEL               %d\n\n",
      PATCH_LEVEL);
                output ("/* Global variables */\n");
                output ("#include \"%s\"\n\n", cb_storage_file_name);

                for (cp = prog; cp; cp = cp->next_program) {
                        if (cp->gen_decset) {
                                output("static void\n");
                                output("cob_decimal_set_int (cob_decimal *d, cons
      t int n)\n");
                                output("{\n");
                                output("        mpz_set_si (d->value, n);\n");
                                output("        d->scale = 0;\n");
                                output("}\n\n");
                                break;
                        }
                }
                for (cp = prog; cp; cp = cp->next_program) {
                        if (cp->gen_udecset) {
                                output("static void\n");
                                output("cob_decimal_set_uint (cob_decimal *d, con
      st unsigned int n)\n");
                                output("{\n");
                                output("        mpz_set_ui (d->value, n);\n");
                                output("        d->scale = 0;\n");
                                output("}\n\n");
                                break;
                        }
                }
                for (cp = prog; cp; cp = cp->next_program) {
                        if (cp->gen_ptrmanip) {
                                output("static void\n");
                                output("cob_pointer_manip (cob_field *f1, cob_fie
      ld *f2, size_t addsub)\n");
                                output("{\n");
```

```
                                output("        unsigned char   *tmptr;\n");
                                output("        memcpy (&tmptr, f1->data, sizeof(
void *));\n");
                                output("        if (addsub) {\n");
                                output("                tmptr -= cob_get_int (f2)
;\n");
                                output("        } else {\n");
                                output("                tmptr += cob_get_int (f2)
;\n");
                                output("        }\n");
                                output("        memcpy (f1->data, &tmptr, sizeof(
void *));\n");
                                output("}\n\n");
                                break;
                        }
                }
                output ("/* Function prototypes */\n\n");
                for (cp = prog; cp; cp = cp->next_program) {
                        /* Build parameter list */
                        for (l = cp->entry_list; l; l = CB_CHAIN (l)) {
                                for (l1 = CB_VALUE (l); l1; l1 = CB_CHAIN (l1)) {

                                        for (l2 = cp->parameter_list; l2; l2 =
CB_CHAIN (l2)) {
                                                if (strcasecmp (cb_field (
CB_VALUE (l1))->name,
                                                                cb_field (
CB_VALUE (l2))->name) == 0) {
                                                        break;
                                                }
                                        }
                                        if (l2 == NULL) {
                                                cp->parameter_list = cb_list_add
(cp->parameter_list, CB_VALUE (l1));
                                        }
                                }
                        }
                        if (cp->flag_main) {
                                output ("int %s ();\n", cp->program_id);
                        } else {
                                for (l = cp->entry_list; l; l = CB_CHAIN (l)) {
                                        output_entry_function (cp, l, cp->
parameter_list, 0);
                                }
                        }
                        output ("static int %s_ (const int", cp->program_id);
                        if (!cp->flag_chained) {
                                for (l = cp->parameter_list; l; l = CB_CHAIN (l))
 {
                                        output (", unsigned char *");
                                }
                        }
                        output (");\n");
                }
                output ("\n");
          }

          /* Class-names */
          if (!prog->nested_level && prog->class_name_list) {
                output ("/* Class names */\n");
                for (l = prog->class_name_list; l; l = CB_CHAIN (l)) {
                        output_class_name_definition (CB_CLASS_NAME (CB_VALUE (l)
```

```
));
          }
  }

  /* Main function */
  if (prog->flag_main) {
          output_main_function (prog);
  }

  /* Functions */
  if (!nested) {
          output ("/* Functions */\n\n");
  }
  for (l = prog->entry_list; l; l = CB_CHAIN (l)) {
          output_entry_function (prog, l, prog->parameter_list, 1);
  }
  output_internal_function (prog, prog->parameter_list);

  if (!prog->next_program) {
          output ("/* End functions */\n\n");
  }

  if (gen_native || gen_full_ebcdic || gen_ebcdic_ascii || prog->
alphabet_name_list) {
          (void)lookup_attr (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL, 0);
  }

  output_target = cb_storage_file;

  /* Program local stuff */
  if (call_cache) {
          output_local ("\n/* Call pointers */\n");
          for (clp = call_cache; clp; clp = clp->next) {
                  output_local ("static union cob_call_union\tcall_%s = { N
ULL };\n", clp->callname);
          }
          output_local ("\n");
  }

  for (i = 0; i < COB_MAX_SUBSCRIPTS; i++) {
          if (i_counters[i]) {
                  output_local ("int\t\ti%d;\n", i);
          }
  }

  if (num_cob_fields) {
          output_local ("\n/* Local cob_field items */\n");
          for (i = 0; i < num_cob_fields; i++) {
                  output_local ("cob_field\tf%d;\n", i);
          }
          output_local ("\n");
  }

  /* Skip to next nested program */

  if (prog->next_program) {
          codegen (prog->next_program, 1);
          return;
  }

  /* Finalize the storage file */
```

```
        if (base_cache) {
                output_storage ("\n/* Storage */\n");
                base_cache = list_cache_sort (base_cache, &base_cache_cmp);
                prevprog = NULL;
                for (blp = base_cache; blp; blp = blp->next) {
                        if (blp->curr_prog != prevprog) {
                                prevprog = blp->curr_prog;
                                output_storage ("\n/* PROGRAM-ID : %s */\n", prev
      prog);
                        }
#ifdef HAVE_ATTRIBUTE_ALIGNED
                        output_storage ("static unsigned char %s%d[%d] __attribut
      e__((aligned));",
#else
                        output_storage ("static unsigned char %s%d[%d];",
#endif
                                        CB_PREFIX_BASE, blp->f->id,
                                        blp->f->memory_size);
                        output_storage ("\t/* %s */\n", blp->f->name);
                }
                output_storage ("\n/* End of storage */\n\n");
        }

        if (attr_cache) {
                output_storage ("\n/* Attributes */\n\n");
                attr_cache = attr_list_reverse (attr_cache);
                for (j = attr_cache; j; j = j->next) {
                        output_storage ("static const cob_field_attr %s%d = ",
                                        CB_PREFIX_ATTR, j->id);
                        output_storage ("{%d, %d, %d, %d, ", j->type, j->digits,
                                        j->scale, j->flags);
                        if (j->pic) {
                                output_storage ("\"");
                                for (s = j->pic; *s; s += 5) {
                                        output_storage ("%c\\%03o\\%03o\\%03o\\%0
      3o",
                                                        s[0], s[1], s[2], s[3], s[4]);
                                }
                                output_storage ("\"");
                        } else {
                                output_storage ("NULL");
                        }
                        output_storage ("};\n");
                }
        }

        if (field_cache) {
                output_storage ("\n/* Fields */\n");
                field_cache = list_cache_sort (field_cache, &field_cache_cmp);
                prevprog = NULL;
                for (k = field_cache; k; k = k->next) {
                        if (k->curr_prog != prevprog) {
                                prevprog = k->curr_prog;
                                output_storage ("\n/* PROGRAM-ID : %s */\n", prev
      prog);
                        }
                        output ("static cob_field %s%d\t= ", CB_PREFIX_FIELD, k->
      f->id);
                        if (!k->f->flag_local && !k->f->flag_item_external) {
                                output_field (k->x);
                        } else {
                                output ("{");
```

```
                        output_size (k->x);
                        output (", NULL, ");
                        output_attr (k->x);
                        output ("}");
                }
                output (";\t/* %s */\n", k->f->name);
        }
        output_storage ("\n/* End of fields */\n\n");
  }
  if (literal_cache) {
        output_storage ("/* Constants */\n");
        literal_cache = literal_list_reverse (literal_cache);
        for (m = literal_cache; m; m = m->next) {
                output ("static cob_field %s%d\t= ", CB_PREFIX_CONST, m->
id);
                output_field (m->x);
                output (";\n");
        }
        output ("\n");
  }

  if (gen_ebcdic) {
        output_storage ("/* EBCDIC translate table */\n");
        output ("static const unsigned char\tcob_a2e[256] = {\n");
        if (alt_ebcdic) {
                output ("\t0x00, 0x01, 0x02, 0x03, 0x37, 0x2D, 0x2E, 0x2F
,\n");
                output ("\t0x16, 0x05, 0x25, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
,\n");
                output ("\t0x10, 0x11, 0x12, 0x13, 0x3C, 0x3D, 0x32, 0x26
,\n");
                output ("\t0x18, 0x19, 0x3F, 0x27, 0x1C, 0x1D, 0x1E, 0x1F
,\n");
                output ("\t0x40, 0x5A, 0x7F, 0x7B, 0x5B, 0x6C, 0x50, 0x7D
,\n");
                output ("\t0x4D, 0x5D, 0x5C, 0x4E, 0x6B, 0x60, 0x4B, 0x61
,\n");
                output ("\t0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7
,\n");
                output ("\t0xF8, 0xF9, 0x7A, 0x5E, 0x4C, 0x7E, 0x6E, 0x6F
,\n");
                output ("\t0x7C, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7
,\n");
                output ("\t0xC8, 0xC9, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6
,\n");
                output ("\t0xD7, 0xD8, 0xD9, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6
,\n");
                output ("\t0xE7, 0xE8, 0xE9, 0xAD, 0xE0, 0xBD, 0x5F, 0x6D
,\n");
                output ("\t0x79, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87
,\n");
                output ("\t0x88, 0x89, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96
,\n");
                output ("\t0x97, 0x98, 0x99, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6
,\n");
                output ("\t0xA7, 0xA8, 0xA9, 0xC0, 0x6A, 0xD0, 0xA1, 0x07
,\n");
                output ("\t0x68, 0xDC, 0x51, 0x42, 0x43, 0x44, 0x47, 0x48
,\n");
                output ("\t0x52, 0x53, 0x54, 0x57, 0x56, 0x58, 0x63, 0x67
,\n");
                output ("\t0x71, 0x9C, 0x9E, 0xCB, 0xCC, 0xCD, 0xDB, 0xDD
```

```
,\n");
                output ("\t0xDF, 0xEC, 0xFC, 0xB0, 0xB1, 0xB2, 0x3E, 0xB4
,\n");
                output ("\t0x45, 0x55, 0xCE, 0xDE, 0x49, 0x69, 0x9A, 0x9B
,\n");
                output ("\t0xAB, 0x9F, 0xBA, 0xB8, 0xB7, 0xAA, 0x8A, 0x8B
,\n");
                output ("\t0xB6, 0xB5, 0x62, 0x4F, 0x64, 0x65, 0x66, 0x20
,\n");
                output ("\t0x21, 0x22, 0x70, 0x23, 0x72, 0x73, 0x74, 0xBE
,\n");
                output ("\t0x76, 0x77, 0x78, 0x80, 0x24, 0x15, 0x8C, 0x8D
,\n");
                output ("\t0x8E, 0x41, 0x06, 0x17, 0x28, 0x29, 0x9D, 0x2A
,\n");
                output ("\t0x2B, 0x2C, 0x09, 0x0A, 0xAC, 0x4A, 0xAE, 0xAF
,\n");
                output ("\t0x1B, 0x30, 0x31, 0xFA, 0x1A, 0x33, 0x34, 0x35
,\n");
                output ("\t0x36, 0x59, 0x08, 0x38, 0xBC, 0x39, 0xA0, 0xBF
,\n");
                output ("\t0xCA, 0x3A, 0xFE, 0x3B, 0x04, 0xCF, 0xDA, 0x14
,\n");
                output ("\t0xE1, 0x8F, 0x46, 0x75, 0xFD, 0xEB, 0xEE, 0xED
,\n");
                output ("\t0x90, 0xEF, 0xB3, 0xFB, 0xB9, 0xEA, 0xBB, 0xFF
\n");
        } else {
                /* MF */
                output ("\t0x00, 0x01, 0x02, 0x03, 0x1D, 0x19, 0x1A, 0x1B
,\n");
                output ("\t0x0F, 0x04, 0x16, 0x06, 0x07, 0x08, 0x09, 0x0A
,\n");
                output ("\t0x0B, 0x0C, 0x0D, 0x0E, 0x1E, 0x1F, 0x1C, 0x17
,\n");
                output ("\t0x10, 0x11, 0x20, 0x18, 0x12, 0x13, 0x14, 0x15
,\n");
                output ("\t0x21, 0x27, 0x3A, 0x36, 0x28, 0x30, 0x26, 0x38
,\n");
                output ("\t0x24, 0x2A, 0x29, 0x25, 0x2F, 0x2C, 0x22, 0x2D
,\n");
                output ("\t0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A
,\n");
                output ("\t0x7B, 0x7C, 0x35, 0x2B, 0x23, 0x39, 0x32, 0x33
,\n");
                output ("\t0x37, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D
,\n");
                output ("\t0x5E, 0x5F, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66
,\n");
                output ("\t0x67, 0x68, 0x69, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F
,\n");
                output ("\t0x70, 0x71, 0x72, 0x7D, 0x6A, 0x7E, 0x7F, 0x31
,\n");
                output ("\t0x34, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41
,\n");
                output ("\t0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49
,\n");
                output ("\t0x4A, 0x4B, 0x4C, 0x4E, 0x4F, 0x50, 0x51, 0x52
,\n");
                output ("\t0x53, 0x54, 0x55, 0x56, 0x2E, 0x60, 0x4D, 0x05
,\n");
                output ("\t0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87
```

```
,\n");
                output ("\t0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E, 0x8F
,\n");
                output ("\t0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97
,\n");
                output ("\t0x98, 0x99, 0x9A, 0x9B, 0x9C, 0x9D, 0x9E, 0x9F
,\n");
                output ("\t0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7
,\n");
                output ("\t0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF
,\n");
                output ("\t0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7
,\n");
                output ("\t0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF
,\n");
                output ("\t0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7
,\n");
                output ("\t0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF
,\n");
                output ("\t0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7
,\n");
                output ("\t0xD8, 0xD9, 0xDA, 0xDB, 0xDC, 0xDD, 0xDE, 0xDF
,\n");
                output ("\t0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7
,\n");
                output ("\t0xE8, 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 0xEE, 0xEF
,\n");
                output ("\t0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7
,\n");
                output ("\t0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF
\n");
        }
        output ("};\n");
        output_storage ("\n");
  }
  if (gen_full_ebcdic) {
        output ("static const unsigned char\tcob_ebcdic[256] = {\n");
        output ("\t0x00, 0x01, 0x02, 0x03, 0x37, 0x2D, 0x2E, 0x2F,\n");
        output ("\t0x16, 0x05, 0x25, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,\n");
        output ("\t0x10, 0x11, 0x12, 0x13, 0x3C, 0x3D, 0x32, 0x26,\n");
        output ("\t0x18, 0x19, 0x3F, 0x27, 0x1C, 0x1D, 0x1E, 0x1F,\n");
        output ("\t0x40, 0x5A, 0x7F, 0x7B, 0x5B, 0x6C, 0x50, 0x7D,\n");
        output ("\t0x4D, 0x5D, 0x5C, 0x4E, 0x6B, 0x60, 0x4B, 0x61,\n");
        output ("\t0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,\n");
        output ("\t0xF8, 0xF9, 0x7A, 0x5E, 0x4C, 0x7E, 0x6E, 0x6F,\n");
        output ("\t0x7C, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7,\n");
        output ("\t0xC8, 0xC9, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6,\n");
        output ("\t0xD7, 0xD8, 0xD9, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6,\n");
        output ("\t0xE7, 0xE8, 0xE9, 0xAD, 0xE0, 0xBD, 0x5F, 0x6D,\n");
        output ("\t0x79, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,\n");
        output ("\t0x88, 0x89, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96,\n");
        output ("\t0x97, 0x98, 0x99, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6,\n");
        output ("\t0xA7, 0xA8, 0xA9, 0xC0, 0x6A, 0xD0, 0xA1, 0x07,\n");
        output ("\t0x68, 0xDC, 0x51, 0x42, 0x43, 0x44, 0x47, 0x48,\n");
        output ("\t0x52, 0x53, 0x54, 0x57, 0x56, 0x58, 0x63, 0x67,\n");
        output ("\t0x71, 0x9C, 0x9E, 0xCB, 0xCC, 0xCD, 0xDB, 0xDD,\n");
        output ("\t0xDF, 0xEC, 0xFC, 0xB0, 0xB1, 0xB2, 0x3E, 0xB4,\n");
        output ("\t0x45, 0x55, 0xCE, 0xDE, 0x49, 0x69, 0x9A, 0x9B,\n");
        output ("\t0xAB, 0x9F, 0xBA, 0xB8, 0xB7, 0xAA, 0x8A, 0x8B,\n");
        output ("\t0xB6, 0xB5, 0x62, 0x4F, 0x64, 0x65, 0x66, 0x20,\n");
        output ("\t0x21, 0x22, 0x70, 0x23, 0x72, 0x73, 0x74, 0xBE,\n");
        output ("\t0x76, 0x77, 0x78, 0x80, 0x24, 0x15, 0x8C, 0x8D,\n");
```

```
            output ("\t0x8E, 0x41, 0x06, 0x17, 0x28, 0x29, 0x9D, 0x2A,\n");
            output ("\t0x2B, 0x2C, 0x09, 0x0A, 0xAC, 0x4A, 0xAE, 0xAF,\n");
            output ("\t0x1B, 0x30, 0x31, 0xFA, 0x1A, 0x33, 0x34, 0x35,\n");
            output ("\t0x36, 0x59, 0x08, 0x38, 0xBC, 0x39, 0xA0, 0xBF,\n");
            output ("\t0xCA, 0x3A, 0xFE, 0x3B, 0x04, 0xCF, 0xDA, 0x14,\n");
            output ("\t0xE1, 0x8F, 0x46, 0x75, 0xFD, 0xEB, 0xEE, 0xED,\n");
            output ("\t0x90, 0xEF, 0xB3, 0xFB, 0xB9, 0xEA, 0xBB, 0xFF\n");
            output ("};\n");
            i = lookup_attr (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL, 0);
            output
                ("static cob_field f_ebcdic = { 256, (unsigned char *)cob_ebc
dic, &%s%d };\n",
                 CB_PREFIX_ATTR, i);
            output_storage ("\n");
  }
  if (gen_ebcdic_ascii) {
            output ("static const unsigned char\tcob_ebcdic_ascii[256] = {\n"
);
            output ("\t0x00, 0x01, 0x02, 0x03, 0xEC, 0x09, 0xCA, 0x7F,\n");
            output ("\t0xE2, 0xD2, 0xD3, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,\n");
            output ("\t0x10, 0x11, 0x12, 0x13, 0xEF, 0xC5, 0x08, 0xCB,\n");
            output ("\t0x18, 0x19, 0xDC, 0xD8, 0x1C, 0x1D, 0x1E, 0x1F,\n");
            output ("\t0xB7, 0xB8, 0xB9, 0xBB, 0xC4, 0x0A, 0x17, 0x1B,\n");
            output ("\t0xCC, 0xCD, 0xCF, 0xD0, 0xD1, 0x05, 0x06, 0x07,\n");
            output ("\t0xD9, 0xDA, 0x16, 0xDD, 0xDE, 0xDF, 0xE0, 0x04,\n");
            output ("\t0xE3, 0xE5, 0xE9, 0xEB, 0x14, 0x15, 0x9E, 0x1A,\n");
            output ("\t0x20, 0xC9, 0x83, 0x84, 0x85, 0xA0, 0xF2, 0x86,\n");
            output ("\t0x87, 0xA4, 0xD5, 0x2E, 0x3C, 0x28, 0x2B, 0xB3,\n");
            output ("\t0x26, 0x82, 0x88, 0x89, 0x8A, 0xA1, 0x8C, 0x8B,\n");
            output ("\t0x8D, 0xE1, 0x21, 0x24, 0x2A, 0x29, 0x3B, 0x5E,\n");
            output ("\t0x2D, 0x2F, 0xB2, 0x8E, 0xB4, 0xB5, 0xB6, 0x8F,\n");
            output ("\t0x80, 0xA5, 0x7C, 0x2C, 0x25, 0x5F, 0x3E, 0x3F,\n");
            output ("\t0xBA, 0x90, 0xBC, 0xBD, 0xBE, 0xF3, 0xC0, 0xC1,\n");
            output ("\t0xC2, 0x60, 0x3A, 0x23, 0x40, 0x27, 0x3D, 0x22,\n");
            output ("\t0xC3, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,\n");
            output ("\t0x68, 0x69, 0xAE, 0xAF, 0xC6, 0xC7, 0xC8, 0xF1,\n");
            output ("\t0xF8, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F, 0x70,\n");
            output ("\t0x71, 0x72, 0xA6, 0xA7, 0x91, 0xCE, 0x92, 0xA9,\n");
            output ("\t0xE6, 0x7E, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,\n");
            output ("\t0x79, 0x7A, 0xAD, 0xA8, 0xD4, 0x5B, 0xD6, 0xD7,\n");
            output ("\t0x9B, 0x9C, 0x9D, 0xFA, 0x9F, 0xB1, 0xB0, 0xAC,\n");
            output ("\t0xAB, 0xFC, 0xAA, 0xFE, 0xE4, 0x5D, 0xBF, 0xE7,\n");
            output ("\t0x7B, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47,\n");
            output ("\t0x48, 0x49, 0xE8, 0x93, 0x94, 0x95, 0xA2, 0xED,\n");
            output ("\t0x7D, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F, 0x50,\n");
            output ("\t0x51, 0x52, 0xEE, 0x96, 0x81, 0x97, 0xA3, 0x98,\n");
            output ("\t0x5C, 0xF0, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,\n");
            output ("\t0x59, 0x5A, 0xFD, 0xF5, 0x99, 0xF7, 0xF6, 0xF9,\n");
            output ("\t0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,\n");
            output ("\t0x38, 0x39, 0xDB, 0xFB, 0x9A, 0xF4, 0xEA, 0xFF\n");
            output ("};\n");
            i = lookup_attr (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL, 0);
            output
                ("static cob_field f_ebcdic_ascii = { 256, (unsigned char *)c
ob_ebcdic_ascii, &%s%d };\n",
                 CB_PREFIX_ATTR, i);
            output_storage ("\n");
  }
  if (gen_native) {
            output ("static const unsigned char\tcob_native[256] = {\n");
            output ("\t0, 1, 2, 3, 4, 5, 6, 7,\n");
            output ("\t8, 9, 10, 11, 12, 13, 14, 15,\n");
```

```
                output ("\t16, 17, 18, 19, 20, 21, 22, 23,\n");
                output ("\t24, 25, 26, 27, 28, 29, 30, 31,\n");
                output ("\t32, 33, 34, 35, 36, 37, 38, 39,\n");
                output ("\t40, 41, 42, 43, 44, 45, 46, 47,\n");
                output ("\t48, 49, 50, 51, 52, 53, 54, 55,\n");
                output ("\t56, 57, 58, 59, 60, 61, 62, 63,\n");
                output ("\t64, 65, 66, 67, 68, 69, 70, 71,\n");
                output ("\t72, 73, 74, 75, 76, 77, 78, 79,\n");
                output ("\t80, 81, 82, 83, 84, 85, 86, 87,\n");
                output ("\t88, 89, 90, 91, 92, 93, 94, 95,\n");
                output ("\t96, 97, 98, 99, 100, 101, 102, 103,\n");
                output ("\t104, 105, 106, 107, 108, 109, 110, 111,\n");
                output ("\t112, 113, 114, 115, 116, 117, 118, 119,\n");
                output ("\t120, 121, 122, 123, 124, 125, 126, 127,\n");
                output ("\t128, 129, 130, 131, 132, 133, 134, 135,\n");
                output ("\t136, 137, 138, 139, 140, 141, 142, 143,\n");
                output ("\t144, 145, 146, 147, 148, 149, 150, 151,\n");
                output ("\t152, 153, 154, 155, 156, 157, 158, 159,\n");
                output ("\t160, 161, 162, 163, 164, 165, 166, 167,\n");
                output ("\t168, 169, 170, 171, 172, 173, 174, 175,\n");
                output ("\t176, 177, 178, 179, 180, 181, 182, 183,\n");
                output ("\t184, 185, 186, 187, 188, 189, 190, 191,\n");
                output ("\t192, 193, 194, 195, 196, 197, 198, 199,\n");
                output ("\t200, 201, 202, 203, 204, 205, 206, 207,\n");
                output ("\t208, 209, 210, 211, 212, 213, 214, 215,\n");
                output ("\t216, 217, 218, 219, 220, 221, 222, 223,\n");
                output ("\t224, 225, 226, 227, 228, 229, 230, 231,\n");
                output ("\t232, 233, 234, 235, 236, 237, 238, 239,\n");
                output ("\t240, 241, 242, 243, 244, 245, 246, 247,\n");
                output ("\t248, 249, 250, 251, 252, 253, 254, 255\n");
                output ("};\n");
                i = lookup_attr (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL, 0);
                output
                    ("static cob_field f_native = { 256, (unsigned char *)cob_nat
      ive, &%s%d };\n",
                     CB_PREFIX_ATTR, i);
                output_storage ("\n");
          }
}
```

7.16.4.199 void finalize_file (struct cb_file * *f*, struct cb_field * *records*)

Definition at line 1611 of file tree.c.

```
{
        struct cb_field *p;
        struct cb_field *v;
        cb_tree         l;
        cb_tree         x;
        char            buff[COB_MINI_BUFF];

        if (f->special) {
                f->organization = COB_ORG_LINE_SEQUENTIAL;
        }
        if (f->fileid_assign && !f->assign) {
                f->assign = cb_build_alphanumeric_literal ((unsigned char *)f->
      name,
                                                           strlen (f->name));
        }
```

```
  /* check the record size if it is limited */
  for (p = records; p; p = p->sister) {
          if (f->record_min > 0) {
                  if (p->size < f->record_min) {
                          cb_error (_("Record size too small '%s'"), p->
name);
                  }
          }
          if (f->record_max > 0) {
                  if (p->size > f->record_max) {
                          cb_error (_("Record size too large '%s' (%d)"),
                                  p->name, p->size);
                  }
          }
  }

  /* compute the record size */
  if (f->record_min == 0) {
          if (records) {
                  f->record_min = records->size;
          } else {
                  f->record_min = 0;
          }
  }
  for (p = records; p; p = p->sister) {
          v = cb_field_variable_size (p);
          if (v && v->offset + v->size * v->occurs_min < f->record_min) {
                  f->record_min = v->offset + v->size * v->occurs_min;
          }
          if (p->size < f->record_min) {
                  f->record_min = p->size;
          }
          if (p->size > f->record_max) {
                  f->record_max = p->size;
          }
  }

  if (f->same_clause) {
          for (l = current_program->file_list; l; l = CB_CHAIN (l)) {
                  if (CB_FILE (CB_VALUE (l))->same_clause == f->
same_clause) {
                          if (CB_FILE (CB_VALUE (l))->finalized) {
                                  if (f->record_max > CB_FILE (CB_VALUE (l)
)->record->memory_size) {
                                          CB_FILE (CB_VALUE (l))->record->m
emory_size =
                                              f->record_max;
                                  }
                                  f->record = CB_FILE (CB_VALUE (l))->recor
d;
                                  for (p = records; p; p = p->sister) {
                                          p->file = f;
                                          p->redefines = f->record;
                                  }
                                  for (p = f->record->sister; p; p = p->
sister) {
                                          if (!p->sister) {
                                                  p->sister = records;
                                                  break;
                                          }
                                  }
```

```
                                        f->finalized = 1;
                                        return;
                                }
                        }
                }
        }
        /* create record */
        snprintf (buff, COB_MINI_MAX, "%s_record", f->name);
        if (f->record_max == 0) {
                f->record_max = 32;
                f->record_min = 32;
        }
        if (f->organization == COB_ORG_LINE_SEQUENTIAL) {
                f->record_min = 0;
        }
        f->record = CB_FIELD (cb_build_implicit_field (cb_build_reference (buff),

                              f->record_max));
        f->record->sister = records;
        f->record->count++;
        if (f->external) {
                has_external = 1;
                f->record->flag_external = 1;
        }

        for (p = records; p; p = p->sister) {
                p->file = f;
                p->redefines = f->record;
        }
        f->finalized = 1;
        if (f->linage) {
                snprintf (buff, COB_MINI_MAX, "LC_%s", f->name);
                x = cb_build_field (cb_build_reference (buff));
                CB_FIELD (x)->pic = CB_PICTURE (cb_build_picture ("9(9)"));
                CB_FIELD (x)->usage = CB_USAGE_COMP_5;
                CB_FIELD (x)->values = cb_list_init (cb_zero);
                CB_FIELD (x)->count++;
                cb_validate_field (CB_FIELD (x));
                f->linage_ctr = cb_build_field_reference (CB_FIELD (x), NULL);
                current_program->working_storage =
                    cb_field_add (current_program->working_storage, CB_FIELD (x))
      ;
        }
}
```

7.16.4.200 void group_error (cb_tree *x*, const char * *clause*)

Definition at line 250 of file error.c.

```
{
        cb_error_x (x, _("Group item '%s' cannot have %s clause"),
      check_filler_name (cb_name (x)), clause);
}
```

7.16.4.201 void level_except_error (cb_tree *x*, const char * *clause*)

Definition at line 270 of file error.c.

```
{
        cb_error_x (x, _("Level %02d item '%s' cannot have other than %s clause")
      ,
                    cb_field (x)->level, check_filler_name (cb_name (x)), clause)
      ;
}
```

7.16.4.202 void level_redundant_error (cb_tree *x*, const char * *clause*)

Definition at line 256 of file error.c.

```
{
        cb_error_x (x, _("Level %02d item '%s' cannot have %s clause"),
                    cb_field (x)->level, check_filler_name (cb_name (x)), clause)
      ;
}
```

7.16.4.203 void level_require_error (cb_tree *x*, const char * *clause*)

Definition at line 263 of file error.c.

```
{
        cb_error_x (x, _("Level %02d item '%s' requires %s clause"),
                    cb_field (x)->level, check_filler_name (cb_name (x)), clause)
      ;
}
```

7.16.4.204 struct cb_intrinsic_table* lookup_intrinsic (const char * *name*, const int *checkres*) [read]

Definition at line 929 of file reserved.c.

```
{
        struct cb_intrinsic_table       *cbp;
        struct noreserve                *noresptr;

        if (checkres) {
                for (noresptr = norestab; noresptr; noresptr = noresptr->next) {
                        if (strcasecmp (name, noresptr->noresword) == 0) {
                                return NULL;
                        }
                }
        }
        cbp = bsearch (name, function_list, NUM_INTRINSICS,
                       sizeof (struct cb_intrinsic_table), intrinsic_comp);
        if (cbp && cbp->implemented) {
                return cbp;
        }
        return NULL;
}
```

7.16.4.205 int lookup_reserved_word (const char * *name*)

Definition at line 906 of file reserved.c.

```
{
        struct reserved *p;
        struct noreserve        *noresptr;

        p = bsearch (name, reserved_words, NUM_RESERVED_WORDS,
                     sizeof (struct reserved), reserve_comp);
        if (!p) {
                return 0;
        }
        for (noresptr = norestab; noresptr; noresptr = noresptr->next) {
                if (strcasecmp (name, noresptr->noresword) == 0) {
                        return 0;
                }
        }
        if (p->token != -1) {
                return p->token;
        }
        cb_error (_("'%s' reserved word, but not supported yet"), name);
        return 0;
}
```

7.16.4.206 cb_tree lookup_system_name (const char * *name*)

Definition at line 893 of file reserved.c.

```
{
        int     i;

        for (i = 0; system_table[i].name != NULL; ++i) {
                if (strcasecmp (name, system_table[i].name) == 0) {
                        return system_table[i].node;
                }
        }
        return cb_error_node;
}
```

7.16.4.207 void redefinition_error (cb_tree *x*)

Definition at line 154 of file error.c.

```
{
        struct cb_word  *w;

        w = CB_REFERENCE (x)->word;
        cb_error_x (x, _("Redefinition of '%s'"), w->name);
        cb_error_x (CB_VALUE (w->items), _("'%s' previously defined here"), w->
      name);
}
```

7.16.4.208 void redefinition_warning (cb_tree *x*, cb_tree *y*)

Definition at line 164 of file error.c.

```
{
        struct cb_word  *w;

        w = CB_REFERENCE (x)->word;
        cb_warning_x (x, _("Redefinition of '%s'"), w->name);
        if (y) {
                cb_warning_x (y, _("'%s' previously defined here"), w->name);
        } else {
                cb_warning_x (CB_VALUE (w->items), _("'%s' previously defined her
      e"), w->name);
        }
}
```

7.16.4.209 void undefined_error (cb_tree *x*)

Definition at line 178 of file error.c.

```
{
        struct cb_reference     *r;
        cb_tree                 c;

        if (!errnamebuff) {
                errnamebuff = cobc_malloc (COB_NORMAL_BUFF);
        }
        r = CB_REFERENCE (x);
        snprintf (errnamebuff, COB_NORMAL_MAX, "'%s'", CB_NAME (x));
        for (c = r->chain; c; c = CB_REFERENCE (c)->chain) {
                strcat (errnamebuff, " in '");
                strcat (errnamebuff, CB_NAME (c));
                strcat (errnamebuff, "'");
        }
        cb_error_x (x, _("%s undefined"), errnamebuff);
}
```

7.16.4.210 void validate_file (struct cb_file * *f*, cb_tree *name*)

Definition at line 1593 of file tree.c.

```
{
        /* check RECORD/RELATIVE KEY clause */
        switch (f->organization) {
        case COB_ORG_INDEXED:
                if (f->key == NULL) {
                        file_error (name, "RECORD KEY");
                }
                break;
        case COB_ORG_RELATIVE:
                if (f->key == NULL && f->access_mode != COB_ACCESS_SEQUENTIAL) {
                        file_error (name, "RELATIVE KEY");
                }
```

```
                break;
        }
}
```

7.16.4.211 int validate_move (cb_tree *src*, cb_tree *dst*, size_t *is_value*)

Definition at line 3944 of file typeck.c.

```
{
        struct cb_field         *f;
        struct cb_literal       *l;
        unsigned char           *p;
        cb_tree                 loc;
        long long               val;
        size_t                  i;
        size_t                  is_numeric_edited = 0;
        int                     src_scale_mod;
        int                     dst_scale_mod;
        int                     dst_size_mod;
        int                     size;
        int                     most_significant;
        int                     least_significant;

        loc = src->source_line ? src : dst;
        if (CB_REFERENCE_P(dst) && CB_ALPHABET_NAME_P(CB_REFERENCE(dst)->value))
      {
                goto invalid;
        }
        if (CB_REFERENCE_P(dst) && CB_FILE_P(CB_REFERENCE(dst)->value)) {
                goto invalid;
        }
        if (CB_TREE_CATEGORY (dst) == CB_CATEGORY_BOOLEAN) {
                cb_error_x (loc, _("Invalid destination for MOVE"));
                return -1;
        }

        if (CB_TREE_CLASS (dst) == CB_CLASS_POINTER) {
                if (CB_TREE_CLASS (src) == CB_CLASS_POINTER) {
                        return 0;
                } else {
                        goto invalid;
                }
        }

        f = cb_field (dst);
        switch (CB_TREE_TAG (src)) {
        case CB_TAG_CONST:
                if (src == cb_space) {
                        if (CB_TREE_CATEGORY (dst) == CB_CATEGORY_NUMERIC
                            || (CB_TREE_CATEGORY (dst) ==
      CB_CATEGORY_NUMERIC_EDITED && !is_value)) {
                                goto invalid;
                        }
                } else if (src == cb_zero) {
                        if (CB_TREE_CATEGORY (dst) == CB_CATEGORY_ALPHABETIC) {
                                goto invalid;
                        }
                }
                break;
```

```
	case CB_TAG_LITERAL:
		/* TODO: ALL literal */

		l = CB_LITERAL (src);
		if (CB_TREE_CLASS (src) == CB_CLASS_NUMERIC) {
			/* Numeric literal */
			if (l->all) {
				goto invalid;
			}
			most_significant = -999;
			least_significant = 999;

			/* compute the most significant figure place */
			for (i = 0; i < l->size; i++) {
				if (l->data[i] != '0') {
					break;
				}
			}
			if (i != l->size) {
				most_significant = (int) (l->size - l->scale - i
- 1);
			}

			/* compute the least significant figure place */
			for (i = 0; i < l->size; i++) {
				if (l->data[l->size - i - 1] != '0') {
					break;
				}
			}
			if (i != l->size) {
				least_significant = (int) (-l->scale + i);
			}

			/* value check */
			switch (CB_TREE_CATEGORY (dst)) {
			case CB_CATEGORY_ALPHANUMERIC:
			case CB_CATEGORY_ALPHANUMERIC_EDITED:
				if (is_value) {
					goto expect_alphanumeric;
				}

				if (l->scale == 0) {
					goto expect_alphanumeric;
				} else {
					goto invalid;
				}
			case CB_CATEGORY_NUMERIC:
				if (f->pic->scale < 0) {
					/* check for PIC 9(n)P(m) */
					if (least_significant < -f->pic->scale) {

						goto value_mismatch;
					}
				} else if (f->pic->scale > f->pic->size) {
					/* check for PIC P(n)9(m) */
					if (most_significant >= f->pic->size - f-
>pic->scale) {
						goto value_mismatch;
					}
				}
				break;
			case CB_CATEGORY_NUMERIC_EDITED:
```

```
                        if (is_value) {
                                goto expect_alphanumeric;
                        }

                        /* TODO */
                        break;
                default:
                        if (is_value) {
                                goto expect_alphanumeric;
                        }
                        goto invalid;
                }

                /* sign check */
                if (l->sign != 0 && !f->pic->have_sign) {
                        if (is_value) {
                                cb_error_x (loc, _("Data item not signed"
));
                                return -1;
                        }
                        if (cb_warn_constant) {
                                cb_warning_x (loc, _("Ignoring negative s
ign"));
                        }
                }

                /* size check */
                if (f->flag_real_binary ||
                    ((f->usage == CB_USAGE_COMP_5 ||
                      f->usage == CB_USAGE_COMP_X ||
                      f->usage == CB_USAGE_BINARY) &&
                      f->pic->scale == 0)) {
                        p = l->data;
                        for (i = 0; i < l->size; i++) {
                                if (l->data[i] != '0') {
                                        p = &l->data[i];
                                        break;
                                }
                        }
                        i = l->size - i;
                        switch (f->size) {
                        case 1:
                                if (i > 18) {
                                        goto numlit_overflow;
                                }
                                val = cb_get_long_long (src);
                                if (f->pic->have_sign) {
                                        if (val < -128LL ||
                                            val > 127LL) {
                                                goto numlit_overflow;
                                        }
                                } else {
                                        if (val > 255LL) {
                                                goto numlit_overflow;
                                        }
                                }
                                break;
                        case 2:
                                if (i > 18) {
                                        goto numlit_overflow;
                                }
                                val = cb_get_long_long (src);
```

```
        if (f->pic->have_sign) {
                if (val < -32768LL ||
                    val > 32767LL) {
                        goto numlit_overflow;
                }
        } else {
                if (val > 65535LL) {
                        goto numlit_overflow;
                }
        }
        break;
case 3:
        if (i > 18) {
                goto numlit_overflow;
        }
        val = cb_get_long_long (src);
        if (f->pic->have_sign) {
                if (val < -8388608LL ||
                    val > 8388607LL) {
                        goto numlit_overflow;
                }
        } else {
                if (val > 16777215LL) {
                        goto numlit_overflow;
                }
        }
        break;
case 4:
        if (i > 18) {
                goto numlit_overflow;
        }
        val = cb_get_long_long (src);
        if (f->pic->have_sign) {
                if (val < -2147483648LL ||
                    val > 2147483647LL) {
                        goto numlit_overflow;
                }
        } else {
                if (val > 4294967295LL) {
                        goto numlit_overflow;
                }
        }
        break;
case 5:
        if (i > 18) {
                goto numlit_overflow;
        }
        val = cb_get_long_long (src);
        if (f->pic->have_sign) {
                if (val < -549755813888LL ||
                    val > 549755813887LL) {
                        goto numlit_overflow;
                }
        } else {
                if (val > 1099511627775LL) {
                        goto numlit_overflow;
                }
        }
        break;
case 6:
        if (i > 18) {
                goto numlit_overflow;
```

```
                                        }
                                        val = cb_get_long_long (src);
                                        if (f->pic->have_sign) {
                                                if (val < -140737488355328LL ||
                                                    val > 140737488355327LL) {
                                                        goto numlit_overflow;
                                                }
                                        } else {
                                                if (val > 281474976710655LL) {
                                                        goto numlit_overflow;
                                                }
                                        }
                                        break;
                                case 7:
                                        if (i > 18) {
                                                goto numlit_overflow;
                                        }
                                        val = cb_get_long_long (src);
                                        if (f->pic->have_sign) {
                                                if (val < -36028797018963968LL ||

                                                    val > 36028797018963967LL) {
                                                        goto numlit_overflow;
                                                }
                                        } else {
                                                if (val > 72057594037927935LL) {
                                                        goto numlit_overflow;
                                                }
                                        }
                                        break;
                                default:
                                        if (f->pic->have_sign) {
                                                if (i < 19) {
                                                        break;
                                                }
                                                if (i > 19) {
                                                        goto numlit_overflow;
                                                }
                                                if (memcmp (p, "92233720368547758
07", 19) > 0) {
                                                        goto numlit_overflow;
                                                }
                                        } else {
                                                if (i < 20) {
                                                        break;
                                                }
                                                if (i > 20) {
                                                        goto numlit_overflow;
                                                }
                                                if (memcmp (p, "18446744073709551
615", 20) > 0) {
                                                        goto numlit_overflow;
                                                }
                                        }
                                        break;
                                }
                                return 0;
                        }
                        if (least_significant < -f->pic->scale) {
                                goto size_overflow;
                        }
                        if (f->pic->scale > 0) {
```

```
                        size = f->pic->digits - f->pic->scale;
                } else {
                        size = f->pic->digits;
                }
                if (most_significant >= size) {
                        goto size_overflow;
                }
        } else {
                /* Alphanumeric literal */

                /* value check */
                switch (CB_TREE_CATEGORY (dst)) {
                case CB_CATEGORY_ALPHABETIC:
                        for (i = 0; i < l->size; i++) {
                                if (!isalpha (l->data[i]) && !isspace (l-
>data[i])) {
                                        goto value_mismatch;
                                }
                        }
                        break;
                case CB_CATEGORY_NUMERIC:
                        goto expect_numeric;
                case CB_CATEGORY_NUMERIC_EDITED:
                        if (!is_value) {
                                goto expect_numeric;
                        }

                        /* TODO: validate the value */
                        break;
                default:
                        break;
                }

                /* size check */
                size = cb_field_size (dst);
                if (size >= 0 && (int)l->size > size) {
                        goto size_overflow;
                }
        }
        break;
  case CB_TAG_FIELD:
  case CB_TAG_REFERENCE:
        if (CB_REFERENCE_P(src) &&
            CB_ALPHABET_NAME_P(CB_REFERENCE(src)->value)) {
                break;
        }
        if (CB_REFERENCE_P(src) &&
            CB_FILE_P(CB_REFERENCE(src)->value)) {
                goto invalid;
        }
        size = cb_field_size (src);
        if (size < 0) {
                size = cb_field (src)->size;
        }
        /* non-elementary move */
        if (cb_field (src)->children || cb_field (dst)->children) {
                if (size > cb_field (dst)->size) {
                        goto size_overflow_1;
                }
                break;
        }
```

```
		/* elementary move */
		switch (CB_TREE_CATEGORY (src)) {
		case CB_CATEGORY_ALPHANUMERIC:
			switch (CB_TREE_CATEGORY (dst)) {
			case CB_CATEGORY_NUMERIC:
			case CB_CATEGORY_NUMERIC_EDITED:
				if (size > cb_field (dst)->pic->digits) {
					goto size_overflow_2;
				}
				break;
			case CB_CATEGORY_ALPHANUMERIC_EDITED:
				if (size >
				    count_pic_alphanumeric_edited (cb_field (dst)
)) {
					goto size_overflow_1;
				}
				break;
			default:
				if (size > cb_field (dst)->size) {
					goto size_overflow_1;
				}
				break;
			}
			break;
		case CB_CATEGORY_ALPHABETIC:
		case CB_CATEGORY_ALPHANUMERIC_EDITED:
			switch (CB_TREE_CATEGORY (dst)) {
			case CB_CATEGORY_NUMERIC:
			case CB_CATEGORY_NUMERIC_EDITED:
				goto invalid;
			case CB_CATEGORY_ALPHANUMERIC_EDITED:
				if (size >
				    count_pic_alphanumeric_edited(cb_field (dst))
) {
					goto size_overflow_1;
				}
				break;
			default:
				if (size > cb_field (dst)->size) {
					goto size_overflow_1;
				}
				break;
			}
			break;
		case CB_CATEGORY_NUMERIC:
		case CB_CATEGORY_NUMERIC_EDITED:
			switch (CB_TREE_CATEGORY (dst)) {
			case CB_CATEGORY_ALPHABETIC:
				goto invalid;
			case CB_CATEGORY_ALPHANUMERIC_EDITED:
				is_numeric_edited = 1;
				/* Drop through */
			case CB_CATEGORY_ALPHANUMERIC:
				if (is_numeric_edited) {
					dst_size_mod = count_pic_alphanumeric_edi
ted (cb_field (dst));
				} else {
					dst_size_mod = cb_field (dst)->size;
				}
				if (CB_TREE_CATEGORY (src) ==
CB_CATEGORY_NUMERIC
				    && cb_field (src)->pic->scale > 0) {
```

```
                                        if (cb_move_noninteger_to_alphanumeric ==
       CB_ERROR) {
                                                goto invalid;
                                        }
                                        cb_warning_x (loc, _("Move non-integer to
       alphanumeric"));
                                        break;
                                }
                                if (CB_TREE_CATEGORY (src) ==
      CB_CATEGORY_NUMERIC
                                    && cb_field (src)->pic->digits > dst_size_mod
      ) {
                                        goto size_overflow_2;
                                }
                                if (CB_TREE_CATEGORY (src) ==
      CB_CATEGORY_NUMERIC_EDITED
                                    && cb_field (src)->size > dst_size_mod) {
                                        goto size_overflow_1;
                                }
                                break;
                        default:
                                src_scale_mod = cb_field (src)->pic->scale < 0 ?
                                                0 : cb_field (src)->pic->scale;
                                dst_scale_mod = cb_field (dst)->pic->scale < 0 ?
                                                0 : cb_field (dst)->pic->scale;
                                if (cb_field (src)->pic->digits - src_scale_mod >

                                    cb_field (dst)->pic->digits - dst_scale_mod
      ||
                                    src_scale_mod > dst_scale_mod) {
                                        goto size_overflow_2;
                                }
                                break;
                        }
                        break;
                default:
                        cb_error_x (loc, _("Invalid source for MOVE"));
                        return -1;
                }
                break;
        case CB_TAG_INTEGER:
        case CB_TAG_BINARY_OP:
        case CB_TAG_INTRINSIC:
                /* TODO: check this */
                break;
        default:
                fprintf (stderr, "Invalid tree tag %d\n", CB_TREE_TAG (src));
                ABORT ();
        }
        return 0;

invalid:
        if (is_value) {
                cb_error_x (loc, _("Invalid VALUE clause"));
        } else {
                cb_error_x (loc, _("Invalid MOVE statement"));
        }
        return -1;

numlit_overflow:
        if (is_value) {
                cb_error_x (loc, _("Invalid VALUE clause - literal exceeds data s
```

```
      ize"));
                return -1;
          }
          if (cb_warn_constant) {
                cb_warning_x (loc, _("Numeric literal exceeds data size"));
          }
          return 0;

expect_numeric:
          return move_error (src, dst, is_value, cb_warn_strict_typing, 0,
                             _("Numeric value is expected"));

expect_alphanumeric:
          return move_error (src, dst, is_value, cb_warn_strict_typing, 0,
                             _("Alphanumeric value is expected"));

value_mismatch:
          return move_error (src, dst, is_value, cb_warn_constant, 0,
                             _("Value does not fit the picture string"));

size_overflow:
          return move_error (src, dst, is_value, cb_warn_constant, 0,
                             _("Value size exceeds data size"));

size_overflow_1:
          return move_error (src, dst, is_value, cb_warn_truncate, 1,
                             _("Sending field larger than receiving field"));

size_overflow_2:
          return move_error (src, dst, is_value, cb_warn_truncate, 1,
                             _("Some digits may be truncated"));
}
```

7.16.5 Variable Documentation

7.16.5.1 cb_tree cb_any

Definition at line 74 of file tree.c.

7.16.5.2 cb_tree cb_error_node

Definition at line 93 of file tree.c.

7.16.5.3 cb_tree cb_false

Definition at line 76 of file tree.c.

7.16.5.4 cb_tree cb_high

Definition at line 82 of file tree.c.

7.16.5.5 cb_tree cb_i[8]

Definition at line 92 of file tree.c.

7.16.5.6 cb_tree cb_int0

Definition at line 86 of file tree.c.

7.16.5.7 cb_tree cb_int1

Definition at line 87 of file tree.c.

7.16.5.8 cb_tree cb_int2

Definition at line 88 of file tree.c.

7.16.5.9 cb_tree cb_int3

Definition at line 89 of file tree.c.

7.16.5.10 cb_tree cb_int4

Definition at line 90 of file tree.c.

7.16.5.11 cb_tree cb_int5

Definition at line 91 of file tree.c.

7.16.5.12 cb_tree cb_intr_e

Definition at line 97 of file tree.c.

7.16.5.13 cb_tree cb_intr_pi

Definition at line 96 of file tree.c.

7.16.5.14 cb_tree cb_intr_whencomp

Definition at line 95 of file tree.c.

7.16.5.15 cb_tree cb_low

Definition at line 81 of file tree.c.

7.16.5.16 size_t cb_needs_01

Definition at line 33 of file field.c.

7.16.5.17 cb_tree cb_norm_high

Definition at line 84 of file tree.c.

7.16.5.18 cb_tree cb_norm_low

Definition at line 83 of file tree.c.

7.16.5.19 cb_tree cb_null

Definition at line 77 of file tree.c.

7.16.5.20 cb_tree cb_one

Definition at line 79 of file tree.c.

7.16.5.21 cb_tree cb_quote

Definition at line 85 of file tree.c.

7.16.5.22 cb_tree cb_space

Definition at line 80 of file tree.c.

7.16.5.23 cb_tree cb_standard_error_handler

Definition at line 99 of file tree.c.

7.16.5.24 cb_tree cb_true

Definition at line 75 of file tree.c.

7.16.5.25 cb_tree cb_zero

Definition at line 78 of file tree.c.

7.16.5.26 size_t gen_screen_ptr

Definition at line 101 of file tree.c.

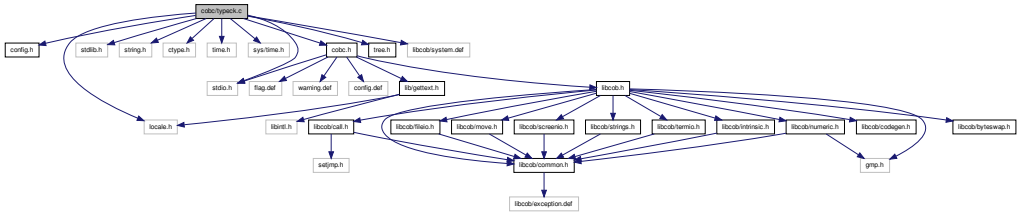
7.16.5.27 int non_const_word

Definition at line 988 of file parser.c.

7.17 cobc/typeck.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <sys/time.h>
#include <locale.h>
#include "cobc.h"
#include "tree.h"
#include "libcob/system.def"
```

Include dependency graph for typeck.c:



Classes

- struct system_table

- struct expr_node

Defines

- #define START_STACK_SIZE 32
- #define TOKEN(offset) (expr_stack[expr_index + offset].token)
- #define VALUE(offset) (expr_stack[expr_index + offset].value)
- #define dpush(x) decimal_stack = cb_cons (x, decimal_stack)
- #define cb_emit(x) current_statement->body = cb_list_add (current_statement->body, x)
- #define cb_emit_list(l) current_statement->body = cb_list_append (current_statement->body, l)
- #define COB_SYSTEM_GEN(x, y, z) { x, y },

Functions

- cb_tree cb_check_numeric_value (cb_tree x)
- void cb_build_registers (void)
- char * cb_encode_program_id (const char *name)
- const char * cb_build_program_id (cb_tree name, cb_tree alt_name)
- void cb_define_switch_name (cb_tree name, cb_tree sname, cb_tree flag, cb_tree ref)
- cb_tree cb_build_section_name (cb_tree name, int sect_or_para)
- cb_tree cb_build_assignment_name (struct cb_file *cfile, cb_tree name)
- cb_tree cb_build_index (cb_tree x, cb_tree values, int indexed_by, struct cb_field *qual)
- cb_tree cb_build_identifier (cb_tree x)
- cb_tree cb_build_const_length (cb_tree x)
- cb_tree cb_build_length (cb_tree x)
- cb_tree cb_build_address (cb_tree x)
- cb_tree cb_build_ppointer (cb_tree x)
- void cb_validate_program_environment (struct cb_program *prog)
- void cb_validate_program_data (struct cb_program *prog)
- void cb_validate_program_body (struct cb_program *prog)
- cb_tree cb_build_expr (cb_tree list)
- void cb_emit_arithmetic (cb_tree vars, int op, cb_tree val)
- cb_tree cb_build_cond (cb_tree x)
- cb_tree cb_build_add (cb_tree v, cb_tree n, cb_tree round_opt)
- cb_tree cb_build_sub (cb_tree v, cb_tree n, cb_tree round_opt)
- void cb_emit_corresponding (cb_tree(*func)(cb_tree f1, cb_tree f2, cb_tree f3), cb_tree x1, cb_tree x2, cb_tree opt)
- void cb_emit_move_corresponding (cb_tree x1, cb_tree x2)
- void cb_emit_accept (cb_tree var, cb_tree pos, cb_tree fgc, cb_tree bgc, cb_tree scroll, int dispattrs)
- void cb_emit_accept_line_or_col (cb_tree var, const int l_or_c)
- void cb_emit_accept_date (cb_tree var)

- void cb_emit_accept_date_yyyymmdd (cb_tree var)
- void cb_emit_accept_day (cb_tree var)
- void cb_emit_accept_day_yyyyddd (cb_tree var)
- void cb_emit_accept_day_of_week (cb_tree var)
- void cb_emit_accept_time (cb_tree var)
- void cb_emit_accept_command_line (cb_tree var)
- void cb_emit_get_environment (cb_tree envvar, cb_tree envval)
- void cb_emit_accept_environment (cb_tree var)
- void cb_emit_accept_arg_number (cb_tree var)
- void cb_emit_accept_arg_value (cb_tree var)
- void cb_emit_accept_mnemonic (cb_tree var, cb_tree mnemonic)
- void cb_emit_accept_name (cb_tree var, cb_tree name)
- void cb_emit_allocate (cb_tree target1, cb_tree target2, cb_tree size, cb_tree initialize)
- void cb_emit_call (cb_tree prog, cb_tree using, cb_tree returning, cb_tree on_exception, cb_tree not_on_exception)
- void cb_emit_cancel (cb_tree prog)
- void cb_emit_close (cb_tree file, cb_tree opt)
- void cb_emit_commit (void)
- void cb_emit_continue (void)
- void cb_emit_delete (cb_tree file)
- void cb_emit_env_name (cb_tree value)
- void cb_emit_env_value (cb_tree value)
- void cb_emit_arg_number (cb_tree value)
- void cb_emit_command_line (cb_tree value)
- void cb_emit_display (cb_tree values, cb_tree upon, cb_tree no_adv, cb_tree pos, cb_tree fgc, cb_tree bgc, cb_tree scroll, int dispattrs)
- cb_tree cb_build_display_upon (cb_tree x)
- cb_tree cb_build_display_upon_direct (cb_tree x)
- void cb_emit_divide (cb_tree dividend, cb_tree divisor, cb_tree quotient, cb_tree remainder)
- void cb_emit_evaluate (cb_tree subject_list, cb_tree case_list)
- void cb_emit_free (cb_tree vars)
- void cb_emit_goto (cb_tree target, cb_tree depending)
- void cb_emit_exit (size_t goback)
- void cb_emit_if (cb_tree cond, cb_tree stmt1, cb_tree stmt2)
- void cb_emit_initialize (cb_tree vars, cb_tree fillinit, cb_tree value, cb_tree replacing, cb_tree def)
- void cb_emit_inspect (cb_tree var, cb_tree body, cb_tree replacing, int replconv)
- void cb_init_tarrying (void)
- cb_tree cb_build_tarrying_data (cb_tree x)
- cb_tree cb_build_tarrying_characters (cb_tree l)
- cb_tree cb_build_tarrying_all (void)
- cb_tree cb_build_tarrying_leading (void)
- cb_tree cb_build_tarrying_trailing (void)
- cb_tree cb_build_tarrying_value (cb_tree x, cb_tree l)
- cb_tree cb_build_replacing_characters (cb_tree x, cb_tree l)

- cb_tree cb_build_replacing_all (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_replacing_leading (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_replacing_first (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_replacing_trailing (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_converting (cb_tree x, cb_tree y, cb_tree l)
- cb_tree cb_build_inspect_region_start (void)
- cb_tree cb_build_inspect_region (cb_tree l, cb_tree pos, cb_tree x)
- int validate_move (cb_tree src, cb_tree dst, size_t is_value)
- cb_tree cb_build_move (cb_tree src, cb_tree dst)
- void cb_emit_move (cb_tree src, cb_tree dsts)
- void cb_emit_open (cb_tree file, cb_tree mode, cb_tree sharing)
- void cb_emit_perform (cb_tree perform, cb_tree body)
- cb_tree cb_build_perform_once (cb_tree body)
- cb_tree cb_build_perform_times (cb_tree times)
- cb_tree cb_build_perform_until (cb_tree condition, cb_tree varying)
- cb_tree cb_build_perform_forever (cb_tree body)
- cb_tree cb_build_perform_exit (struct cb_label *label)
- void cb_emit_read (cb_tree ref, cb_tree next, cb_tree into, cb_tree key, cb_tree lock_opts)
- void cb_emit_rewrite (cb_tree record, cb_tree from, cb_tree lockopt)
- void cb_emit_release (cb_tree record, cb_tree from)
- void cb_emit_return (cb_tree ref, cb_tree into)
- void cb_emit_rollback (void)
- void cb_emit_search (cb_tree table, cb_tree varying, cb_tree at_end, cb_tree whens)
- void cb_emit_search_all (cb_tree table, cb_tree at_end, cb_tree when, cb_tree stmts)
- void cb_emit_setenv (cb_tree x, cb_tree y)
- void cb_emit_set_to (cb_tree vars, cb_tree x)
- void cb_emit_set_up_down (cb_tree l, cb_tree flag, cb_tree x)
- void cb_emit_set_on_off (cb_tree l, cb_tree flag)
- void cb_emit_set_true (cb_tree l)
- void cb_emit_set_false (cb_tree l)
- void cb_emit_sort_init (cb_tree name, cb_tree keys, cb_tree col)
- void cb_emit_sort_using (cb_tree file, cb_tree l)
- void cb_emit_sort_input (cb_tree proc)
- void cb_emit_sort_giving (cb_tree file, cb_tree l)
- void cb_emit_sort_output (cb_tree proc)
- void cb_emit_sort_finish (cb_tree file)
- void cb_emit_start (cb_tree file, cb_tree op, cb_tree key)
- void cb_emit_stop_run (cb_tree x)
- void cb_emit_string (cb_tree items, cb_tree into, cb_tree pointer)
- void cb_emit_unlock (cb_tree ref)
- void cb_emit_unstring (cb_tree name, cb_tree delimited, cb_tree into, cb_tree pointer, cb_tree tallying)
- cb_tree cb_build_unstring_delimited (cb_tree all, cb_tree value)
- cb_tree cb_build_unstring_into (cb_tree name, cb_tree delimiter, cb_tree count)

- void cb_emit_write (cb_tree record, cb_tree from, cb_tree opt, cb_tree lockopt)
- cb_tree cb_build_write_advancing_lines (cb_tree pos, cb_tree lines)
- cb_tree cb_build_write_advancing_mnemonic (cb_tree pos, cb_tree mnemonic)
- cb_tree cb_build_write_advancing_page (cb_tree pos)

Variables

- size_t sending_id = 0
- size_t suppress_warn = 0

7.17.1 Define Documentation

7.17.1.1 #define cb_emit(*x*) current_statement->body = cb_list_add (current_statement->body, x)

Definition at line 67 of file typeck.c.

7.17.1.2 #define cb_emit_list(*l*) current_statement->body = cb_list_append (current_statement->body, l)

Definition at line 69 of file typeck.c.

7.17.1.3 #define COB_SYSTEM_GEN(*x, y, z*) { x, y },

7.17.1.4 #define dpush(*x*) decimal_stack = cb_cons (x, decimal_stack)

Definition at line 65 of file typeck.c.

7.17.1.5 #define START_STACK_SIZE 32

Definition at line 61 of file typeck.c.

7.17.1.6 #define TOKEN(*offset*) (expr_stack[expr_index + offset].token)

Definition at line 62 of file typeck.c.

7.17.1.7 #define VALUE(*offset*) (expr_stack[expr_index + offset].value)

Definition at line 63 of file typeck.c.

7.17.2 Function Documentation

7.17.2.1 cb_tree cb_build_add (cb_tree *v*, cb_tree *n*, cb_tree *round_opt*)

Definition at line 2591 of file typeck.c.

```
{
        cb_tree         opt;
        struct cb_field *f;

#ifdef  COB_NON_ALIGNED
        if (CB_INDEX_P (v)) {
                return cb_build_move (cb_build_binary_op (v, '+', n), v);
        }
        if (CB_TREE_CLASS (v) == CB_CLASS_POINTER) {
                current_program->gen_ptrmanip = 1;
                return cb_build_funcall_3 ("cob_pointer_manip", v, n, cb_int0);
        }
#else
        if (CB_INDEX_P (v) || CB_TREE_CLASS (v) == CB_CLASS_POINTER) {
                return cb_build_move (cb_build_binary_op (v, '+', n), v);
        }
#endif

        if (CB_REF_OR_FIELD_P (v)) {
                f = cb_field (v);
                f->count++;
        }
        if (CB_REF_OR_FIELD_P (n)) {
                f = cb_field (n);
                f->count++;
        }
        if (round_opt == cb_high) {
                if (cb_fits_int (n)) {
                        return cb_build_optim_add (v, n);
                } else {
                        return cb_build_funcall_3 ("cob_add", v, n, cb_int0);
                }
        }
        opt = build_store_option (v, round_opt);
        if (opt == cb_int0 && cb_fits_int (n)) {
                return cb_build_optim_add (v, n);
        }
        return cb_build_funcall_3 ("cob_add", v, n, opt);
}
```

7.17.2.2 cb_tree cb_build_address (cb_tree *x*)

Definition at line 1027 of file typeck.c.

```
{
        if (x == cb_error_node ||
            (CB_REFERENCE_P (x) && cb_ref (x) == cb_error_node)) {
                return cb_error_node;
        }

        return cb_build_cast_address (x);
}
```

7.17.2.3 cb_tree cb_build_assignment_name (struct cb_file * *cfile*, cb_tree *name*)

Definition at line 677 of file typeck.c.

```
{
        const char      *s;
        const char      *p;

        if (name == cb_error_node) {
                return cb_error_node;
        }

        switch (CB_TREE_TAG (name)) {
        case CB_TAG_LITERAL:
                if (strcmp ((char *)(CB_LITERAL(name)->data), "$#@DUMMY@#$") == 0
      ) {
                        cfile->special = 2;
                }
                return name;

        case CB_TAG_REFERENCE:
                s = CB_REFERENCE (name)->word->name;
                if (strcasecmp (s, "KEYBOARD") == 0) {
                        s = "#DUMMY#";
                        cfile->special = 1;
                        return cb_build_alphanumeric_literal ((ucharptr)s, strlen
       (s));
                }
                switch (cb_assign_clause) {
                case CB_ASSIGN_COBOL2002:
                        /* TODO */
                        return cb_error_node;

                case CB_ASSIGN_MF:
                        if (cfile->external_assign) {
                                p = strrchr (s, '-');
                                if (p) {
                                        s = p + 1;
                                }
                                return cb_build_alphanumeric_literal ((ucharptr)s
      , strlen (s));
                        }
                        current_program->reference_list =
                            cb_list_add (current_program->reference_list, name);
                        return name;

                case CB_ASSIGN_IBM:
                        /* check organization */
                        if (strncmp (s, "S-", 2) == 0 ||
                            strncmp (s, "AS-", 3) == 0) {
                                goto org;
                        }
                        /* skip the device label if exists */
                        if ((p = strchr (s, '-')) != NULL) {
                                s = p + 1;
                        }
                        /* check organization again */
                        if (strncmp (s, "S-", 2) == 0 ||
                            strncmp (s, "AS-", 3) == 0) {
org:
                                /* skip it for now */
                                s = strchr (s, '-') + 1;
```

```
                        }
                        /* convert the name into literal */
                        return cb_build_alphanumeric_literal ((ucharptr)s, strlen
       (s));
                }

        default:
                return cb_error_node;
        }
}
```

7.17.2.4 cb_tree cb_build_cond (cb_tree *x*)

Definition at line 2354 of file typeck.c.

```
{
        int                     size1;
        int                     size2;
        struct cb_field         *f;
        struct cb_binary_op     *p;
        cb_tree                 d1;
        cb_tree                 d2;

        switch (CB_TREE_TAG (x)) {
        case CB_TAG_CONST:
        case CB_TAG_FUNCALL:
                return x;
        case CB_TAG_REFERENCE:
                if (!CB_FIELD_P (cb_ref (x))) {
                        return cb_build_cond (cb_ref (x));
                }

                f = cb_field (x);

                /* level 88 condition */
                if (f->level == 88) {
                        /* We need to build a 88 condition at every occurrence
                           instead of once at the beginning because a 88 item
                           may be subscripted (i.e., it is not a constant tree).
      */
                        return cb_build_cond (build_cond_88 (x));
                }

                cb_error_x (x, _("Invalid expression"));
                return cb_error_node;
          case CB_TAG_BINARY_OP:
                p = CB_BINARY_OP (x);
                switch (p->op) {
                case '!':
                        return cb_build_negation (cb_build_cond (p->x));
                case '&':
                case '|':
                        return cb_build_binary_op (cb_build_cond (p->x), p->op,
      cb_build_cond (p->y));
                default:
                        if (CB_INDEX_P (p->x) || CB_INDEX_P (p->y)
                            || CB_TREE_CLASS (p->x) == CB_CLASS_POINTER
                            || CB_TREE_CLASS (p->y) == CB_CLASS_POINTER) {
                                x = cb_build_binary_op (p->x, '-', p->y);
```

```
                } else if (CB_BINARY_OP_P (p->x) || CB_BINARY_OP_P (p->y)
) {
                        /* decimal comparison */
                        d1 = decimal_alloc ();
                        d2 = decimal_alloc ();

                        decimal_expand (d1, p->x);
                        decimal_expand (d2, p->y);
                        dpush (cb_build_funcall_2 ("cob_decimal_cmp", d1,
 d2));
                        decimal_free ();
                        decimal_free ();
                        x = cb_list_reverse (decimal_stack);
                        decimal_stack = NULL;
                } else {
                        if (cb_chk_num_cond (p->x, p->y)) {
                                size1 = cb_field_size (p->x);
                                x = cb_build_funcall_3 ("memcmp",
                                        cb_build_cast_address (p->x),
                                        cb_build_cast_address (p->y),
                                        cb_int (size1));
                                break;
                        }
                        if (CB_TREE_CLASS (p->x) == CB_CLASS_NUMERIC
                            && CB_TREE_CLASS (p->y) == CB_CLASS_NUMERIC
                            && cb_fits_int (p->y)) {
                                x = cb_build_optim_cond (p);
                                break;
                        }

                        /* field comparison */
                        if ((CB_REF_OR_FIELD_P (p->x))
                           && (CB_TREE_CATEGORY (p->x) ==
CB_CATEGORY_ALPHANUMERIC ||
                               CB_TREE_CATEGORY (p->x) ==
CB_CATEGORY_ALPHABETIC)
                           && (cb_field_size (p->x) == 1)
                           && (!current_program->alphabet_name_list)
                           && (p->y == cb_space || p->y == cb_low ||
                               p->y == cb_high || p->y == cb_zero)) {
                                x = cb_build_funcall_2 ("$G", p->x, p->y)
;
                                break;
                        }
                        if (cb_chk_alpha_cond (p->x) && cb_chk_alpha_cond
 (p->y)) {
                                size1 = cb_field_size (p->x);
                                size2 = cb_field_size (p->y);
                        } else {
                                size1 = 0;
                                size2 = 0;
                        }
                        if (size1 == 1 && size2 == 1) {
                                x = cb_build_funcall_2 ("$G", p->x, p->y)
;
                        } else if (size1 != 0 && size1 == size2) {
                                x = cb_build_funcall_3 ("memcmp",
                                        cb_build_cast_address (p->x),
                                        cb_build_cast_address (p->y),
                                        cb_int (size1));
                        } else {
                                if (CB_TREE_CLASS (p->x) ==
```

```
      CB_CLASS_NUMERIC && p->y == cb_zero) {
                                              x = cb_build_optim_cond (p);
                                          } else {
                                              x = cb_build_funcall_2 ("cob_cmp"
      , p->x, p->y);
                                          }
                                  }
                          }
                  }
                  return cb_build_binary_op (x, p->op, p->y);
        default:
                  cb_error_x (x, _("Invalid expression"));
                  return cb_error_node;
        }
/* NOT REACHED */
        return x;
}
```

7.17.2.5 cb_tree cb_build_const_length (cb_tree *x*)

Definition at line 956 of file typeck.c.

```
{
        struct cb_field         *f;
        char                    buff[64];

        if (x == cb_error_node) {
                return cb_error_node;
        }
        if (CB_REFERENCE_P (x) && cb_ref (x) == cb_error_node) {
                return cb_error_node;
        }

        memset (buff, 0, sizeof (buff));
        f = CB_FIELD (cb_ref (x));
        if (f->flag_any_length) {
                cb_error (_("ANY LENGTH item not allowed here"));
                return cb_error_node;
        }
        if (f->level == 88) {
                cb_error (_("88 level item not allowed here"));
                return cb_error_node;
        }
        if (!f->flag_is_verified) {
                cb_validate_field (f);
        }
        sprintf (buff, "%d", f->memory_size);
        return cb_build_numeric_literal (0, (ucharptr)buff, 0);
}
```

7.17.2.6 cb_tree cb_build_converting (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3847 of file typeck.c.

```
{
```

```
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_converting", x, y
      ));
}
```

7.17.2.7 cb_tree cb_build_display_upon (cb_tree *x*)

Definition at line 3406 of file typeck.c.

```
{
        if (x == cb_error_node) {
                return cb_error_node;
        }

        switch (CB_SYSTEM_NAME (cb_ref (x))->token) {
        case CB_DEVICE_CONSOLE:
        case CB_DEVICE_SYSOUT:
                return cb_int0;
        case CB_DEVICE_SYSERR:
                return cb_int1;
        default:
                cb_error_x (x, _("Invalid output stream"));
                return cb_error_node;
        }
}
```

7.17.2.8 cb_tree cb_build_display_upon_direct (cb_tree *x*)

Definition at line 3425 of file typeck.c.

```
{
        const char      *name;
        cb_tree         sys;

        if (x == cb_error_node) {
                return cb_error_node;
        }
        name = CB_NAME (x);
        if (CB_REFERENCE (x)->word->count == 0) {
                sys = lookup_system_name (CB_NAME (x));
                if (sys != cb_error_node) {
                        switch (CB_SYSTEM_NAME (sys)->token) {
                        case CB_DEVICE_CONSOLE:
                        case CB_DEVICE_SYSOUT:
                                cb_warning_x (x, _("'%s' undefined in SPECIAL-NAM
      ES"), name);
                                return cb_int0;
                        case CB_DEVICE_SYSERR:
                                cb_warning_x (x, _("'%s' undefined in SPECIAL-NAM
      ES"), name);
                                return cb_int1;
                        default:
                                break;
                        }
                }
        }
```

```
        cb_error_x (x, _("'%s' undefined in SPECIAL-NAMES"), name);
        return cb_error_node;
}
```

7.17.2.9 cb_tree cb_build_expr (cb_tree *list*)

Definition at line 1820 of file typeck.c.

```
{
        cb_tree l;
/* RXW
        cb_tree x;
*/
        int     op;

        cb_expr_init ();

        for (l = list; l; l = CB_CHAIN (l)) {
                op = CB_PURPOSE_INT (l);
                switch (op) {
                case '9': /* NUMERIC */
                        cb_expr_shift_class ("cob_is_numeric");
                        break;
                case 'A': /* ALPHABETIC */
                        cb_expr_shift_class ("cob_is_alpha");
                        break;
                case 'L': /* ALPHABETIC_LOWER */
                        cb_expr_shift_class ("cob_is_lower");
                        break;
                case 'U': /* ALPHABETIC_UPPER */
                        cb_expr_shift_class ("cob_is_upper");
                        break;
                case 'P': /* POSITIVE */
                        cb_expr_shift_sign ('>');
                        break;
                case 'N': /* NEGATIVE */
                        cb_expr_shift_sign ('<');
                        break;
                case 'O': /* OMITTED */
                        current_statement->null_check = NULL;
                        cb_expr_shift_class ("cob_is_omitted");
                        break;
/* RXW
                case 'x':
                        if (CB_VALUE (l) && CB_REFERENCE_P (CB_VALUE (l))) {
                                x = CB_CHAIN (l);
                                if (x && cb_field (CB_VALUE (l))->level == 88) {
                                        switch (CB_PURPOSE_INT (x)) {
                                        case '&':
                                        case '|':
                                        case '(':
                                        case ')':
                                                break;
                                        default:
                                                cb_error (_("Invalid condition"))
      ;
                                                break;
                                        }
```

```
                                }
                        }
                        cb_expr_shift (op, CB_VALUE (l));
                        break;
*/
                default:
                        cb_expr_shift (op, CB_VALUE (l));
                        break;
                }
        }

        return cb_expr_finish ();
}
```

7.17.2.10 cb_tree cb_build_identifier (cb_tree *x*)

Definition at line 763 of file typeck.c.

```
{
        struct cb_reference     *r;
        struct cb_field         *f;
        struct cb_field         *p;
        const char              *name;
        cb_tree                 v;
        cb_tree                 e1;
        cb_tree                 e2;
        cb_tree                 l;
        cb_tree                 sub;
        int                     offset;
        int                     length;
        int                     n;

        if (x == cb_error_node) {
                return cb_error_node;
        }

        r = CB_REFERENCE (x);
        name = r->word->name;

        /* resolve reference */
        v = cb_ref (x);
        if (v == cb_error_node) {
                return cb_error_node;
        }

        /* check if it is a data name */
        if (!CB_FIELD_P (v)) {
                if (r->subs) {
                        cb_error_x (x, _("'%s' cannot be subscripted"), name);
                        return cb_error_node;
                }
                if (r->offset) {
                        cb_error_x (x, _("'%s' cannot be reference modified"), na
      me);
                        return cb_error_node;
                }
                return x;
        }
        f = CB_FIELD (v);
```

```
  /* BASED check */
  if (CB_EXCEPTION_ENABLE (COB_EC_BOUND_PTR)) {
          for (p = f; p->parent; p = p->parent) {
                  ;
          }
          if (current_statement) {
                  if (p->flag_item_based ||
                     (f->storage == CB_STORAGE_LINKAGE &&
                      !p->flag_is_pdiv_parm)) {
                          current_statement->null_check =
cb_build_funcall_2 (
                                  "cob_check_based",
                                  cb_build_address (
cb_build_field_reference (p, NULL)),
                                  cb_build_string0 ((ucharptr)name));
                  }
          }
  }

  /* check the number of subscripts */
  if (!r->all && cb_list_length (r->subs) != f->indexes) {
          switch (f->indexes) {
          case 0:
                  cb_error_x (x, _("'%s' cannot be subscripted"), name);
                  return cb_error_node;
          case 1:
                  cb_error_x (x, _("'%s' requires 1 subscript"), name);
                  return cb_error_node;
          default:
                  cb_error_x (x, _("'%s' requires %d subscripts"), name, f-
>indexes);
                  return cb_error_node;
          }
  }

  /* subscript check */
  if (!r->all && r->subs) {
          l = r->subs;
          for (p = f; p; p = p->parent) {
                  if (p->flag_occurs) {
                          sub = cb_check_integer_value (CB_VALUE (l));

                          l = CB_CHAIN (l);

                          if (sub == cb_error_node) {
                                  continue;
                          }

                          /* compile-time check */
                          if (CB_LITERAL_P (sub)) {
                                  n = cb_get_int (sub);
                                  if (n < 1 || n > p->occurs_max) {
                                          cb_error_x (x, _("Subscript of '%
s' out of bounds: %d"),
                                                      name, n);
                                  }
                          }

                          /* run-time check */
                          if (CB_EXCEPTION_ENABLE (COB_EC_BOUND_SUBSCRIPT))
 {
```

```
                                        if (p->occurs_depending) {
                                                e1 = cb_build_funcall_4 ("cob_che
ck_odo",
                                                         cb_build_cast_integer (p
->occurs_depending),
                                                         cb_int (p->occurs_min),
                                                         cb_int (p->occurs_max),
                                                         cb_build_string0
                                                         ((ucharptr)(cb_field (p-
>occurs_depending)->name)));
                                                e2 = cb_build_funcall_4 ("cob_che
ck_subscript",
                                                         cb_build_cast_integer (s
ub),
                                                         cb_int1,
                                                         cb_build_cast_integer (p
->occurs_depending),
                                                         cb_build_string0 ((
ucharptr)name));
                                                r->check = cb_list_add (r->check,
 e1);
                                                r->check = cb_list_add (r->check,
 e2);
                                        } else {
                                                if (!CB_LITERAL_P (sub)) {
                                                        e1 = cb_build_funcall_4 (
"cob_check_subscript",
                                                                                 
cb_build_cast_integer (sub),
                                                                                 
                                                                 cb_int1,
                                                                 cb_int (p->
occurs_max),
                                                                 cb_build_string0
((ucharptr)name));
                                                        r->check = cb_list_add (r
->check, e1);
                                                }
                                        }
                                }
                        }
                }
          }
  }

  /* reference modification check */
  if (r->offset) {
          /* compile-time check */
          if (CB_LITERAL_P (r->offset)) {
                  offset = cb_get_int (r->offset);
                  if (offset < 1 || offset > f->size) {
                          cb_error_x (x, _("Offset of '%s' out of bounds: %
d"), name, offset);
                  } else if (r->length && CB_LITERAL_P (r->length)) {
                          length = cb_get_int (r->length);
                          if (length < 1 || length > f->size - offset + 1)
{
                                  cb_error_x (x, _("Length of '%s' out of b
ounds: %d"),
                                              name, length);
                          }
                  }
          }
```

```
                /* run-time check */
                if (CB_EXCEPTION_ENABLE (COB_EC_BOUND_REF_MOD)) {
                        if (!CB_LITERAL_P (r->offset)
                            || (r->length && !CB_LITERAL_P (r->length))) {
                                e1 = cb_build_funcall_4 ("cob_check_ref_mod",
                                                         cb_build_cast_integer (r
      ->offset),
                                                         r->length ?
      cb_build_cast_integer (r->length) :
                                                         cb_int1, cb_int (f->
      size),
                                                         cb_build_string0 ((
      ucharptr)f->name));
                                r->check = cb_list_add (r->check, e1);
                        }
                }
          }

          if (f->storage == CB_STORAGE_CONSTANT) {
                  return CB_VALUE (f->values);
          }

          return x;
}
```

7.17.2.11 cb_tree cb_build_index (cb_tree *x*, cb_tree *values*, int *indexed_by*, struct cb_field * *qual*)

Definition at line 744 of file typeck.c.

```
{
          struct cb_field *f;

          f = CB_FIELD (cb_build_field (x));
          f->usage = CB_USAGE_INDEX;
          cb_validate_field (f);
          if (values) {
                  f->values = cb_list_init (values);
          }
          if (qual) {
                  f->index_qual = qual;
          }
          f->flag_indexed_by = indexed_by;
          current_program->working_storage = cb_field_add (current_program->
      working_storage, f);
          return x;
}
```

7.17.2.12 cb_tree cb_build_inspect_region (cb_tree *l*, cb_tree *pos*, cb_tree *x*)

Definition at line 3859 of file typeck.c.

```
{
          if (pos == CB_BEFORE) {
                  return cb_list_add (l, cb_build_funcall_1 ("cob_inspect_before",
```

```
      x));
        } else {
                return cb_list_add (l, cb_build_funcall_1 ("cob_inspect_after", x
      ));
        }
}
```

7.17.2.13 cb_tree cb_build_inspect_region_start (void)

Definition at line 3853 of file typeck.c.

```
{
        return cb_list_init (cb_build_funcall_0 ("cob_inspect_start"));
}
```

7.17.2.14 cb_tree cb_build_length (cb_tree *x*)

Definition at line 986 of file typeck.c.

```
{
        struct cb_field         *f;
        struct cb_literal       *l;
        cb_tree                 temp;
        char                    buff[64];

        if (x == cb_error_node) {
                return cb_error_node;
        }
        if (CB_REFERENCE_P (x) && cb_ref (x) == cb_error_node) {
                return cb_error_node;
        }

        memset (buff, 0, sizeof (buff));
        if (CB_LITERAL_P (x)) {
                l = CB_LITERAL (x);
                sprintf (buff, "%d", (int)l->size);
                return cb_build_numeric_literal (0, (ucharptr)buff, 0);
        }
        if (CB_REF_OR_FIELD_P (x)) {
                f = CB_FIELD (cb_ref (x));
                if (f->flag_any_length) {
                        return cb_build_any_intrinsic (cb_list_init (x));
                }
                if (cb_field_variable_size (f) == NULL) {
                        sprintf (buff, "%d", cb_field_size (x));
                        return cb_build_numeric_literal (0, (ucharptr)buff, 0);
                }
        }
        if (CB_INTRINSIC_P (x)) {
                return cb_build_any_intrinsic (cb_list_init (x));
        }
        temp = cb_build_index (cb_build_filler (), NULL, 0, NULL);
        CB_FIELD (cb_ref (temp))->usage = CB_USAGE_LENGTH;
        CB_FIELD (cb_ref (temp))->count++;
        cb_emit (cb_build_assign (temp, cb_build_length_1 (x)));
        return temp;
}
```

7.17.2.15 cb_tree cb_build_move (cb_tree *src*, cb_tree *dst*)

Definition at line 4964 of file typeck.c.

```
{
        struct cb_field *f;
        struct cb_field *p;

        if (src == cb_error_node || dst == cb_error_node) {
                return cb_error_node;
        }

        if (validate_move (src, dst, 0) < 0) {
                return cb_error_node;
        }

        if (CB_REFERENCE_P (src)) {
                CB_REFERENCE (src)->type = CB_SENDING_OPERAND;
        }
        if (CB_REFERENCE_P (dst)) {
                CB_REFERENCE (dst)->type = CB_RECEIVING_OPERAND;
        }

        if (CB_TREE_CLASS (dst) == CB_CLASS_POINTER) {
                return cb_build_assign (dst, src);
        }

        if (CB_REFERENCE_P (src) && CB_ALPHABET_NAME_P(CB_REFERENCE(src)->value))
       {
                return cb_build_move_call (src, dst);
        }
        if (CB_INDEX_P (dst)) {
                if (src == cb_null) {
                        return cb_build_assign (dst, cb_zero);
                }
                return cb_build_assign (dst, src);
        }

        if (CB_INDEX_P (src)) {
                return cb_build_funcall_2 ("cob_set_int", dst,
      cb_build_cast_integer (src));
        }

        if (CB_INTRINSIC_P (src) || CB_INTRINSIC_P (dst)) {
                return cb_build_move_call (src, dst);
        }

        f = cb_field (dst);

        if (CB_EXCEPTION_ENABLE (COB_EC_BOUND_SUBSCRIPT)) {
                for (p = f; p; p = p->parent) {
                        if (p->flag_occurs) {
                                return cb_build_move_call (src, dst);
                        }
                }
                if (CB_REF_OR_FIELD_P (src)) {
                        for (p = cb_field (src); p; p = p->parent) {
                                if (p->flag_occurs) {
                                        return cb_build_move_call (src, dst);
                                }
                        }
                }
```

```
        }

        /* output optimal code */
        if (src == cb_zero) {
                return cb_build_move_zero (dst);
        } else if (src == cb_space) {
                return cb_build_move_space (dst);
        } else if (src == cb_high) {
                return cb_build_move_high (dst);
        } else if (src == cb_low) {
                return cb_build_move_low (dst);
        } else if (src == cb_quote) {
                return cb_build_move_quote (dst);
        } else if (CB_LITERAL_P (src)) {
                return cb_build_move_literal (src, dst);
        }
        return cb_build_move_field (src, dst);
}
```

7.17.2.16 cb_tree cb_build_perform_exit (struct cb_label * *label*)

Definition at line 5156 of file typeck.c.

```
{
        cb_tree x;

        x = cb_build_perform (CB_PERFORM_EXIT);
        CB_PERFORM (x)->data = CB_TREE (label);
        return x;
}
```

7.17.2.17 cb_tree cb_build_perform_forever (cb_tree *body*)

Definition at line 5143 of file typeck.c.

```
{
        cb_tree x;

        if (body == cb_error_node) {
                return cb_error_node;
        }
        x = cb_build_perform (CB_PERFORM_FOREVER);
        CB_PERFORM (x)->body = body;
        return x;
}
```

7.17.2.18 cb_tree cb_build_perform_once (cb_tree *body*)

Definition at line 5105 of file typeck.c.

```
{
        cb_tree x;
```

```
        if (body == cb_error_node) {
                return cb_error_node;
        }
        x = cb_build_perform (CB_PERFORM_ONCE);
        CB_PERFORM (x)->body = body;
        return x;
}
```

7.17.2.19 cb_tree cb_build_perform_times (cb_tree *times*)

Definition at line 5118 of file typeck.c.

```
{
        cb_tree x;

        if (cb_check_integer_value (times) == cb_error_node) {
                return cb_error_node;
        }

        x = cb_build_perform (CB_PERFORM_TIMES);
        CB_PERFORM (x)->data = times;
        return x;
}
```

7.17.2.20 cb_tree cb_build_perform_until (cb_tree *condition*, cb_tree *varying*)

Definition at line 5132 of file typeck.c.

```
{
        cb_tree x;

        x = cb_build_perform (CB_PERFORM_UNTIL);
        CB_PERFORM (x)->test = condition;
        CB_PERFORM (x)->varying = varying;
        return x;
}
```

7.17.2.21 cb_tree cb_build_ppointer (cb_tree *x*)

Definition at line 1038 of file typeck.c.

```
{
        struct cb_field *f;

        if (x == cb_error_node ||
            (CB_REFERENCE_P (x) && cb_ref (x) == cb_error_node)) {
                return cb_error_node;
        }

        if (CB_REFERENCE_P (x)) {
                f = cb_field (cb_ref(x));
```

```
                f->count++;
        }
        return cb_build_cast_ppointer (x);
}
```

7.17.2.22 const char* cb_build_program_id (cb_tree *name*, cb_tree *alt_name*)

Definition at line 591 of file typeck.c.

```
{
        const char      *s;

/* This needs some more thought, should we generate an entry
        point per program source name ?
        if (alt_name) {
                s = (char *)CB_LITERAL (alt_name)->data;
        } else if (CB_LITERAL_P (name)) {
                s = (char *)CB_LITERAL (name)->data;
        } else {
                s = (char *)CB_NAME (name);
        }

        if (!cb_flag_main && strcmp (s, source_name)) {
                cb_warning (_("Source name '%s' differs from PROGRAM-ID '%s'"),
                                source_name, s);
                current_program->source_name = strdup (source_name);
        }
 End comment out */

        if (alt_name) {
                current_program->orig_source_name = strdup ((char *)CB_LITERAL (a
      lt_name)->data);
                s = (char *)CB_LITERAL (alt_name)->data;
        } else if (CB_LITERAL_P (name)) {
                current_program->orig_source_name = strdup ((char *)CB_LITERAL (n
      ame)->data);
                s = cb_encode_program_id ((char *)CB_LITERAL (name)->data);
        } else {
                current_program->orig_source_name = strdup (CB_NAME (name));
                s = cb_encode_program_id (CB_NAME (name));
        }
        if (cobc_check_valid_name (current_program->orig_source_name)) {
                cb_error (_("PROGRAM-ID '%s' invalid"), current_program->
      orig_source_name);
        }
        return s;
}
```

7.17.2.23 void cb_build_registers (void)

Definition at line 494 of file typeck.c.

```
{
#if !defined(__linux__) && !defined(__CYGWIN__) && defined(HAVE_TIMEZONE)
        long    contz;
```

```
#endif
        time_t  t;
        char    buff[48];

        /* RETURN-CODE */
        if (!current_program->nested_level) {
                current_program->cb_return_code =
                        cb_build_index (cb_build_reference ("RETURN-CODE"),
                                        cb_zero, 0, NULL);
                cb_field (current_program->cb_return_code)->flag_is_global = 1;
        }

        /* SORT-RETURN */
        current_program->cb_sort_return =
                cb_build_index (cb_build_reference ("SORT-RETURN"), cb_zero, 0,
      NULL);
        cb_field (current_program->cb_sort_return)->flag_no_init = 1;

        /* NUMBER-OF-CALL-PARAMETERS */
        current_program->cb_call_params =
                cb_build_index (cb_build_reference ("NUMBER-OF-CALL-PARAMETERS"),
       cb_zero, 0, NULL);
        cb_field (current_program->cb_call_params)->flag_no_init = 1;

        /* TALLY */
        /* 01 TALLY GLOBAL PICTURE 9(9) USAGE COMP-5 VALUE ZERO. */
        /* TALLY/EXAMINE  not standard/supported */

        t = time (NULL);

        /* WHEN-COMPILED */
        memset (buff, 0, sizeof (buff));
        strftime (buff, 17, "%m/%d/%y%H.%M.%S", localtime (&t));
        cb_build_constant (cb_build_reference ("WHEN-COMPILED"),
                           cb_build_alphanumeric_literal ((ucharptr)buff, 16));

        /* FUNCTION WHEN-COMPILED */
        memset (buff, 0, sizeof (buff));
#if defined(__linux__) || defined(__CYGWIN__)
        strftime (buff, 22, "%Y%m%d%H%M%S00%z", localtime (&t));
#elif defined(HAVE_TIMEZONE)
        strftime (buff, 17, "%Y%m%d%H%M%S00", localtime (&t));
        if (timezone <= 0) {
                contz = -timezone;
                buff[16] = '+';
        } else {
                contz = timezone;
                buff[16] = '-';
        }
        sprintf (&buff[17], "%2.2ld%2.2ld", contz / 3600, contz % 60);
#else
        strftime (buff, 22, "%Y%m%d%H%M%S0000000", localtime (&t));
#endif
        cb_intr_whencomp = cb_build_alphanumeric_literal ((ucharptr)buff, 21);

        /* FUNCTION PI */
        memset (buff, 0, sizeof (buff));
        strcpy (buff, "31415926535897932384626433832795029");
        cb_intr_pi = cb_build_numeric_literal (0, (ucharptr)buff, 34);

        /* FUNCTION E */
        memset (buff, 0, sizeof (buff));
```

```
        strcpy (buff, "2718281828459045235360287471352662 5");
        cb_intr_e = cb_build_numeric_literal (0, (ucharptr)buff, 34);
}
```

7.17.2.24 cb_tree cb_build_replacing_all (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3823 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_all", y, x));
}
```

7.17.2.25 cb_tree cb_build_replacing_characters (cb_tree *x*, cb_tree *l*)

Definition at line 3817 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_1 ("cob_inspect_characters", x));
}
```

7.17.2.26 cb_tree cb_build_replacing_first (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3835 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_first", y, x));
}
```

7.17.2.27 cb_tree cb_build_replacing_leading (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3829 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_leading", y, x));
}
```

7.17.2.28 cb_tree cb_build_replacing_trailing (cb_tree *x*, cb_tree *y*, cb_tree *l*)

Definition at line 3841 of file typeck.c.

```
{
        return cb_list_add (l, cb_build_funcall_2 ("cob_inspect_trailing", y, x))
      ;
}
```

7.17.2.29 cb_tree cb_build_section_name (cb_tree *name*, int *sect_or_para*)

Definition at line 653 of file typeck.c.

```
{
        cb_tree x;

        if (name == cb_error_node) {
                return cb_error_node;
        }

        if (CB_REFERENCE (name)->word->count > 0) {
                x = CB_VALUE (CB_REFERENCE (name)->word->items);
                /* Used as a non-label name or used as a section name.
                   Duplicate paragraphs are allowed if not referenced;
                   Checked in typeck.c */
                if (!CB_LABEL_P (x) || sect_or_para == 0
                    || (sect_or_para && CB_LABEL_P (x) && CB_LABEL (x)->is_sectio
      n)) {
                        redefinition_error (name);
                        return cb_error_node;
                }
        }

        return name;
}
```

7.17.2.30 cb_tree cb_build_sub (cb_tree *v*, cb_tree *n*, cb_tree *round_opt*)

Definition at line 2633 of file typeck.c.

```
{
        cb_tree         opt;
        struct cb_field *f;

#ifdef  COB_NON_ALIGNED
        if (CB_INDEX_P (v)) {
                return cb_build_move (cb_build_binary_op (v, '-', n), v);
        }
        if (CB_TREE_CLASS (v) == CB_CLASS_POINTER) {
                current_program->gen_ptrmanip = 1;
                return cb_build_funcall_3 ("cob_pointer_manip", v, n, cb_int1);
        }
#else
        if (CB_INDEX_P (v) || CB_TREE_CLASS (v) == CB_CLASS_POINTER) {
                return cb_build_move (cb_build_binary_op (v, '-', n), v);
        }
#endif

        if (CB_REF_OR_FIELD_P (v)) {
                f = cb_field (v);
                f->count++;
        }
        if (CB_REF_OR_FIELD_P (n)) {
                f = cb_field (n);
                f->count++;
        }
        opt = build_store_option (v, round_opt);
```

```
        if (opt == cb_int0 && cb_fits_int (n)) {
                return cb_build_optim_sub (v, n);
        }
        return cb_build_funcall_3 ("cob_sub", v, n, opt);
}
```

7.17.2.31 cb_tree cb_build_tarrying_all (void)

Definition at line 3778 of file typeck.c.

```
{
        if (inspect_data == NULL) {
                cb_error (_("Data name expected before ALL"));
        }
        inspect_func = "cob_inspect_all";
        return NULL;
}
```

7.17.2.32 cb_tree cb_build_tarrying_characters (cb_tree *l*)

Definition at line 3768 of file typeck.c.

```
{
        if (inspect_data == NULL) {
                cb_error (_("Data name expected before CHARACTERS"));
        }
        inspect_func = NULL;
        return cb_list_add (l, cb_build_funcall_1 ("cob_inspect_characters", insp
      ect_data));
}
```

7.17.2.33 cb_tree cb_build_tarrying_data (cb_tree *x*)

Definition at line 3761 of file typeck.c.

```
{
        inspect_data = x;
        return NULL;
}
```

7.17.2.34 cb_tree cb_build_tarrying_leading (void)

Definition at line 3788 of file typeck.c.

```
{
        if (inspect_data == NULL) {
                cb_error (_("Data name expected before LEADING"));
        }
        inspect_func = "cob_inspect_leading";
        return NULL;
}
```

7.17.2.35 cb_tree cb_build_tarrying_trailing (void)

Definition at line 3798 of file typeck.c.

```
{
        if (inspect_data == NULL) {
                cb_error (_("Data name expected before TRAILING"));
        }
        inspect_func = "cob_inspect_trailing";
        return NULL;
}
```

7.17.2.36 cb_tree cb_build_tarrying_value (cb_tree *x*, cb_tree *l*)

Definition at line 3808 of file typeck.c.

```
{
        if (inspect_func == NULL) {
                cb_error_x (x, _("ALL, LEADING or TRAILING expected before '%s'")
      , cb_name (x));
        }
        return cb_list_add (l, cb_build_funcall_2 (inspect_func, inspect_data, x)
      );
}
```

7.17.2.37 cb_tree cb_build_unstring_delimited (cb_tree *all*, cb_tree *value*)

Definition at line 5866 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return cb_error_node;
        }
        return cb_build_funcall_2 ("cob_unstring_delimited", value, all);
}
```

7.17.2.38 cb_tree cb_build_unstring_into (cb_tree *name*, cb_tree *delimiter*, cb_tree *count*)

Definition at line 5875 of file typeck.c.

```
{
        if (cb_validate_one (name)) {
                return cb_error_node;
        }
        if (delimiter == NULL) {
                delimiter = cb_int0;
        }
        if (count == NULL) {
                count = cb_int0;
        }
        return cb_build_funcall_3 ("cob_unstring_into", name, delimiter, count);
}
```

7.17.2.39 cb_tree cb_build_write_advancing_lines (cb_tree *pos*, cb_tree *lines*)

Definition at line 5957 of file typeck.c.

```
{
        cb_tree e;
        int     opt;

        opt = (pos == CB_BEFORE) ? COB_WRITE_BEFORE : COB_WRITE_AFTER;
        e = cb_build_binary_op (cb_int (opt | COB_WRITE_LINES), '+', lines);
        return cb_build_cast_integer (e);
}
```

7.17.2.40 cb_tree cb_build_write_advancing_mnemonic (cb_tree *pos*, cb_tree *mnemonic*)

Definition at line 5968 of file typeck.c.

```
{
        int     opt;
        int     token;

        token = CB_SYSTEM_NAME (cb_ref (mnemonic))->token;
        switch (token) {
        case CB_FEATURE_FORMFEED:
                opt = (pos == CB_BEFORE) ? COB_WRITE_BEFORE : COB_WRITE_AFTER;
                return cb_int (opt | COB_WRITE_PAGE);
        case CB_FEATURE_C01:
        case CB_FEATURE_C02:
        case CB_FEATURE_C03:
        case CB_FEATURE_C04:
        case CB_FEATURE_C05:
        case CB_FEATURE_C06:
        case CB_FEATURE_C07:
        case CB_FEATURE_C08:
        case CB_FEATURE_C09:
        case CB_FEATURE_C10:
        case CB_FEATURE_C11:
        case CB_FEATURE_C12:
                opt = (pos == CB_BEFORE) ? COB_WRITE_BEFORE : COB_WRITE_AFTER;
                return cb_int (opt | COB_WRITE_CHANNEL | COB_WRITE_PAGE | token);

        default:
                cb_error_x (mnemonic, _("Invalid mnemonic name"));
                return cb_error_node;
        }
}
```

7.17.2.41 cb_tree cb_build_write_advancing_page (cb_tree *pos*)

Definition at line 5999 of file typeck.c.

```
{
        int opt = (pos == CB_BEFORE) ? COB_WRITE_BEFORE : COB_WRITE_AFTER;

        return cb_int (opt | COB_WRITE_PAGE);
}
```

7.17.2.42 cb_tree cb_check_numeric_value (cb_tree *x*)

Definition at line 426 of file typeck.c.

```
{
        if (x == cb_error_node) {
                return cb_error_node;
        }

        if (CB_TREE_CATEGORY (x) == CB_CATEGORY_NUMERIC) {
                return x;
        }

        cb_error_x (x, _("'%s' is not a numeric value"), cb_name (x));
        return cb_error_node;
}
```

7.17.2.43 void cb_define_switch_name (cb_tree *name*, cb_tree *sname*, cb_tree *flag*, cb_tree *ref*)

Definition at line 629 of file typeck.c.

```
{
        cb_tree switch_id;
        cb_tree value;

        if (name == cb_error_node) {
                return;
        }
        if (sname == cb_error_node) {
                return;
        }
        if (CB_SYSTEM_NAME (sname)->category != CB_SWITCH_NAME) {
                cb_error_x (ref, _("Switch-name is expected '%s'"), CB_NAME (ref)
      );
        } else {
                switch_id = cb_int (CB_SYSTEM_NAME (sname)->token);
                value = cb_build_funcall_1 ("cob_get_switch", switch_id);
                if (flag == cb_int0) {
                        value = cb_build_negation (value);
                }
                cb_build_constant (name, value);
        }
}
```

7.17.2.44 void cb_emit_accept (cb_tree *var*, cb_tree *pos*, cb_tree *fgc*, cb_tree *bgc*, cb_tree *scroll*, int *dispattrs*)

Definition at line 2802 of file typeck.c.

```
{
        cb_tree line;
        cb_tree column;
```

```
  if (cb_validate_one (var)) {
          return;
  }
  if (cb_validate_one (pos)) {
          return;
  }
  if (cb_validate_one (fgc)) {
          return;
  }
  if (cb_validate_one (bgc)) {
          return;
  }
  if (cb_validate_one (scroll)) {
          return;
  }
  if (current_program->flag_screen) {
          /* Bump ref count to force CRT STATUS field generation */
          cb_field (current_program->crt_status)->count++;
          if ((CB_REF_OR_FIELD_P (var)) &&
               CB_FIELD (cb_ref (var))->storage == CB_STORAGE_SCREEN) {
                  output_screen_from (CB_FIELD (cb_ref (var)), 0);
                  gen_screen_ptr = 1;
                  if (pos) {
                          if (CB_PAIR_P (pos)) {
                                  line = CB_PAIR_X (pos);
                                  column = CB_PAIR_Y (pos);
                                  cb_emit (cb_build_funcall_3 ("cob_screen_
accept",
                                          var, line, column));
                          } else {
                                  cb_emit (cb_build_funcall_3 ("cob_screen_
accept",
                                          var, pos, NULL));
                          }
                  } else {
                          cb_emit (cb_build_funcall_3 ("cob_screen_accept",

                                  var, NULL, NULL));
                  }
                  gen_screen_ptr = 0;
                  output_screen_to (CB_FIELD (cb_ref (var)), 0);
          } else {
                  if (pos || fgc || bgc) {
                          if (!pos) {
                                  cb_emit (cb_build_funcall_7 ("cob_field_a
ccept",
                                          var, NULL, NULL, fgc, bgc,
                                          scroll, cb_int (dispattrs)));
                          } else if (CB_PAIR_P (pos)) {
                                  line = CB_PAIR_X (pos);
                                  column = CB_PAIR_Y (pos);
                                  cb_emit (cb_build_funcall_7 ("cob_field_a
ccept",
                                          var, line, column, fgc, bgc,
                                          scroll, cb_int (dispattrs)));
                          } else {
                                  cb_emit (cb_build_funcall_7 ("cob_field_a
ccept",
                                          var, pos, NULL, fgc, bgc,
                                          scroll, cb_int (dispattrs)));
                          }
                  } else {
```

```
                                cb_emit (cb_build_funcall_7 ("cob_field_accept",
                                        var, NULL, NULL, fgc, bgc,
                                        scroll, cb_int (dispattrs)));
                        }
                }
        } else if (pos || fgc || bgc || scroll) {
                /* Bump ref count to force CRT STATUS field generation */
                cb_field (current_program->crt_status)->count++;
                if (!pos) {
                        cb_emit (cb_build_funcall_7 ("cob_field_accept",
                                var, NULL, NULL, fgc, bgc, scroll,
                                cb_int (dispattrs)));
                } else if (CB_PAIR_P (pos)) {
                        line = CB_PAIR_X (pos);
                        column = CB_PAIR_Y (pos);
                        cb_emit (cb_build_funcall_7 ("cob_field_accept",
                                var, line, column, fgc, bgc, scroll,
                                cb_int (dispattrs)));
                } else {
                        cb_emit (cb_build_funcall_7 ("cob_field_accept",
                                var, pos, NULL, fgc, bgc, scroll,
                                cb_int (dispattrs)));
                }
        } else {
                cb_emit (cb_build_funcall_1 ("cob_accept", var));
        }
}
```

7.17.2.45 void cb_emit_accept_arg_number (cb_tree *var*)

Definition at line 2986 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_arg_number", var));
}
```

7.17.2.46 void cb_emit_accept_arg_value (cb_tree *var*)

Definition at line 2995 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_arg_value", var));
}
```

7.17.2.47 void cb_emit_accept_command_line (cb_tree *var*)

Definition at line 2956 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_command_line", var));
}
```

7.17.2.48 void cb_emit_accept_date (cb_tree *var*)

Definition at line 2902 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_date", var));
}
```

7.17.2.49 void cb_emit_accept_date_yyyymmdd (cb_tree *var*)

Definition at line 2911 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_date_yyyymmdd", var));
}
```

7.17.2.50 void cb_emit_accept_day (cb_tree *var*)

Definition at line 2920 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_day", var));
}
```

7.17.2.51 void cb_emit_accept_day_of_week (cb_tree *var*)

Definition at line 2938 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_day_of_week", var));
}
```

7.17.2.52 void cb_emit_accept_day_yyyyddd (cb_tree *var*)

Definition at line 2929 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_day_yyyyddd", var));
}
```

7.17.2.53 void cb_emit_accept_environment (cb_tree *var*)

Definition at line 2977 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_environment", var));
}
```

7.17.2.54 void cb_emit_accept_line_or_col (cb_tree *var,* const int *l_or_c*)

Definition at line 2893 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_2 ("cob_screen_line_col", var, cb_int (l_or_c))
      );
}
```

7.17.2.55 void cb_emit_accept_mnemonic (cb_tree *var,* cb_tree *mnemonic*)

Definition at line 3004 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        switch (CB_SYSTEM_NAME (cb_ref (mnemonic))->token) {
        case CB_DEVICE_CONSOLE:
        case CB_DEVICE_SYSIN:
                cb_emit (cb_build_funcall_1 ("cob_accept", var));
                break;
        default:
                cb_error_x (mnemonic, _("Invalid input stream '%s'"),
                            cb_name (mnemonic));
                break;
        }
}
```

7.17.2.56 void cb_emit_accept_name (cb_tree *var*, cb_tree *name*)

Definition at line 3022 of file typeck.c.

```
{
        cb_tree sys;

        if (cb_validate_one (var)) {
                return;
        }
        if (CB_REFERENCE (name)->word->count == 0) {
                sys = lookup_system_name (CB_NAME (name));

                if (sys != cb_error_node) {
                        switch (CB_SYSTEM_NAME (sys)->token) {
                        case CB_DEVICE_CONSOLE:
                        case CB_DEVICE_SYSIN:
                                cb_warning_x (name, _("'%s' undefined in SPECIAL-
      NAMES"), CB_NAME (name));
                                cb_emit (cb_build_funcall_1 ("cob_accept", var));

                                return;
                        default:
                                break;
                        }
                }
        }

        cb_error_x (name, _("'%s' undefined in SPECIAL-NAMES"), CB_NAME (name));
}
```

7.17.2.57 void cb_emit_accept_time (cb_tree *var*)

Definition at line 2947 of file typeck.c.

```
{
        if (cb_validate_one (var)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_accept_time", var));
}
```

7.17.2.58 void cb_emit_allocate (cb_tree *target1*, cb_tree *target2*, cb_tree *size*, cb_tree *initialize*)

Definition at line 3053 of file typeck.c.

```
{
        cb_tree x;
        char    buff[32];

        if (cb_validate_one (target1)) {
                return;
        }
```

```
        if (cb_validate_one (target2)) {
                return;
        }
        if (cb_validate_one (size)) {
                return;
        }
        if (target1) {
                if (!(CB_REFERENCE_P(target1) &&
                      cb_field (target1)->flag_item_based)) {
                        cb_error_x (CB_TREE(current_statement),
                                _("Target of ALLOCATE is not a BASED item"));
                }
        }
        if (target2) {
                if (!(CB_REFERENCE_P(target2) &&
                      CB_TREE_CLASS (target2) == CB_CLASS_POINTER)) {
                        cb_error_x (CB_TREE(current_statement),
                                _("Target of RETURNING is not a data pointer"));
                }
        }
        if (size) {
                if (CB_TREE_CLASS (size) != CB_CLASS_NUMERIC) {
                        cb_error_x (CB_TREE(current_statement),
                                _("The CHARACTERS field of ALLOCATE must be numer
      ic"));
                }
        }
        if (target1) {
                sprintf (buff, "%d", cb_field (target1)->memory_size);
                x = cb_build_numeric_literal (0, (ucharptr)buff, 0);
                cb_emit (cb_build_funcall_3 ("cob_allocate",
                         cb_build_cast_addr_of_addr (target1), target2, x));
        } else {
                cb_emit (cb_build_funcall_3 ("cob_allocate",
                         NULL, target2, size));
        }
        if (initialize && target1) {
                current_statement->handler2 =
                        cb_build_initialize (target1, cb_true, NULL, cb_true, 0);

        }
}
```

7.17.2.59 void cb_emit_arg_number (cb_tree *value*)

Definition at line 3273 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_display_arg_number", value));
}
```

7.17.2.60 void cb_emit_arithmetic (cb_tree *vars*, int *op*, cb_tree *val*)

Definition at line 2094 of file typeck.c.

```
{
        cb_tree         l;
        struct cb_field *f;

        val = cb_check_numeric_value (val);
        if (op) {
                cb_list_map (cb_check_numeric_name, vars);
        } else {
                cb_list_map (cb_check_numeric_edited_name, vars);
        }

        if (cb_validate_one (val)) {
                return;
        }
        if (cb_validate_list (vars)) {
                return;
        }

        if (!CB_BINARY_OP_P (val)) {
                if (op == '+' || op == '-') {
                        if (CB_EXCEPTION_ENABLE (COB_EC_DATA_INCOMPATIBLE) &&
                           (CB_REF_OR_FIELD_P (val))) {
                                f = cb_field (val);
                                if (f->usage == CB_USAGE_DISPLAY ||
                                    f->usage == CB_USAGE_PACKED) {
                                        cb_emit (cb_build_funcall_2 ("cob_check_n
umeric",
                                                                     val,
                                                                     cb_build_string0 ((
ucharptr)(f->name))));
                                }
                        }
                        for (l = vars; l; l = CB_CHAIN (l)) {
                                if (CB_EXCEPTION_ENABLE (COB_EC_DATA_INCOMPATIBLE
) &&
                                   (CB_REF_OR_FIELD_P (CB_VALUE(l)))) {
                                        f = cb_field (CB_VALUE(l));
                                        if (f->usage == CB_USAGE_DISPLAY ||
                                            f->usage == CB_USAGE_PACKED) {
                                                cb_emit (cb_build_funcall_2 ("cob
_check_numeric",
                                                                     CB_VALUE(l),
                                                                     cb_build_string0 ((
ucharptr)(f->name))));
                                        }
                                }
                                if (op == '+') {
                                        CB_VALUE (l) = cb_build_add (CB_VALUE (l)
, val, CB_PURPOSE (l));
                                } else {
                                        CB_VALUE (l) = cb_build_sub (CB_VALUE (l)
, val, CB_PURPOSE (l));
                                }
                        }
                        cb_emit_list (vars);
                        return;
                }
        }

        cb_emit (build_decimal_assign (vars, op, val));
}
```

7.17.2.61 void cb_emit_call (cb_tree *prog*, cb_tree *using*, cb_tree *returning*, cb_tree *on_exception*, cb_tree *not_on_exception*)

Definition at line 3108 of file typeck.c.

```
{
        cb_tree                         l;
        cb_tree                         x;
        const struct system_table       *psyst;
        int                             is_sys_call = 0;

        if (CB_INTRINSIC_P (prog)) {
                if (CB_INTRINSIC(prog)->intr_tab->category !=
      CB_CATEGORY_ALPHANUMERIC) {
                        cb_error (_("Only alphanumeric FUNCTION types are allowed
       here"));
                        return;
                }
        }
        if (returning) {
                if (CB_TREE_CLASS(returning) != CB_CLASS_NUMERIC &&
                    CB_TREE_CLASS(returning) != CB_CLASS_POINTER) {
                        cb_error (_("Invalid RETURNING field"));
                        return;
                }
        }
        for (l = using; l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
                if (x == cb_error_node) {
                        continue;
                }
                if (CB_CONST_P (x) && x != cb_null) {
                        cb_error_x (x, _("Figurative constant invalid here"));
                }
                if ((CB_REFERENCE_P (x) && CB_FIELD_P(CB_REFERENCE(x)->value))
                     || CB_FIELD_P (x)) {
                        if (cb_field (x)->level == 88) {
                                cb_error_x (x, _("'%s' Not a data name"),
      CB_NAME (x));
                                return;
                        }
                        if (cb_warn_call_params &&
                            CB_PURPOSE_INT (l) == CB_CALL_BY_REFERENCE) {
                                if (cb_field (x)->level != 01 &&
                                    cb_field (x)->level != 77) {
                                        cb_warning_x (x, _("'%s' is not 01 or 77
      level item"), CB_NAME (x));
                                }
                        }
                }
        }

        if (CB_LITERAL_P(prog)) {
                for (psyst = (const struct system_table *)&system_tab[0]; psyst->
      syst_name; psyst++) {
                        if (!strcmp((const char *)CB_LITERAL(prog)->data,
                             (const char *)psyst->syst_name)) {
                                if (psyst->syst_params > cb_list_length (using))
      {
                                        cb_error (_("Wrong number of CALL paramet
      ers for '%s'"),
```

```
                                                        (char *)psyst->syst_name);
                                        return;
                                }
                                is_sys_call = 1;
                                break;
                        }
                }
        }

        cb_emit (cb_build_call (prog, using, on_exception, not_on_exception,
                 returning, is_sys_call));
}
```

7.17.2.62 void cb_emit_cancel (cb_tree *prog*)

Definition at line 3177 of file typeck.c.

```
{
        if (cb_validate_one (prog)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_field_cancel", prog));
}
```

7.17.2.63 void cb_emit_close (cb_tree *file,* cb_tree *opt*)

Definition at line 3190 of file typeck.c.

```
{
        if (file == cb_error_node) {
                return;
        }
        file = cb_ref (file);
        if (file == cb_error_node) {
                return;
        }
        current_statement->file = file;
        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        }
        cb_emit (cb_build_funcall_3 ("cob_close", file, opt,
                CB_FILE(file)->file_status));
}
```

7.17.2.64 void cb_emit_command_line (cb_tree *value*)

Definition at line 3282 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return;
```

```
        }
        cb_emit (cb_build_funcall_1 ("cob_display_command_line", value));
}
```

7.17.2.65 void cb_emit_commit (void)

Definition at line 3213 of file typeck.c.

```
{
        cb_emit (cb_build_funcall_0 ("cob_commit"));
}
```

7.17.2.66 void cb_emit_continue (void)

Definition at line 3223 of file typeck.c.

```
{
        cb_emit (cb_build_continue ());
}
```

7.17.2.67 void cb_emit_corresponding (cb_tree(*)(cb_tree f1, cb_tree f2, cb_tree f3) *func*, cb_tree *x1*, cb_tree *x2*, cb_tree *opt*)

Definition at line 2695 of file typeck.c.

```
{
        x1 = cb_check_group_name (x1);
        x2 = cb_check_group_name (x2);

        if (cb_validate_one (x1)) {
                return;
        }
        if (cb_validate_one (x2)) {
                return;
        }

        emit_corresponding (func, x1, x2, opt);
}
```

7.17.2.68 void cb_emit_delete (cb_tree *file*)

Definition at line 3233 of file typeck.c.

```
{
        if (file == cb_error_node) {
                return;
        }
        file = cb_ref (file);
```

```
        if (file == cb_error_node) {
                return;
        }
        current_statement->file = file;
        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        }
        cb_emit (cb_build_funcall_2 ("cob_delete", file, CB_FILE(file)->file_stat
      us));
}
```

7.17.2.69 void cb_emit_display (cb_tree *values*, cb_tree *upon*, cb_tree *no_adv*, cb_tree *pos*, cb_tree *fgc*, cb_tree *bgc*, cb_tree *scroll*, int *dispattrs*)

Definition at line 3291 of file typeck.c.

```
{
        cb_tree l;
        cb_tree x;
        cb_tree line;
        cb_tree column;
        cb_tree p;

        if (cb_validate_list (values)) {
                return;
        }
        if (cb_validate_one (pos)) {
                return;
        }
        if (cb_validate_one (fgc)) {
                return;
        }
        if (cb_validate_one (bgc)) {
                return;
        }
        if (cb_validate_one (scroll)) {
                return;
        }
        for (l = values; l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
                if (x == cb_error_node) {
                        return;
                }

                switch (CB_TREE_TAG (x)) {
                case CB_TAG_LITERAL:
                case CB_TAG_INTRINSIC:
                case CB_TAG_CONST:
                case CB_TAG_STRING:
                case CB_TAG_INTEGER:
                        break;
                case CB_TAG_REFERENCE:
                        if (!CB_FIELD_P(CB_REFERENCE(x)->value)) {
                                cb_error_x (x, _("'%s' is an invalid type for DIS
      PLAY operand"), cb_name (x));
                                return;
                        }
                        break;
```

```
                default:
                        cb_error_x (x, _("Invalid type for DISPLAY operand"));
                        return;
                }
        }
        if (upon == cb_error_node) {
                return;
        }

        x = CB_VALUE (values);
        if ((CB_REF_OR_FIELD_P (x)) &&
             CB_FIELD (cb_ref (x))->storage == CB_STORAGE_SCREEN) {
                output_screen_from (CB_FIELD (cb_ref (x)), 0);
                gen_screen_ptr = 1;
                if (pos) {
                        if (CB_PAIR_P (pos)) {
                                line = CB_PAIR_X (pos);
                                column = CB_PAIR_Y (pos);
                                if (line == NULL) {
                                        line = cb_one;
                                }
                                if (column == NULL) {
                                        column = cb_one;
                                }
                                cb_emit (cb_build_funcall_3 ("cob_screen_display"
, x,
                                        line, column));
                        } else {
                                cb_emit (cb_build_funcall_3 ("cob_screen_display"
, x,
                                        pos, NULL));
                        }
                } else {
                        cb_emit (cb_build_funcall_3 ("cob_screen_display", x,
                                NULL, NULL));
                }
                gen_screen_ptr = 0;
        } else if (pos || fgc || bgc || scroll || dispattrs) {
                if (!pos) {
                        cb_emit (cb_build_funcall_7 ("cob_field_display",
                                CB_VALUE (values), NULL, NULL, fgc, bgc,
                                scroll, cb_int (dispattrs)));
                } else if (CB_PAIR_P (pos)) {
                        line = CB_PAIR_X (pos);
                        column = CB_PAIR_Y (pos);
                        if (line == NULL) {
                                line = cb_one;
                        }
                        if (column == NULL) {
                                column = cb_one;
                        }
                        cb_emit (cb_build_funcall_7 ("cob_field_display",
                                CB_VALUE (values), line, column, fgc, bgc,
                                scroll, cb_int (dispattrs)));
                } else {
                        cb_emit (cb_build_funcall_7 ("cob_field_display",
                                CB_VALUE (values), pos, NULL, fgc, bgc,
                                scroll, cb_int (dispattrs)));
                }
        } else {
                /* DISPLAY x ... [UPON device-name] */
                p = cb_build_funcall_3 ("cob_display", upon, no_adv, values);
```

```
                CB_FUNCALL(p)->varcnt = cb_list_length (values);
                cb_emit (p);
                for (l = values; l; l = CB_CHAIN (l)) {
                        x = CB_VALUE (l);
                        if (CB_FIELD_P (x)) {
                                CB_FIELD (cb_ref (x))->count++;
                        }
                }
        }
}
```

7.17.2.70 void cb_emit_divide (cb_tree *dividend,* cb_tree *divisor,* cb_tree *quotient,* cb_tree *remainder*)

Definition at line 3460 of file typeck.c.

```
{
        if (cb_validate_one (dividend)) {
                return;
        }
        if (cb_validate_one (divisor)) {
                return;
        }
        CB_VALUE (quotient) = cb_check_numeric_edited_name (CB_VALUE (quotient));

        CB_VALUE (remainder) = cb_check_numeric_edited_name (CB_VALUE (remainder)
      );

        if (cb_validate_one (CB_VALUE (quotient))) {
                return;
        }
        if (cb_validate_one (CB_VALUE (remainder))) {
                return;
        }

        cb_emit (cb_build_funcall_4 ("cob_div_quotient", dividend, divisor,
                                     CB_VALUE (quotient),
                                     build_store_option (CB_VALUE (quotient),
      CB_PURPOSE (quotient))));
        cb_emit (cb_build_funcall_2 ("cob_div_remainder", CB_VALUE (remainder),
                                     build_store_option (CB_VALUE (remainder),
      cb_int0)));
}
```

7.17.2.71 void cb_emit_env_name (cb_tree *value*)

Definition at line 3255 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_display_environment", value));
}
```

7.17.2.72 void cb_emit_env_value (cb_tree *value*)

Definition at line 3264 of file typeck.c.

```
{
        if (cb_validate_one (value)) {
                return;
        }
        cb_emit (cb_build_funcall_1 ("cob_display_env_value", value));
}
```

7.17.2.73 void cb_emit_evaluate (cb_tree *subject_list*, cb_tree *case_list*)

Definition at line 3601 of file typeck.c.

```
{
        cb_emit (build_evaluate (subject_list, case_list));
}
```

7.17.2.74 void cb_emit_exit (size_t *goback*)

Definition at line 3675 of file typeck.c.

```
{
        if (goback) {
                cb_emit (cb_build_goto (cb_int1, NULL));
        } else {
                cb_emit (cb_build_goto (NULL, NULL));
        }
}
```

7.17.2.75 void cb_emit_free (cb_tree *vars*)

Definition at line 3611 of file typeck.c.

```
{
        cb_tree         l;
        struct cb_field *f;
        int             i;

        if (cb_validate_list (vars)) {
                return;
        }
        for (l = vars, i = 1; l; l = CB_CHAIN (l), i++) {
                if (CB_TREE_CLASS (CB_VALUE (l)) == CB_CLASS_POINTER) {
                        if (CB_CAST_P (CB_VALUE (l))) {
                                f = cb_field (CB_CAST (CB_VALUE(l))->val);
                                if (!f->flag_item_based) {
                                        cb_error_x (CB_TREE (current_statement),
                                                _("Target %d of FREE, a data addr
```

```
      ess identifier, must address a BASED data item"), i);
                                }
                                cb_emit (cb_build_funcall_2 ("cob_free_alloc",
                                        cb_build_cast_address (CB_VALUE (l)),
      NULL));
                        } else {
                                cb_emit (cb_build_funcall_2 ("cob_free_alloc",
                                        NULL, cb_build_cast_address (CB_VALUE (l)
      )));
                        }
                } else if (CB_REF_OR_FIELD_P (CB_VALUE (l))) {
                                f = cb_field (CB_VALUE (l));
                                if (!f->flag_item_based) {
                                        cb_error_x (CB_TREE (current_statement),
                                                _("Target %d of FREE, a data addr
      ess identifier, must address a BASED data item"), i);
                                }
                                cb_emit (cb_build_funcall_2 ("cob_free_alloc",
                                        cb_build_cast_addr_of_addr (CB_VALUE (l))
      , NULL));
                } else {
                        cb_error_x (CB_TREE (current_statement),
                                _("Target %d of FREE must be a data pointer"), i)
      ;
                }
          }
}
```

7.17.2.76 void cb_emit_get_environment (cb_tree *envvar,* cb_tree *envval*)

Definition at line 2965 of file typeck.c.

```
{
        if (cb_validate_one (envvar)) {
                return;
        }
        if (cb_validate_one (envval)) {
                return;
        }
        cb_emit (cb_build_funcall_2 ("cob_get_environment", envvar, envval));
}
```

7.17.2.77 void cb_emit_goto (cb_tree *target,* cb_tree *depending*)

Definition at line 3654 of file typeck.c.

```
{
        if (target == cb_error_node) {
                return;
        }
        if (depending) {
                /* GO TO procedure-name ... DEPENDING ON identifier */
                cb_emit (cb_build_goto (target, depending));
        } else {
                /* GO TO procedure-name */
```

```
                if (target == NULL) {
                        cb_verify (cb_goto_statement_without_name, "GO TO without
       procedure-name");
                } else if (CB_CHAIN (target)) {
                        cb_error (_("GO TO with multiple procedure-names"));
                } else {
                        cb_emit (cb_build_goto (CB_VALUE (target), NULL));
                }
        }
}
```

7.17.2.78 void cb_emit_if (cb_tree *cond*, cb_tree *stmt1*, cb_tree *stmt2*)

Definition at line 3689 of file typeck.c.

```
{
        cb_emit (cb_build_if (cond, stmt1, stmt2));
}
```

7.17.2.79 void cb_emit_initialize (cb_tree *vars*, cb_tree *fillinit*, cb_tree *value*, cb_tree *replacing*, cb_tree *def*)

Definition at line 3699 of file typeck.c.

```
{
        cb_tree l;
        int fill_init = 1;

        if (cb_validate_list (vars)) {
                return;
        }
        if (value == NULL && replacing == NULL) {
                def = cb_true;
        }
        if (fillinit == cb_true) {
                fill_init = 0;
        }
        for (l = vars; l; l = CB_CHAIN (l)) {
                cb_emit (cb_build_initialize (CB_VALUE (l), value, replacing, def
      , fill_init));
        }
}
```

7.17.2.80 void cb_emit_inspect (cb_tree *var*, cb_tree *body*, cb_tree *replacing*, int *replconv*)

Definition at line 3723 of file typeck.c.

```
{
        switch (CB_TREE_TAG(var)) {
        case CB_TAG_REFERENCE:
```

```
                break;
        case CB_TAG_INTRINSIC:
                switch (CB_TREE_CATEGORY(var)) {
                case CB_CATEGORY_ALPHABETIC:
                case CB_CATEGORY_ALPHANUMERIC:
                case CB_CATEGORY_NATIONAL:
                        break;
                default:
                        cb_error (_("Invalid target for INSPECT"));
                        return;
                }
                break;
        case CB_TAG_LITERAL:
                break;
        default:
                cb_error (_("Invalid target for REPLACING/CONVERTING"));
                return;
        }
        if (replconv && sending_id) {
                cb_error (_("Invalid target for REPLACING/CONVERTING"));
        }
        cb_emit (cb_build_funcall_2 ("cob_inspect_init", var, replacing));
        cb_emit_list (body);
        cb_emit (cb_build_funcall_0 ("cob_inspect_finish"));
}
```

7.17.2.81 void cb_emit_move (cb_tree *src*, cb_tree *dsts*)

Definition at line 5041 of file typeck.c.

```
{
        cb_tree l;

        if (cb_validate_one (src)) {
                return;
        }
        if (cb_validate_list (dsts)) {
                return;
        }

        for (l = dsts; l; l = CB_CHAIN (l)) {
                cb_emit (cb_build_move (src, CB_VALUE (l)));
        }
}
```

7.17.2.82 void cb_emit_move_corresponding (cb_tree *x1*, cb_tree *x2*)

Definition at line 2738 of file typeck.c.

```
{
        cb_tree         l;
        cb_tree         v;

        x1 = cb_check_group_name (x1);
        if (cb_validate_one (x1)) {
```

```
                return;
        }
        for (l = x2; l; l = CB_CHAIN(l)) {
                v = CB_VALUE(l);
                v = cb_check_group_name (v);
                if (cb_validate_one (v)) {
                        return;
                }
                emit_move_corresponding (x1, v);
        }
}
```

7.17.2.83 void cb_emit_open (cb_tree *file*, cb_tree *mode*, cb_tree *sharing*)

Definition at line 5062 of file typeck.c.

```
{
        if (file == cb_error_node) {
                return;
        }
        file = cb_ref (file);
        if (file == cb_error_node) {
                return;
        }
        current_statement->file = file;

        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        }
        if (sharing == NULL) {
                sharing = CB_FILE (file)->sharing ? CB_FILE (file)->sharing :
      cb_int0;
        }

        /* READ ONLY */
        if (sharing == cb_int0 && CB_INTEGER (mode)->val != COB_OPEN_INPUT) {
                sharing = cb_int1;
        }

        cb_emit (cb_build_funcall_4 ("cob_open", file, mode,
                 sharing, CB_FILE(file)->file_status));
}
```

7.17.2.84 void cb_emit_perform (cb_tree *perform*, cb_tree *body*)

Definition at line 5095 of file typeck.c.

```
{
        if (perform == cb_error_node) {
                return;
        }
        CB_PERFORM (perform)->body = body;
        cb_emit (perform);
}
```

7.17.2.85 void cb_emit_read (cb_tree *ref*, cb_tree *next*, cb_tree *into*, cb_tree *key*, cb_tree *lock_opts*)

Definition at line 5170 of file typeck.c.

```
{
        int     read_opts = 0;
        cb_tree file;
        cb_tree rec;

        if (lock_opts == cb_int1) {
                read_opts = COB_READ_LOCK;
        } else if (lock_opts == cb_int2) {
                read_opts = COB_READ_NO_LOCK;
        } else if (lock_opts == cb_int3) {
                read_opts = COB_READ_IGNORE_LOCK;
        } else if (lock_opts == cb_int4) {
                read_opts = COB_READ_WAIT_LOCK;
        }
        if (ref == cb_error_node) {
                return;
        }
        file = cb_ref (ref);
        if (file == cb_error_node) {
                return;
        }
        rec = cb_build_field_reference (CB_FILE (file)->record, ref);
        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        }
        if (next == cb_int1 || next == cb_int2 ||
            CB_FILE (file)->access_mode == COB_ACCESS_SEQUENTIAL) {
                /* READ NEXT/PREVIOUS */
                if (next == cb_int2) {
                        if (CB_FILE (file)->organization != COB_ORG_INDEXED) {
                                cb_error_x (CB_TREE (current_statement),
                                _("READ PREVIOUS only allowed for INDEXED SEQUENT
      IAL files"));
                        }
                        read_opts |= COB_READ_PREVIOUS;
                } else {
                        read_opts |= COB_READ_NEXT;
                }
                if (key) {
                        cb_warning (_("KEY ignored with sequential READ"));
                }
                cb_emit (cb_build_funcall_4 ("cob_read", file, cb_int0,
                         CB_FILE(file)->file_status,
                         cb_int (read_opts)));
        } else {
                /* READ */
                cb_emit (cb_build_funcall_4 ("cob_read",
                         file, key ? key : CB_FILE (file)->key,
                         CB_FILE(file)->file_status, cb_int (read_opts)));
        }
        if (into) {
                current_statement->handler3 = cb_build_move (rec, into);
        }
        current_statement->file = file;
}
```

7.17.2.86 void cb_emit_release (cb_tree *record,* cb_tree *from*)

Definition at line 5278 of file typeck.c.

```
{
        struct cb_field *f;
        cb_tree         file;

        if (record == cb_error_node) {
                return;
        }
        if (from == cb_error_node) {
                return;
        }
        if (cb_ref (record) == cb_error_node) {
                return;
        }
        if (!CB_REF_OR_FIELD_P (cb_ref (record))) {
                cb_error_x (CB_TREE (current_statement),
                        _("RELEASE requires a record name as subject"));
                return;
        }
        if (cb_field (record)->storage != CB_STORAGE_FILE) {
                cb_error_x (CB_TREE (current_statement),
                        _("RELEASE subject does not refer to a record name"));
                return;
        }
        f = CB_FIELD (cb_ref (record));
        file = CB_TREE (f->file);
        if (CB_FILE (file)->organization != COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                        _("RELEASE not allowed on this record item"));
                return;
        }
        current_statement->file = file;
        if (from) {
                cb_emit (cb_build_move (from, record));
        }
        cb_emit (cb_build_funcall_1 ("cob_file_release", file));
}
```

7.17.2.87 void cb_emit_return (cb_tree *ref,* cb_tree *into*)

Definition at line 5321 of file typeck.c.

```
{
        cb_tree file;
        cb_tree rec;

        if (ref == cb_error_node) {
                return;
        }
        if (into == cb_error_node) {
                return;
        }
        file = cb_ref (ref);
        if (file == cb_error_node) {
                return;
```

```
        }
        rec = cb_build_field_reference (CB_FILE (file)->record, ref);
        cb_emit (cb_build_funcall_1 ("cob_file_return", file));
        if (into) {
                current_statement->handler3 = cb_build_move (rec, into);
        }
        current_statement->file = file;
}
```

7.17.2.88 void cb_emit_rewrite (cb_tree *record,* cb_tree *from,* cb_tree *lockopt*)

Definition at line 5232 of file typeck.c.

```
{
        cb_tree file;
        int     opts = 0;

        if (record == cb_error_node || cb_ref (record) == cb_error_node) {
                return;
        }
        if (!CB_REF_OR_FIELD_P (cb_ref (record))) {
                cb_error_x (CB_TREE (current_statement),
                        _("REWRITE requires a record name as subject"));
                return;
        }
        if (cb_field (record)->storage != CB_STORAGE_FILE) {
                cb_error_x (CB_TREE (current_statement),
                        _("REWRITE subject does not refer to a record name"));
                return;
        }
        file = CB_TREE (CB_FIELD (cb_ref (record))->file);
        current_statement->file = file;
        if (CB_FILE (file)->organization == COB_ORG_SORT) {
                cb_error_x (CB_TREE (current_statement),
                _("Operation not allowed on SORT files"));
        } else if (current_statement->handler_id == COB_EC_I_O_INVALID_KEY &&
                   (CB_FILE(file)->organization != COB_ORG_RELATIVE &&
                    CB_FILE(file)->organization != COB_ORG_INDEXED)) {
                        cb_error_x (CB_TREE(current_statement),
                        _("INVALID KEY clause invalid with this file type"));
        } else if ((CB_FILE (file)->lock_mode & COB_LOCK_AUTOMATIC) && lockopt) {

                cb_error_x (CB_TREE (current_statement),
                _("LOCK clause invalid with file LOCK AUTOMATIC"));
        } else if (lockopt == cb_int1) {
                opts = COB_WRITE_LOCK;
        }
        if (from) {
                cb_emit (cb_build_move (from, record));
        }
        cb_emit (cb_build_funcall_4 ("cob_rewrite", file, record,
                        cb_int (opts), CB_FILE(file)->file_status));
}
```

7.17.2.89 void cb_emit_rollback (void)

Definition at line 5349 of file typeck.c.

```
{
        cb_emit (cb_build_funcall_0 ("cob_rollback"));
}
```

7.17.2.90 void cb_emit_search (cb_tree *table*, cb_tree *varying*, cb_tree *at_end*, cb_tree *whens*)

Definition at line 5427 of file typeck.c.

```
{
        if (cb_validate_one (table)) {
                return;
        }
        if (cb_validate_one (varying)) {
                return;
        }
        if (table == cb_error_node) {
                return;
        }
        cb_emit (cb_build_search (0, table, varying, at_end, whens));
}
```

7.17.2.91 void cb_emit_search_all (cb_tree *table*, cb_tree *at_end*, cb_tree *when*, cb_tree *stmts*)

Definition at line 5442 of file typeck.c.

```
{
        if (cb_validate_one (table)) {
                return;
        }
        if (table == cb_error_node) {
                return;
        }
        cb_emit (cb_build_search (1, table, NULL, at_end,
                                  cb_build_if (cb_build_search_all (table, when),
       stmts, NULL)));
}
```

7.17.2.92 void cb_emit_set_false (cb_tree *l*)

Definition at line 5604 of file typeck.c.

```
{
        cb_tree         x;
        struct cb_field *f;
        cb_tree         ref;
        cb_tree         val;

        for (; l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
```

```
                if (x == cb_error_node) {
                        return;
                }
                if (!(CB_REFERENCE_P (x) && CB_FIELD_P(CB_REFERENCE(x)->value))
                     && !CB_FIELD_P (x)) {
                        cb_error_x (x, _("Invalid SET statement"));
                        return;
                }
                f = cb_field (x);
                if (f->level != 88) {
                        cb_error_x (x, _("Invalid SET statement"));
                        return;
                }
                if (!f->false_88) {
                        cb_error_x (x, _("Field does not have FALSE clause"));
                        return;
                }
                ref = cb_build_field_reference (f->parent, x);
                val = CB_VALUE (f->false_88);
                if (CB_PAIR_P (val)) {
                        val = CB_PAIR_X (val);
                }
                cb_emit (cb_build_move (val, ref));
        }
}
```

7.17.2.93 void cb_emit_set_on_off (cb_tree *l*, cb_tree *flag*)

Definition at line 5558 of file typeck.c.

```
{
        struct cb_system_name *s;

        if (cb_validate_list (l)) {
                return;
        }
        for (; l; l = CB_CHAIN (l)) {
                s = CB_SYSTEM_NAME (cb_ref (CB_VALUE (l)));
                cb_emit (cb_build_funcall_2 ("cob_set_switch", cb_int (s->token),
       flag));
        }
}
```

7.17.2.94 void cb_emit_set_to (cb_tree *vars*, cb_tree *x*)

Definition at line 5465 of file typeck.c.

```
{
        cb_tree         l;
        cb_tree         v;
        struct cb_cast  *p;
#if 0
        enum cb_class class = CB_CLASS_UNKNOWN;
#endif
```

```
        if (cb_validate_one (x)) {
                return;
        }
        if (cb_validate_list (vars)) {
                return;
        }

#if 0
        /* determine the class of targets */
        for (l = vars; l; l = CB_CHAIN (l)) {
                if (CB_TREE_CLASS (CB_VALUE (l)) != CB_CLASS_UNKNOWN) {
                        if (class == CB_CLASS_UNKNOWN) {
                                class = CB_TREE_CLASS (CB_VALUE (l));
                        } else if (class != CB_TREE_CLASS (CB_VALUE (l))) {
                                break;
                        }
                }
        }
        if (l || (class != CB_CLASS_INDEX && class != CB_CLASS_POINTER)) {
                cb_error_x (CB_TREE (current_statement),
                            _("The targets of SET must be either indexes or point
      ers"));
                return;
        }
#endif

        if (CB_CAST_P (x)) {
                p = CB_CAST (x);
                if (p->type == CB_CAST_PROGRAM_POINTER) {
                        for (l = vars; l; l = CB_CHAIN (l)) {
                                v = CB_VALUE (l);
                                if (!CB_REFERENCE_P (v)) {
                                        cb_error_x (CB_TREE (current_statement),
                                        _("SET targets must be PROGRAM-POINTER"))
      ;
                                        CB_VALUE (l) = cb_error_node;
                                } else if (CB_FIELD(cb_ref(v))->usage !=
      CB_USAGE_PROGRAM_POINTER) {
                                        cb_error_x (CB_TREE (current_statement),
                                        _("SET targets must be PROGRAM-POINTER"))
      ;
                                        CB_VALUE (l) = cb_error_node;
                                }
                        }
                }
        }
        /* validate the targets */
        for (l = vars; l; l = CB_CHAIN (l)) {
                v = CB_VALUE (l);
                if (CB_CAST_P (v)) {
                        p = CB_CAST (v);
                        if (p->type == CB_CAST_ADDRESS
                            && !CB_FIELD (cb_ref (p->val))->flag_item_based
                            && CB_FIELD (cb_ref (p->val))->storage !=
      CB_STORAGE_LINKAGE) {
                                cb_error_x (p->val, _("The address of '%s' cannot
       be changed"),
                                            cb_name (p->val));
                                CB_VALUE (l) = cb_error_node;
                        }
                }
        }
```

```
        if (cb_validate_list (vars)) {
                return;
        }

        for (l = vars; l; l = CB_CHAIN (l)) {
                cb_emit (cb_build_move (x, CB_VALUE (l)));
        }
}
```

7.17.2.95 void cb_emit_set_true (cb_tree *l*)

Definition at line 5572 of file typeck.c.

```
{
        cb_tree         x;
        struct cb_field *f;
        cb_tree         ref;
        cb_tree         val;

        for (; l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
                if (x == cb_error_node) {
                        return;
                }
                if (!(CB_REFERENCE_P (x) && CB_FIELD_P(CB_REFERENCE(x)->value))
                     && !CB_FIELD_P (x)) {
                        cb_error_x (x, _("Invalid SET statement"));
                        return;
                }
                f = cb_field (x);
                if (f->level != 88) {
                        cb_error_x (x, _("Invalid SET statement"));
                        return;
                }
                ref = cb_build_field_reference (f->parent, x);
                val = CB_VALUE (f->values);
                if (CB_PAIR_P (val)) {
                        val = CB_PAIR_X (val);
                }
                cb_emit (cb_build_move (val, ref));
        }
}
```

7.17.2.96 void cb_emit_set_up_down (cb_tree *l*, cb_tree *flag*, cb_tree *x*)

Definition at line 5540 of file typeck.c.

```
{
        if (cb_validate_one (x)) {
                return;
        }
        if (cb_validate_list (l)) {
                return;
        }
        for (; l; l = CB_CHAIN (l)) {
```

```
                if (flag == cb_int0) {
                        cb_emit (cb_build_add (CB_VALUE (l), x, cb_int0));
                } else {
                        cb_emit (cb_build_sub (CB_VALUE (l), x, cb_int0));
                }
        }
}
```

7.17.2.97 void cb_emit_setenv (cb_tree *x*, cb_tree *y*)

Definition at line 5459 of file typeck.c.

```
{
        cb_emit (cb_build_funcall_2 ("cob_set_environment", x, y));
}
```

7.17.2.98 void cb_emit_sort_finish (cb_tree *file*)

Definition at line 5741 of file typeck.c.

```
{
        if (CB_FILE_P (cb_ref (file))) {
                cb_emit (cb_build_funcall_1 ("cob_file_sort_close", cb_ref (file)
      ));
        }
}
```

7.17.2.99 void cb_emit_sort_giving (cb_tree *file*, cb_tree *l*)

Definition at line 5715 of file typeck.c.

```
{
        cb_tree         p;
        int             listlen;

        if (cb_validate_list (l)) {
                return;
        }
        for (p = l; p; p = CB_CHAIN (p)) {
                if (CB_FILE (cb_ref(CB_VALUE(p)))->organization == COB_ORG_SORT)
      {
                        cb_error (_("Invalid SORT GIVING parameter"));
                }
        }
        listlen = cb_list_length (l);
        p = cb_build_funcall_2 ("cob_file_sort_giving", cb_ref (file), l);
        CB_FUNCALL(p)->varcnt = listlen;
        cb_emit (p);
}
```

7.17.2.100 void cb_emit_sort_init (cb_tree *name*, cb_tree *keys*, cb_tree *col*)

Definition at line 5644 of file typeck.c.

```
{
        cb_tree         l;
        struct cb_field *f;

        if (cb_validate_list (keys)) {
                return;
        }
        for (l = keys; l; l = CB_CHAIN (l)) {
                if (CB_VALUE (l) == NULL) {
                        CB_VALUE (l) = name;
                }
                cb_ref (CB_VALUE (l));
        }

        if (CB_FILE_P (cb_ref (name))) {
                if (CB_FILE (cb_ref (name))->organization != COB_ORG_SORT) {
                        cb_error_x (name, _("Invalid SORT filename"));
                }
                cb_field (current_program->cb_sort_return)->count++;
                cb_emit (cb_build_funcall_5 ("cob_file_sort_init", cb_ref (name),

                                             cb_int (cb_list_length (keys)), col,

                                             cb_build_cast_address (
      current_program->cb_sort_return),
                                             CB_FILE(cb_ref (name))->file_status)
      );
                for (l = keys; l; l = CB_CHAIN (l)) {
                        cb_emit (cb_build_funcall_4 ("cob_file_sort_init_key",
      cb_ref (name),
                                        CB_PURPOSE (l),
                                        CB_VALUE (l),
                                        cb_int (cb_field (CB_VALUE(l))->offset)))
      ;
                }
        } else {
                f = CB_FIELD (cb_ref (name));
                if (keys == NULL) {
                        cb_error_x (name, _("Table sort without keys not implemen
      ted yet"));
                }
                cb_emit (cb_build_funcall_2 ("cob_table_sort_init", cb_int (
      cb_list_length (keys)), col));
                for (l = keys; l; l = CB_CHAIN (l)) {
                        cb_emit (cb_build_funcall_3 ("cob_table_sort_init_key",
                                        CB_PURPOSE (l),
                                        CB_VALUE (l),
                                        cb_int (cb_field (CB_VALUE(l))->offset)))
      ;
                }
                cb_emit (cb_build_funcall_2 ("cob_table_sort", name,
                                             (f->occurs_depending
                                              ? cb_build_cast_integer (f->
      occurs_depending)
                                              : cb_int (f->occurs_max))));
        }
}
```

7.17.2.101 void cb_emit_sort_input (cb_tree *proc*)

Definition at line 5709 of file typeck.c.

```
{
        cb_emit (cb_build_perform_once (proc));
}
```

7.17.2.102 void cb_emit_sort_output (cb_tree *proc*)

Definition at line 5735 of file typeck.c.

```
{
        cb_emit (cb_build_perform_once (proc));
}
```

7.17.2.103 void cb_emit_sort_using (cb_tree *file,* cb_tree *l*)

Definition at line 5694 of file typeck.c.

```
{
        if (cb_validate_list (l)) {
                return;
        }
        for (; l; l = CB_CHAIN (l)) {
                if (CB_FILE (cb_ref(CB_VALUE(l)))->organization == COB_ORG_SORT)
      {
                        cb_error (_("Invalid SORT USING parameter"));
                }
                cb_emit (cb_build_funcall_2 ("cob_file_sort_using",
                        cb_ref (file), cb_ref (CB_VALUE (l))));
        }
}
```

7.17.2.104 void cb_emit_start (cb_tree *file,* cb_tree *op,* cb_tree *key*)

Definition at line 5753 of file typeck.c.

```
{
        if (cb_validate_one (key)) {
                return;
        }
        if (file != cb_error_node) {
                current_statement->file = cb_ref (file);
                cb_emit (cb_build_funcall_4 ("cob_start", cb_ref (file), op,
                                             key ? key : CB_FILE (cb_ref (file))-
      >key,
                                                CB_FILE(cb_ref(file))->file_statu
      s));
        }
}
```

7.17.2.105 void cb_emit_stop_run (cb_tree *x*)

Definition at line 5771 of file typeck.c.

```
{
        cb_emit (cb_build_funcall_1 ("cob_stop_run", cb_build_cast_integer (x)));

}
```

7.17.2.106 void cb_emit_string (cb_tree *items*, cb_tree *into*, cb_tree *pointer*)

Definition at line 5781 of file typeck.c.

```
{
        cb_tree start;
        cb_tree l;
        cb_tree end;
        cb_tree dlm;

        if (cb_validate_one (into)) {
                return;
        }
        if (cb_validate_one (pointer)) {
                return;
        }
        start = items;
        cb_emit (cb_build_funcall_2 ("cob_string_init", into, pointer));
        while (start) {

                /* find DELIMITED item */
                for (end = start; end; end = CB_CHAIN (end)) {
                        if (CB_PAIR_P (CB_VALUE (end))) {
                                break;
                        }
                }

                /* cob_string_delimited */
                dlm = end ? CB_PAIR_X (CB_VALUE (end)) : cb_int0;
                cb_emit (cb_build_funcall_1 ("cob_string_delimited", dlm));

                /* cob_string_append */
                for (l = start; l != end; l = CB_CHAIN (l)) {
                        cb_emit (cb_build_funcall_1 ("cob_string_append",
      CB_VALUE (l)));
                }

                start = end ? CB_CHAIN (end) : NULL;
        }
        cb_emit (cb_build_funcall_0 ("cob_string_finish"));
}
```

7.17.2.107 void cb_emit_unlock (cb_tree *ref*)

Definition at line 5824 of file typeck.c.

```
{
        cb_tree file;

        if (ref != cb_error_node) {
                file = cb_ref (ref);
                cb_emit (cb_build_funcall_2 ("cob_unlock_file",
                         file, CB_FILE(file)->file_status));
                current_statement->file = file;
        }
}
```

7.17.2.108 void cb_emit_unstring (cb_tree *name*, cb_tree *delimited*, cb_tree *into*, cb_tree *pointer*, cb_tree *tallying*)

Definition at line 5841 of file typeck.c.

```
{
        if (cb_validate_one (name)) {
                return;
        }
        if (cb_validate_one (tallying)) {
                return;
        }
        if (cb_validate_list (delimited)) {
                return;
        }
        if (cb_validate_list (into)) {
                return;
        }
        cb_emit (cb_build_funcall_3 ("cob_unstring_init", name, pointer,
                cb_int (cb_list_length (delimited))));
        cb_emit_list (delimited);
        cb_emit_list (into);
        if (tallying) {
                cb_emit (cb_build_funcall_1 ("cob_unstring_tallying", tallying));

        }
        cb_emit (cb_build_funcall_0 ("cob_unstring_finish"));
}
```

7.17.2.109 void cb_emit_write (cb_tree *record*, cb_tree *from*, cb_tree *opt*, cb_tree *lockopt*)

Definition at line 5894 of file typeck.c.

```
{
        cb_tree         file;
        int             val;

        if (record != cb_error_node && cb_ref (record) != cb_error_node) {
                if (!CB_REF_OR_FIELD_P (cb_ref (record))) {
                        cb_error_x (CB_TREE (current_statement),
                                _("WRITE requires a record name as subject"));
                        return;
                }
```

```
                if (cb_field (record)->storage != CB_STORAGE_FILE) {
                        cb_error_x (CB_TREE (current_statement),
                                _("WRITE subject does not refer to a record name"
      ));
                        return;
                }
                file = CB_TREE (CB_FIELD (cb_ref (record))->file);
                current_statement->file = file;
                if (CB_FILE (file)->organization == COB_ORG_SORT) {
                        cb_error_x (CB_TREE (current_statement),
                        _("Operation not allowed on SORT files"));
                } else if (current_statement->handler_id == COB_EC_I_O_INVALID_KE
      Y &&
                          (CB_FILE(file)->organization != COB_ORG_RELATIVE &&
                           CB_FILE(file)->organization != COB_ORG_INDEXED)) {
                                cb_error_x (CB_TREE(current_statement),
                                _("INVALID KEY clause invalid with this file type
      "));
                } else if (lockopt) {
                        if ((CB_FILE (file)->lock_mode & COB_LOCK_AUTOMATIC)) {
                                cb_error_x (CB_TREE (current_statement),
                                _("LOCK clause invalid with file LOCK AUTOMATIC")
      );
                        } else if (opt != cb_int0) {
                                cb_error_x (CB_TREE (current_statement),
                                _("LOCK clause invalid here"));
                        } else if (lockopt == cb_int1) {
                                opt = cb_int (COB_WRITE_LOCK);
                        }
                }
                if (from) {
                        cb_emit (cb_build_move (from, record));
                }
                if (CB_FILE (file)->organization == COB_ORG_LINE_SEQUENTIAL &&
                    opt == cb_int0) {
                        opt = cb_int (COB_WRITE_BEFORE | COB_WRITE_LINES | 1);
                }
                /* RXW - This is horrible */
                if (current_statement->handler_id == COB_EC_I_O_EOP &&
                    current_statement->handler1) {
                        if (CB_CAST_P(opt)) {
                                val = CB_INTEGER(CB_BINARY_OP(CB_CAST(opt)->val)-
      >x)->val;
                                val |= COB_WRITE_EOP;
                                CB_BINARY_OP(CB_CAST(opt)->val)->x = cb_int (val)
      ;
                        } else {
                                val = CB_INTEGER(opt)->val;
                                val |= COB_WRITE_EOP;
                                opt = cb_int (val);
                        }
                }
                cb_emit (cb_build_funcall_4 ("cob_write", file, record, opt,
                                             CB_FILE(file)->file_status));
        }
}
```

7.17.2.110 char* cb_encode_program_id (const char * *name*)

Definition at line 563 of file typeck.c.

```
{
        unsigned char           *p;
        const unsigned char     *s;
        unsigned char           buff[COB_SMALL_BUFF];

        p = buff;
        s = (const unsigned char *)name;
        /* encode the initial digit */
        if (isdigit (*s)) {
                p += sprintf ((char *)p, "_%02X", *s++);
        }
        /* encode invalid letters */
        for (; *s; s++) {
                if (isalnum (*s) || *s == '_') {
                        *p++ = *s;
                } else if (*s == '-') {
                        *p++ = '_';
                        *p++ = '_';
                } else {
                        p += sprintf ((char *)p, "_%02X", *s);
                }
        }
        *p = 0;
        return strdup ((char *)buff);
}
```

7.17.2.111 void cb_init_tarrying (void)

Definition at line 3754 of file typeck.c.

```
{
        inspect_func = NULL;
        inspect_data = NULL;
}
```

7.17.2.112 void cb_validate_program_body (struct cb_program * *prog*)

Definition at line 1416 of file typeck.c.

```
{
        /* resolve all labels */
        cb_tree l;
        cb_tree x;
        cb_tree v;

        for (l = cb_list_reverse (prog->label_list); l; l = CB_CHAIN (l)) {
                x = CB_VALUE (l);
                v = cb_ref (x);
                if (CB_LABEL_P (v)) {
                        CB_LABEL (v)->need_begin = 1;
                        if (CB_REFERENCE (x)->length) {
                                CB_LABEL (v)->need_return = 1;
                        }
                } else if (v != cb_error_node) {
                        cb_error_x (x, _("'%s' not procedure name"), cb_name (x))
```

```
      ;
                }
        }

        prog->file_list = cb_list_reverse (prog->file_list);
        prog->exec_list = cb_list_reverse (prog->exec_list);
}
```

7.17.2.113 void cb_validate_program_data (struct cb_program * *prog*)

Definition at line 1286 of file typeck.c.

```
{
        cb_tree         l;
        cb_tree         x;
        cb_tree         assign;
        struct cb_field *p;
        struct cb_file  *f;
        unsigned char   *c;

        for (l = current_program->file_list; l; l = CB_CHAIN (l)) {
                f = CB_FILE (CB_VALUE (l));
                if (!f->finalized) {
                        finalize_file (f, NULL);
                }
        }
        /* build undeclared assignment name now */
        if (cb_assign_clause == CB_ASSIGN_MF) {
                for (l = current_program->file_list; l; l = CB_CHAIN (l)) {
                        assign = CB_FILE (CB_VALUE (l))->assign;
                        if (!assign) {
                                continue;
                        }
                        if (CB_REFERENCE_P (assign)) {
                                for (x = current_program->file_list; x; x =
      CB_CHAIN (x)) {
                                        if (!strcmp (CB_FILE (CB_VALUE (x))->
      name,
                                             CB_REFERENCE (assign)->word->name))
      {
                                                redefinition_error (assign);
                                        }
                                }
                                p = check_level_78 (CB_REFERENCE (assign)->word->
      name);
                                if (p) {
                                        c = (unsigned char *)CB_LITERAL(CB_VALUE(
      p->values))->data;
                                        assign = CB_TREE (build_literal (
      CB_CATEGORY_ALPHANUMERIC, c, strlen ((char *)c)));
                                        CB_FILE (CB_VALUE (l))->assign = assign;
                                }
                        }
                        if (CB_REFERENCE_P (assign) && CB_REFERENCE (assign)->wor
      d->count == 0) {
                                if (cb_warn_implicit_define) {
                                        cb_warning (_("'%s' will be implicitly de
      fined"), CB_NAME (assign));
                                }
```

```
                        x = cb_build_implicit_field (assign,
COB_SMALL_BUFF);
                        p = current_program->working_storage;
                        CB_FIELD (x)->count++;
                        if (p) {
                                while (p->sister) {
                                        p = p->sister;
                                }
                                p->sister = CB_FIELD (x);
                        } else {
                                current_program->working_storage =
CB_FIELD (x);
                        }
                }
                if (CB_REFERENCE_P (assign)) {
                        x = cb_ref (assign);
                        if (CB_FIELD_P (x) && CB_FIELD (x)->level == 88)
{
                                cb_error_x (assign, _("ASSIGN data item '
%s' invalid"), CB_NAME (assign));
                        }
                }
        }
  }

  if (prog->cursor_pos) {
          x = cb_ref (prog->cursor_pos);
          if (x == cb_error_node) {
                  prog->cursor_pos = NULL;
          } else if (CB_FIELD(x)->size != 6 && CB_FIELD(x)->size != 4) {
                  cb_error_x (prog->cursor_pos, _("'%s' CURSOR is not 4 or
6 characters long"),
                              cb_name (prog->cursor_pos));
                  prog->cursor_pos = NULL;
          }
  }
  if (prog->crt_status) {
          x = cb_ref (prog->crt_status);
          if (x == cb_error_node) {
                  prog->crt_status = NULL;
          } else if (CB_FIELD(x)->size != 4) {
                  cb_error_x (prog->crt_status, _("'%s' CRT STATUS is not 4
 characters long"),
                              cb_name (prog->crt_status));
                  prog->crt_status = NULL;
          }
  } else {
          l = cb_build_reference ("COB-CRT-STATUS");
          p = CB_FIELD (cb_build_field (l));
          p->usage = CB_USAGE_DISPLAY;
          p->pic = CB_PICTURE (cb_build_picture ("9(4)"));
          cb_validate_field (p);
          p->flag_no_init = 1;
          /* Do not initialize/bump ref count here
          p->values = cb_list_init (cb_zero);
          p->count++;
          */
          current_program->working_storage =
                  cb_field_add (current_program->working_storage, p);
          prog->crt_status = l;
          /* RXWRXW - Maybe better
          prog->crt_status = cb_build_index (cb_build_reference ("COB-CRT-S
```

```
      TATUS"), cb_zero, 0, NULL);
                */
        }

        /* resolve all references so far */
        for (l = cb_list_reverse (prog->reference_list); l; l = CB_CHAIN (l)) {
                cb_ref (CB_VALUE (l));
        }
        for (l = current_program->file_list; l; l = CB_CHAIN (l)) {
                f = CB_FILE (CB_VALUE (l));
                if (f->record_depending && f->record_depending != cb_error_node)
      {
                        x = f->record_depending;
                        if (cb_ref (x) != cb_error_node) {
/* RXW - This breaks old legacy programs
                                if (CB_REF_OR_FIELD_P(x)) {
                                        p = cb_field (x);
                                        switch (p->storage) {
                                        case CB_STORAGE_WORKING:
                                        case CB_STORAGE_LOCAL:
                                        case CB_STORAGE_LINKAGE:
                                                break;
                                        default:
                                                cb_error (_("RECORD DEPENDING ite
      m must be in WORKING/LOCAL/LINKAGE section"));
                                        }
                                } else {
*/
                                if (!CB_REFERENCE_P(x) && !CB_FIELD_P(x)) {
                                        cb_error (_("Invalid RECORD DEPENDING ite
      m"));
                                }
                        }
                }
        }
}
```

7.17.2.114 void cb_validate_program_environment (struct cb_program * *prog*)

Definition at line 1079 of file typeck.c.

```
{
        cb_tree                 x;
        cb_tree                 y;
        cb_tree                 l;
        cb_tree                 ls;
        struct cb_alphabet_name *ap;
        unsigned char           *data;
        size_t                  dupls;
        size_t                  unvals;
        size_t                  count;
        int                     lower;
        int                     upper;
        int                     size;
        int                     n;
        int                     i;
        int                     lastval;
        int                     values[256];
```

```
	/* Check ALPHABET clauses */
	for (l = current_program->alphabet_name_list; l; l = CB_CHAIN (l)) {
		ap = CB_ALPHABET_NAME (CB_VALUE (l));
		if (ap->type != CB_ALPHABET_CUSTOM) {
			continue;
		}
		ap->low_val_char = 0;
		ap->high_val_char = 255;
		dupls = 0;
		unvals = 0;
		count = 0;
		lastval = 0;
		for (n = 0; n < 256; n++) {
			values[n] = -1;
		}
		for (y = ap->custom_list; y; y = CB_CHAIN (y)) {
			if (count > 255) {
				unvals = 1;
				break;
			}
			x = CB_VALUE (y);
			if (CB_PAIR_P (x)) {
				/* X THRU Y */
				lower = get_value (CB_PAIR_X (x));
				upper = get_value (CB_PAIR_Y (x));
				lastval = upper;
				if (!count) {
					ap->low_val_char = lower;
				}
				if (lower < 0 || lower > 255) {
					unvals = 1;
					continue;
				}
				if (upper < 0 || upper > 255) {
					unvals = 1;
					continue;
				}
				if (lower <= upper) {
					for (i = lower; i <= upper; i++) {
						if (values[i] != -1) {
							dupls = 1;
						}
						values[i] = i;
						count++;
					}
				} else {
					for (i = lower; i >= upper; i--) {
						if (values[i] != -1) {
							dupls = 1;
						}
						values[i] = i;
						count++;
					}
				}
			} else if (CB_LIST_P (x)) {
				/* X ALSO Y ... */
				if (!count) {
					ap->low_val_char = get_value (CB_VALUE (x
));
				}
				for (ls = x; ls; ls = CB_CHAIN (ls)) {
					n = get_value (CB_VALUE (ls));
```

```
                        if (!CB_CHAIN (ls)) {
                                lastval = n;
                        }
                        if (n < 0 || n > 255) {
                                unvals = 1;
                                continue;
                        }
                        if (values[n] != -1) {
                                dupls = 1;
                        }
                        values[n] = n;
                        count++;
                }
        } else {
                /* literal */
                if (CB_TREE_CLASS (x) == CB_CLASS_NUMERIC) {
                        n = get_value (x);
                        lastval = n;
                        if (!count) {
                                ap->low_val_char = n;
                        }
                        if (n < 0 || n > 255) {
                                unvals = 1;
                                continue;
                        }
                        if (values[n] != -1) {
                                dupls = 1;
                        }
                        values[n] = n;
                        count++;
                } else if (CB_LITERAL_P (x)) {
                        size = (int)CB_LITERAL (x)->size;
                        data = CB_LITERAL (x)->data;
                        if (!count) {
                                ap->low_val_char = data[0];
                        }
                        lastval = data[size - 1];
                        for (i = 0; i < size; i++) {
                                n = data[i];
                                if (values[n] != -1) {
                                        dupls = 1;
                                }
                                values[n] = n;
                                count++;
                        }
                } else {
                        n = get_value (x);
                        lastval = n;
                        if (!count) {
                                ap->low_val_char = n;
                        }
                        if (n < 0 || n > 255) {
                                unvals = 1;
                                continue;
                        }
                        if (values[n] != -1) {
                                dupls = 1;
                        }
                        values[n] = n;
                        count++;
                }
        }
```

```
                }
                if (dupls || unvals) {
                        if (dupls) {
                                cb_error_x (l, _("Duplicate character values in a
      lphabet '%s'"),
                                            cb_name (CB_VALUE(l)));
                        }
                        if (unvals) {
                                cb_error_x (l, _("Invalid character values in alp
      habet '%s'"),
                                            cb_name (CB_VALUE(l)));
                        }
                        ap->low_val_char = 0;
                        ap->high_val_char = 255;
                        continue;
                }
                /* Calculate HIGH-VALUE */
                /* If all 256 values have been specified, HIGH-VALUE is the last
      one */
                /* Otherwise if HIGH-VALUE has been specified, find the highest *
      /
                /* value that has not been used */
                if (count == 256) {
                        ap->high_val_char = lastval;
                } else if (values[255] != -1) {
                        for (n = 254; n >= 0; n--) {
                                if (values[n] == -1) {
                                        ap->high_val_char = n;
                                        break;
                                }
                        }
                }
          }
          /* Rest HIGH/LOW-VALUES */
          cb_low = cb_norm_low;
          cb_high = cb_norm_high;
          /* resolve the program collating sequence */
          if (!prog->collating_sequence) {
                  return;
          }
          x = cb_ref (prog->collating_sequence);
/* RXWRXW
          if (x == cb_error_node) {
                  prog->collating_sequence = NULL;
                  return;
          }
*/
          if (!CB_ALPHABET_NAME_P (x)) {
                  cb_error_x (prog->collating_sequence, _("'%s' not alphabet name")
      ,
                              cb_name (prog->collating_sequence));
                  prog->collating_sequence = NULL;
                  return;
          }
          if (CB_ALPHABET_NAME (x)->type != CB_ALPHABET_CUSTOM) {
                  return;
          }
          if (CB_ALPHABET_NAME (x)->low_val_char) {
                  cb_low = cb_build_alphanumeric_literal ((ucharptr)"\0", 1);
                  CB_LITERAL(cb_low)->data[0] = CB_ALPHABET_NAME (x)->low_val_char;

                  CB_LITERAL(cb_low)->all = 1;
```

```
        }
        if (CB_ALPHABET_NAME (x)->high_val_char != 255){
                cb_high = cb_build_alphanumeric_literal ((ucharptr)"\0", 1);
                CB_LITERAL(cb_high)->data[0] = CB_ALPHABET_NAME (x)->high_val_cha
      r;
                CB_LITERAL(cb_high)->all = 1;
        }
}
```

7.17.2.115 int validate_move (cb_tree *src*, cb_tree *dst*, size_t *is_value*)

Definition at line 3944 of file typeck.c.

```
{
        struct cb_field         *f;
        struct cb_literal       *l;
        unsigned char           *p;
        cb_tree                 loc;
        long long               val;
        size_t                  i;
        size_t                  is_numeric_edited = 0;
        int                     src_scale_mod;
        int                     dst_scale_mod;
        int                     dst_size_mod;
        int                     size;
        int                     most_significant;
        int                     least_significant;

        loc = src->source_line ? src : dst;
        if (CB_REFERENCE_P(dst) && CB_ALPHABET_NAME_P(CB_REFERENCE(dst)->value))
      {
                goto invalid;
        }
        if (CB_REFERENCE_P(dst) && CB_FILE_P(CB_REFERENCE(dst)->value)) {
                goto invalid;
        }
        if (CB_TREE_CATEGORY (dst) == CB_CATEGORY_BOOLEAN) {
                cb_error_x (loc, _("Invalid destination for MOVE"));
                return -1;
        }

        if (CB_TREE_CLASS (dst) == CB_CLASS_POINTER) {
                if (CB_TREE_CLASS (src) == CB_CLASS_POINTER) {
                        return 0;
                } else {
                        goto invalid;
                }
        }

        f = cb_field (dst);
        switch (CB_TREE_TAG (src)) {
        case CB_TAG_CONST:
                if (src == cb_space) {
                        if (CB_TREE_CATEGORY (dst) == CB_CATEGORY_NUMERIC
                            || (CB_TREE_CATEGORY (dst) ==
      CB_CATEGORY_NUMERIC_EDITED && !is_value)) {
                                goto invalid;
                        }
                } else if (src == cb_zero) {
```

```
                        if (CB_TREE_CATEGORY (dst) == CB_CATEGORY_ALPHABETIC) {
                                goto invalid;
                        }
                }
                break;
        case CB_TAG_LITERAL:
                /* TODO: ALL literal */

                l = CB_LITERAL (src);
                if (CB_TREE_CLASS (src) == CB_CLASS_NUMERIC) {
                        /* Numeric literal */
                        if (l->all) {
                                goto invalid;
                        }
                        most_significant = -999;
                        least_significant = 999;

                        /* compute the most significant figure place */
                        for (i = 0; i < l->size; i++) {
                                if (l->data[i] != '0') {
                                        break;
                                }
                        }
                        if (i != l->size) {
                                most_significant = (int) (l->size - l->scale - i
- 1);
                        }

                        /* compute the least significant figure place */
                        for (i = 0; i < l->size; i++) {
                                if (l->data[l->size - i - 1] != '0') {
                                        break;
                                }
                        }
                        if (i != l->size) {
                                least_significant = (int) (-l->scale + i);
                        }

                        /* value check */
                        switch (CB_TREE_CATEGORY (dst)) {
                        case CB_CATEGORY_ALPHANUMERIC:
                        case CB_CATEGORY_ALPHANUMERIC_EDITED:
                                if (is_value) {
                                        goto expect_alphanumeric;
                                }

                                if (l->scale == 0) {
                                        goto expect_alphanumeric;
                                } else {
                                        goto invalid;
                                }
                        case CB_CATEGORY_NUMERIC:
                                if (f->pic->scale < 0) {
                                        /* check for PIC 9(n)P(m) */
                                        if (least_significant < -f->pic->scale) {

                                                goto value_mismatch;
                                        }
                                } else if (f->pic->scale > f->pic->size) {
                                        /* check for PIC P(n)9(m) */
                                        if (most_significant >= f->pic->size - f-
>pic->scale) {
```

```
                                        goto value_mismatch;
                                }
                        }
                        break;
                case CB_CATEGORY_NUMERIC_EDITED:
                        if (is_value) {
                                goto expect_alphanumeric;
                        }

                        /* TODO */
                        break;
                default:
                        if (is_value) {
                                goto expect_alphanumeric;
                        }
                        goto invalid;
                }

                /* sign check */
                if (l->sign != 0 && !f->pic->have_sign) {
                        if (is_value) {
                                cb_error_x (loc, _("Data item not signed"
));
                                return -1;
                        }
                        if (cb_warn_constant) {
                                cb_warning_x (loc, _("Ignoring negative s
ign"));
                        }
                }

                /* size check */
                if (f->flag_real_binary ||
                    ((f->usage == CB_USAGE_COMP_5 ||
                      f->usage == CB_USAGE_COMP_X ||
                      f->usage == CB_USAGE_BINARY) &&
                      f->pic->scale == 0)) {
                        p = l->data;
                        for (i = 0; i < l->size; i++) {
                                if (l->data[i] != '0') {
                                        p = &l->data[i];
                                        break;
                                }
                        }
                        i = l->size - i;
                        switch (f->size) {
                        case 1:
                                if (i > 18) {
                                        goto numlit_overflow;
                                }
                                val = cb_get_long_long (src);
                                if (f->pic->have_sign) {
                                        if (val < -128LL ||
                                            val > 127LL) {
                                                goto numlit_overflow;
                                        }
                                } else {
                                        if (val > 255LL) {
                                                goto numlit_overflow;
                                        }
                                }
                                break;
```

```
case 2:
        if (i > 18) {
                goto numlit_overflow;
        }
        val = cb_get_long_long (src);
        if (f->pic->have_sign) {
                if (val < -32768LL ||
                    val > 32767LL) {
                        goto numlit_overflow;
                }
        } else {
                if (val > 65535LL) {
                        goto numlit_overflow;
                }
        }
        break;
case 3:
        if (i > 18) {
                goto numlit_overflow;
        }
        val = cb_get_long_long (src);
        if (f->pic->have_sign) {
                if (val < -8388608LL ||
                    val > 8388607LL) {
                        goto numlit_overflow;
                }
        } else {
                if (val > 16777215LL) {
                        goto numlit_overflow;
                }
        }
        break;
case 4:
        if (i > 18) {
                goto numlit_overflow;
        }
        val = cb_get_long_long (src);
        if (f->pic->have_sign) {
                if (val < -2147483648LL ||
                    val > 2147483647LL) {
                        goto numlit_overflow;
                }
        } else {
                if (val > 4294967295LL) {
                        goto numlit_overflow;
                }
        }
        break;
case 5:
        if (i > 18) {
                goto numlit_overflow;
        }
        val = cb_get_long_long (src);
        if (f->pic->have_sign) {
                if (val < -549755813888LL ||
                    val > 549755813887LL) {
                        goto numlit_overflow;
                }
        } else {
                if (val > 1099511627775LL) {
                        goto numlit_overflow;
                }
```

```
                }
                break;
        case 6:
                if (i > 18) {
                        goto numlit_overflow;
                }
                val = cb_get_long_long (src);
                if (f->pic->have_sign) {
                        if (val < -140737488355328LL ||
                            val > 140737488355327LL) {
                                goto numlit_overflow;
                        }
                } else {
                        if (val > 281474976710655LL) {
                                goto numlit_overflow;
                        }
                }
                break;
        case 7:
                if (i > 18) {
                        goto numlit_overflow;
                }
                val = cb_get_long_long (src);
                if (f->pic->have_sign) {
                        if (val < -36028797018963968LL ||

                            val > 36028797018963967LL) {
                                goto numlit_overflow;
                        }
                } else {
                        if (val > 72057594037927935LL) {
                                goto numlit_overflow;
                        }
                }
                break;
        default:
                if (f->pic->have_sign) {
                        if (i < 19) {
                                break;
                        }
                        if (i > 19) {
                                goto numlit_overflow;
                        }
                        if (memcmp (p, "92233720368547758
07", 19) > 0) {
                                goto numlit_overflow;
                        }
                } else {
                        if (i < 20) {
                                break;
                        }
                        if (i > 20) {
                                goto numlit_overflow;
                        }
                        if (memcmp (p, "18446744073709551
615", 20) > 0) {
                                goto numlit_overflow;
                        }
                }
                break;
        }
        return 0;
```

```
                        }
                        if (least_significant < -f->pic->scale) {
                                goto size_overflow;
                        }
                        if (f->pic->scale > 0) {
                                size = f->pic->digits - f->pic->scale;
                        } else {
                                size = f->pic->digits;
                        }
                        if (most_significant >= size) {
                                goto size_overflow;
                        }
                } else {
                        /* Alphanumeric literal */

                        /* value check */
                        switch (CB_TREE_CATEGORY (dst)) {
                        case CB_CATEGORY_ALPHABETIC:
                                for (i = 0; i < l->size; i++) {
                                        if (!isalpha (l->data[i]) && !isspace (l-
>data[i])) {
                                                goto value_mismatch;
                                        }
                                }
                                break;
                        case CB_CATEGORY_NUMERIC:
                                goto expect_numeric;
                        case CB_CATEGORY_NUMERIC_EDITED:
                                if (!is_value) {
                                        goto expect_numeric;
                                }

                                /* TODO: validate the value */
                                break;
                        default:
                                break;
                        }

                        /* size check */
                        size = cb_field_size (dst);
                        if (size >= 0 && (int)l->size > size) {
                                goto size_overflow;
                        }
                }
                break;
        case CB_TAG_FIELD:
        case CB_TAG_REFERENCE:
                if (CB_REFERENCE_P(src) &&
                    CB_ALPHABET_NAME_P(CB_REFERENCE(src)->value)) {
                        break;
                }
                if (CB_REFERENCE_P(src) &&
                    CB_FILE_P(CB_REFERENCE(src)->value)) {
                        goto invalid;
                }
                size = cb_field_size (src);
                if (size < 0) {
                        size = cb_field (src)->size;
                }
                /* non-elementary move */
                if (cb_field (src)->children || cb_field (dst)->children) {
                        if (size > cb_field (dst)->size) {
```

```
                        goto size_overflow_1;
                }
                break;
        }

        /* elementary move */
        switch (CB_TREE_CATEGORY (src)) {
        case CB_CATEGORY_ALPHANUMERIC:
                switch (CB_TREE_CATEGORY (dst)) {
                case CB_CATEGORY_NUMERIC:
                case CB_CATEGORY_NUMERIC_EDITED:
                        if (size > cb_field (dst)->pic->digits) {
                                goto size_overflow_2;
                        }
                        break;
                case CB_CATEGORY_ALPHANUMERIC_EDITED:
                        if (size >
                            count_pic_alphanumeric_edited (cb_field (dst)
)) {
                                goto size_overflow_1;
                        }
                        break;
                default:
                        if (size > cb_field (dst)->size) {
                                goto size_overflow_1;
                        }
                        break;
                }
                break;
        case CB_CATEGORY_ALPHABETIC:
        case CB_CATEGORY_ALPHANUMERIC_EDITED:
                switch (CB_TREE_CATEGORY (dst)) {
                case CB_CATEGORY_NUMERIC:
                case CB_CATEGORY_NUMERIC_EDITED:
                        goto invalid;
                case CB_CATEGORY_ALPHANUMERIC_EDITED:
                        if (size >
                            count_pic_alphanumeric_edited(cb_field (dst))
) {
                                goto size_overflow_1;
                        }
                        break;
                default:
                        if (size > cb_field (dst)->size) {
                                goto size_overflow_1;
                        }
                        break;
                }
                break;
        case CB_CATEGORY_NUMERIC:
        case CB_CATEGORY_NUMERIC_EDITED:
                switch (CB_TREE_CATEGORY (dst)) {
                case CB_CATEGORY_ALPHABETIC:
                        goto invalid;
                case CB_CATEGORY_ALPHANUMERIC_EDITED:
                        is_numeric_edited = 1;
                        /* Drop through */
                case CB_CATEGORY_ALPHANUMERIC:
                        if (is_numeric_edited) {
                                dst_size_mod = count_pic_alphanumeric_edi
ted (cb_field (dst));
                        } else {
```

```
                                        dst_size_mod = cb_field (dst)->size;
                                }
                                if (CB_TREE_CATEGORY (src) ==
      CB_CATEGORY_NUMERIC
                                    && cb_field (src)->pic->scale > 0) {
                                        if (cb_move_noninteger_to_alphanumeric ==
       CB_ERROR) {
                                                goto invalid;
                                        }
                                        cb_warning_x (loc, _("Move non-integer to
       alphanumeric"));
                                        break;
                                }
                                if (CB_TREE_CATEGORY (src) ==
      CB_CATEGORY_NUMERIC
                                    && cb_field (src)->pic->digits > dst_size_mod
      ) {
                                        goto size_overflow_2;
                                }
                                if (CB_TREE_CATEGORY (src) ==
      CB_CATEGORY_NUMERIC_EDITED
                                    && cb_field (src)->size > dst_size_mod) {
                                        goto size_overflow_1;
                                }
                                break;
                        default:
                                src_scale_mod = cb_field (src)->pic->scale < 0 ?
                                                0 : cb_field (src)->pic->scale;
                                dst_scale_mod = cb_field (dst)->pic->scale < 0 ?
                                                0 : cb_field (dst)->pic->scale;
                                if (cb_field (src)->pic->digits - src_scale_mod >

                                    cb_field (dst)->pic->digits - dst_scale_mod
      ||
                                    src_scale_mod > dst_scale_mod) {
                                        goto size_overflow_2;
                                }
                                break;
                        }
                        break;
                default:
                        cb_error_x (loc, _("Invalid source for MOVE"));
                        return -1;
                }
                break;
          case CB_TAG_INTEGER:
          case CB_TAG_BINARY_OP:
          case CB_TAG_INTRINSIC:
                /* TODO: check this */
                break;
          default:
                fprintf (stderr, "Invalid tree tag %d\n", CB_TREE_TAG (src));
                ABORT ();
          }
          return 0;

invalid:
          if (is_value) {
                cb_error_x (loc, _("Invalid VALUE clause"));
          } else {
                cb_error_x (loc, _("Invalid MOVE statement"));
          }
```

```
        return -1;

numlit_overflow:
        if (is_value) {
                cb_error_x (loc, _("Invalid VALUE clause - literal exceeds data s
      ize"));
                return -1;
        }
        if (cb_warn_constant) {
                cb_warning_x (loc, _("Numeric literal exceeds data size"));
        }
        return 0;

expect_numeric:
        return move_error (src, dst, is_value, cb_warn_strict_typing, 0,
                           _("Numeric value is expected"));

expect_alphanumeric:
        return move_error (src, dst, is_value, cb_warn_strict_typing, 0,
                           _("Alphanumeric value is expected"));

value_mismatch:
        return move_error (src, dst, is_value, cb_warn_constant, 0,
                           _("Value does not fit the picture string"));

size_overflow:
        return move_error (src, dst, is_value, cb_warn_constant, 0,
                           _("Value size exceeds data size"));

size_overflow_1:
        return move_error (src, dst, is_value, cb_warn_truncate, 1,
                           _("Sending field larger than receiving field"));

size_overflow_2:
        return move_error (src, dst, is_value, cb_warn_truncate, 1,
                           _("Some digits may be truncated"));
}
```

7.17.3 Variable Documentation

7.17.3.1 size_t sending_id = 0

Definition at line 74 of file typeck.c.

7.17.3.2 size_t suppress_warn = 0

Definition at line 75 of file typeck.c.

7.18 config.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define COB_EXEEXT ""
- #define COB_EXPORT_DYN "-Wl,--export-dynamic"
- #define COB_EXTRA_FLAGS ""
- #define COB_HAS_INLINE 1
- #define COB_LI_IS_LL 1
- #define COB_PIC_FLAGS "-fPIC -DPIC"
- #define COB_SHARED_OPT "-shared"
- #define COB_STRFTIME 1
- #define COB_STRIP_CMD "strip --strip-unneeded"
- #define ENABLE_NLS 1
- #define HAVE_ALLOCA 1
- #define HAVE_ALLOCA_H 1
- #define HAVE_ATTRIBUTE_ALIGNED 1
- #define HAVE_COLOR_SET 1
- #define HAVE_DCGETTEXT 1
- #define HAVE_DLFCN_H 1
- #define HAVE_FCNTL 1
- #define HAVE_FCNTL_H 1
- #define HAVE_GETOPT_H 1
- #define HAVE_GETTEXT 1
- #define HAVE_GETTIMEOFDAY 1
- #define HAVE_GMP_H 1
- #define HAVE_ICONV 1
- #define HAVE_INTTYPES_H 1
- #define HAVE_LANGINFO_CODESET 1
- #define HAVE_LIBGMP 1
- #define HAVE_LIBNCURSES 1
- #define HAVE_LOCALE_H 1
- #define HAVE_MALLOC_H 1
- #define HAVE_MEMMOVE 1
- #define HAVE_MEMORY_H 1
- #define HAVE_MEMSET 1
- #define HAVE_NCURSES_H 1
- #define HAVE_PSIGN_OPT 1
- #define HAVE_SETLOCALE 1

- #define HAVE_SIGNAL_H 1
- #define HAVE_STDDEF_H 1
- #define HAVE_STDINT_H 1
- #define HAVE_STDLIB_H 1
- #define HAVE_STRCASECMP 1
- #define HAVE_STRCHR 1
- #define HAVE_STRDUP 1
- #define HAVE_STRERROR 1
- #define HAVE_STRINGS_H 1
- #define HAVE_STRING_H 1
- #define HAVE_STRRCHR 1
- #define HAVE_STRSTR 1
- #define HAVE_STRTOL 1
- #define HAVE_SYS_STAT_H 1
- #define HAVE_SYS_TIME_H 1
- #define HAVE_SYS_TYPES_H 1
- #define HAVE_TIMEZONE 1
- #define HAVE_UNISTD_H 1
- #define HAVE_VPRINTF 1
- #define HAVE_WCHAR_H 1
- #define ICONV_CONST
- #define PACKAGE "open-cobol"
- #define PACKAGE_BUGREPORT "open-cobol-list@lists.sourceforge.net"
- #define PACKAGE_NAME "OpenCOBOL"
- #define PACKAGE_STRING "OpenCOBOL 1.1"
- #define PACKAGE_TARNAME "open-cobol"
- #define PACKAGE_VERSION "1.1"
- #define PATCH_LEVEL 0
- #define STDC_HEADERS 1
- #define USE_DB41 1
- #define USE_LIBDL 1
- #define VERSION "1.1"
- #define WITH_DB 3
- #define WITH_VARSEQ 0
- #define __USE_STRING_INLINES 1

7.18.1 Define Documentation

7.18.1.1 #define __USE_STRING_INLINES 1

Definition at line 348 of file config.h.

7.18.1.2 #define COB_EXEEXT ""

Definition at line 5 of file config.h.

7.18.1.3 #define COB_EXPORT_DYN "-Wl,--export-dynamic"

Definition at line 11 of file config.h.

7.18.1.4 #define COB_EXTRA_FLAGS ""

Definition at line 14 of file config.h.

7.18.1.5 #define COB_HAS_INLINE 1

Definition at line 17 of file config.h.

7.18.1.6 #define COB_LI_IS_LL 1

Definition at line 20 of file config.h.

7.18.1.7 #define COB_PIC_FLAGS "-fPIC -DPIC"

Definition at line 29 of file config.h.

7.18.1.8 #define COB_SHARED_OPT "-shared"

Definition at line 32 of file config.h.

7.18.1.9 #define COB_STRFTIME 1

Definition at line 35 of file config.h.

7.18.1.10 #define COB_STRIP_CMD "strip --strip-unneeded"

Definition at line 38 of file config.h.

7.18.1.11 #define ENABLE_NLS 1

Definition at line 50 of file config.h.

7.18.1.12 #define HAVE_ALLOCA 1

Definition at line 53 of file config.h.

7.18.1.13 #define HAVE_ALLOCA_H 1

Definition at line 57 of file config.h.

7.18.1.14 #define HAVE_ATTRIBUTE_ALIGNED 1

Definition at line 60 of file config.h.

7.18.1.15 #define HAVE_COLOR_SET 1

Definition at line 63 of file config.h.

7.18.1.16 #define HAVE_DCGETTEXT 1

Definition at line 100 of file config.h.

7.18.1.17 #define HAVE_DLFCN_H 1

Definition at line 106 of file config.h.

7.18.1.18 #define HAVE_FCNTL 1

Definition at line 112 of file config.h.

7.18.1.19 #define HAVE_FCNTL_H 1

Definition at line 115 of file config.h.

7.18.1.20 #define HAVE_GETOPT_H 1

Definition at line 118 of file config.h.

7.18.1.21 #define HAVE_GETTEXT 1

Definition at line 121 of file config.h.

7.18.1.22 #define HAVE_GETTIMEOFDAY 1

Definition at line 124 of file config.h.

7.18.1.23 #define HAVE_GMP_H 1

Definition at line 127 of file config.h.

7.18.1.24 #define HAVE_ICONV 1

Definition at line 130 of file config.h.

7.18.1.25 #define HAVE_INTTYPES_H 1

Definition at line 133 of file config.h.

7.18.1.26 #define HAVE_LANGINFO_CODESET 1

Definition at line 142 of file config.h.

7.18.1.27 #define HAVE_LIBGMP 1

Definition at line 151 of file config.h.

7.18.1.28 #define HAVE_LIBNCURSES 1

Definition at line 160 of file config.h.

7.18.1.29 #define HAVE_LOCALE_H 1

Definition at line 175 of file config.h.

7.18.1.30 #define HAVE_MALLOC_H 1

Definition at line 181 of file config.h.

7.18.1.31 #define HAVE_MEMMOVE 1

Definition at line 184 of file config.h.

7.18.1.32 #define HAVE_MEMORY_H 1

Definition at line 187 of file config.h.

7.18.1.33 #define HAVE_MEMSET 1

Definition at line 190 of file config.h.

7.18.1.34 #define HAVE_NCURSES_H 1

Definition at line 193 of file config.h.

7.18.1.35 #define HAVE_PSIGN_OPT 1

Definition at line 202 of file config.h.

7.18.1.36 #define HAVE_SETLOCALE 1

Definition at line 205 of file config.h.

7.18.1.37 #define HAVE_SIGNAL_H 1

Definition at line 208 of file config.h.

7.18.1.38 #define HAVE_STDDEF_H 1

Definition at line 211 of file config.h.

7.18.1.39 #define HAVE_STDINT_H 1

Definition at line 214 of file config.h.

7.18.1.40 #define HAVE_STDLIB_H 1

Definition at line 217 of file config.h.

7.18.1.41 #define HAVE_STRCASECMP 1

Definition at line 220 of file config.h.

7.18.1.42 #define HAVE_STRCHR 1

Definition at line 223 of file config.h.

7.18.1.43 #define HAVE_STRDUP 1

Definition at line 226 of file config.h.

7.18.1.44 #define HAVE_STRERROR 1

Definition at line 229 of file config.h.

7.18.1.45 #define HAVE_STRING_H 1

Definition at line 235 of file config.h.

7.18.1.46 #define HAVE_STRINGS_H 1

Definition at line 232 of file config.h.

7.18.1.47 #define HAVE_STRRCHR 1

Definition at line 238 of file config.h.

7.18.1.48 #define HAVE_STRSTR 1

Definition at line 241 of file config.h.

7.18.1.49 #define HAVE_STRTOL 1

Definition at line 244 of file config.h.

7.18.1.50 #define HAVE_SYS_STAT_H 1

Definition at line 247 of file config.h.

7.18.1.51 #define HAVE_SYS_TIME_H 1

Definition at line 250 of file config.h.

7.18.1.52 #define HAVE_SYS_TYPES_H 1

Definition at line 253 of file config.h.

7.18.1.53 #define HAVE_TIMEZONE 1

Definition at line 256 of file config.h.

7.18.1.54 #define HAVE_UNISTD_H 1

Definition at line 259 of file config.h.

7.18.1.55 #define HAVE_VPRINTF 1

Definition at line 265 of file config.h.

7.18.1.56 #define HAVE_WCHAR_H 1

Definition at line 268 of file config.h.

7.18.1.57 #define ICONV_CONST

Definition at line 271 of file config.h.

7.18.1.58 #define PACKAGE "open-cobol"

Definition at line 274 of file config.h.

7.18.1.59 #define PACKAGE_BUGREPORT "open-cobol-list@lists.sourceforge.net"

Definition at line 277 of file config.h.

7.18.1.60 #define PACKAGE_NAME "OpenCOBOL"

Definition at line 280 of file config.h.

7.18.1.61 #define PACKAGE_STRING "OpenCOBOL 1.1"

Definition at line 283 of file config.h.

7.18.1.62 #define PACKAGE_TARNAME "open-cobol"

Definition at line 286 of file config.h.

7.18.1.63 #define PACKAGE_VERSION "1.1"

Definition at line 289 of file config.h.

7.18.1.64 #define PATCH_LEVEL 0

Definition at line 292 of file config.h.

7.18.1.65 #define STDC_HEADERS 1

Definition at line 303 of file config.h.

7.18.1.66 #define USE_DB41 1

Definition at line 309 of file config.h.

7.18.1.67 #define USE_LIBDL 1

Definition at line 312 of file config.h.

7.18.1.68 #define VERSION "1.1"

Definition at line 315 of file config.h.

7.18.1.69 #define WITH_DB 3

Definition at line 321 of file config.h.

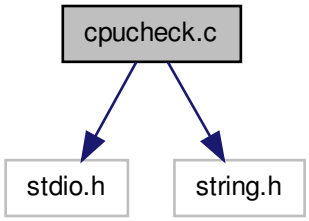
7.18.1.70 #define WITH_VARSEQ 0

Definition at line 333 of file config.h.

7.19 cpucheck.c File Reference

```
#include <stdio.h>
#include <string.h>
```

Include dependency graph for cpucheck.c:



Functions

- int main ()

7.19.1 Function Documentation

7.19.1.1 int main ( )

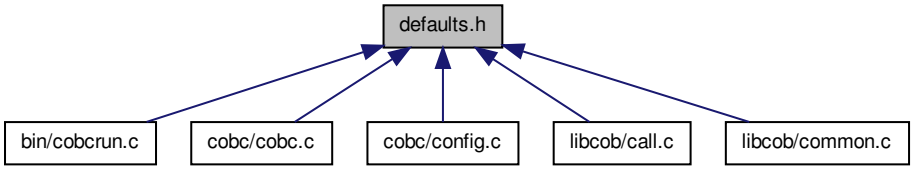
Definition at line 5 of file cpucheck.c.

```
{
#ifdef __i386__
#if ( __GNUC__ == 3 && __GNUC_MINOR__ > 0 ) || __GNUC__ > 3
#if __GNUC__ > 3 || __GNUC_MINOR__ > 3
        char *ctune = "-mtune=";
#else
        char *ctune = "-mcpu=";
#endif
        char vendor_string[16];
        int eax, ebx, edx, ecx;
        int i, hv;
        int family, model, stepping;

        __asm__ (".byte 0x0f,0xa2"
        : "=a" (hv), "=b" (ebx), "=d" (edx), "=c" (ecx) : "0" (0));

        *(int *) (vendor_string + 0) = ebx;
        *(int *) (vendor_string + 4) = edx;
        *(int *) (vendor_string + 8) = ecx;
        vendor_string[12] = 0;
        i = 1;

        __asm__ (".byte 0x0f,0xa2"
        : "=a" (eax), "=b" (ebx), "=d" (edx), "=c" (ecx) : "0" (i));

        family = (eax >> 8) & 0xf;
```

```
        model = (eax >> 4) & 0xf;
        stepping = eax & 0xf;
        if (family == 0xf) {
                /* "extended" mode. */
                family += (eax >> 20) & 0xff;
                model += (eax >> 12) & 0xf0;
        }

        if (strcmp (vendor_string, "GenuineIntel") == 0) {
                if (family == 5) {
                        printf ("-march=i686 %spentium", ctune);
                } else if (family == 6) {
                        if (model <= 2) {
                                printf ("-march=i686 %spentiumpro",ctune);
                        } else if (model >= 3 && model <= 6) {
                                printf ("-march=i686 %spentium2",ctune);
                        } else if (model <= 11) {
                                printf ("-march=i686 %spentium3", ctune);
                        } else {
                                printf ("-march=i686 %spentium4", ctune);
                        }
                }
                else if (family == 15) {
                        printf ("-march=i686 %spentium4", ctune);
                }
        } else if (strcmp (vendor_string, "AuthenticAMD") == 0) {
                if (family == 6) {
                        printf ("-march=i686 %sathlon", ctune);
                }
        }
#endif
#endif
        return 0;
}
```

7.20 defaults.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define COB_CC "gcc"

- #define COB_CFLAGS "-I/usr/local/include "
- #define COB_LDFLAGS ""
- #define COB_LIBS "-L/usr/local/lib -lcob -lm -lgmp -lncurses -ldb -ldl"
- #define COB_CONFIG_DIR "/usr/local/share/open-cobol/config"
- #define COB_COPY_DIR "/usr/local/share/open-cobol/copy"
- #define COB_LIBRARY_PATH "/usr/local/lib/open-cobol"
- #define COB_MODULE_EXT "so"
- #define LOCALEDIR "/usr/local/share/locale"

7.20.1 Define Documentation

7.20.1.1 #define COB_CC "gcc"

Definition at line 2 of file defaults.h.

7.20.1.2 #define COB_CFLAGS "-I/usr/local/include "

Definition at line 3 of file defaults.h.

7.20.1.3 #define COB_CONFIG_DIR "/usr/local/share/open-cobol/config"

Definition at line 6 of file defaults.h.

7.20.1.4 #define COB_COPY_DIR "/usr/local/share/open-cobol/copy"

Definition at line 7 of file defaults.h.

7.20.1.5 #define COB_LDFLAGS ""

Definition at line 4 of file defaults.h.

7.20.1.6 #define COB_LIBRARY_PATH "/usr/local/lib/open-cobol"

Definition at line 8 of file defaults.h.

7.20.1.7 #define COB_LIBS "-L/usr/local/lib -lcob -lm -lgmp -lncurses -ldb -ldl"

Definition at line 5 of file defaults.h.

7.20.1.8 #define COB_MODULE_EXT "so"

Definition at line 9 of file defaults.h.

7.20.1.9 #define LOCALEDIR "/usr/local/share/locale"

Definition at line 10 of file defaults.h.

7.21 lib/dummymac.c File Reference

Functions

- void dummymacfix (void)

7.21.1 Function Documentation

7.21.1.1 void dummymacfix (void)

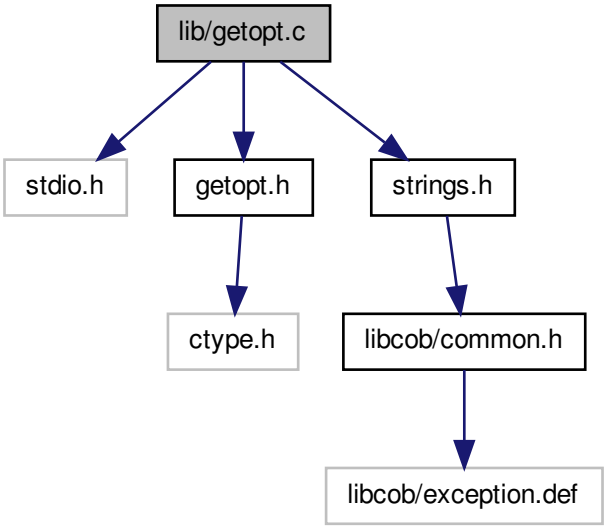
Definition at line 3 of file dummymac.c.

```
{
}
```

7.22 lib/getopt.c File Reference

```
#include <stdio.h>
#include "getopt.h"
#include <strings.h>
```

Include dependency graph for getopt.c:



Defines

- #define _NO_PROTO
- #define const
- #define GETOPT_INTERFACE_VERSION 2
- #define _(msgid) (msgid)
- #define attribute_hidden
- #define SWAP_FLAGS(ch1, ch2)
- #define NONOPTION_P (argv[optind][0] != '-' || argv[optind][1] == '\0')

Enumerations

- enum { REQUIRE_ORDER, PERMUTE, RETURN_IN_ORDER }

Functions

- char * getenv ()
- int _getopt_internal (int argc, char *const *argv, const char *optstring, const struct option *longopts, int *longind, int long_only)
- int getopt (int argc, char *const *argv, const char *optstring)

Variables

- char * optarg
- int optind = 1
- int __getopt_initialized attribute_hidden
- int opterr = 1
- int optopt = '?'

7.22.1 Define Documentation

7.22.1.1 #define _(*msgid*) (msgid)

Definition at line 89 of file getopt.c.

7.22.1.2 #define _NO_PROTO

Definition at line 27 of file getopt.c.

7.22.1.3 #define attribute_hidden

Definition at line 97 of file getopt.c.

7.22.1.4 #define const

Definition at line 38 of file getopt.c.

7.22.1.5 #define GETOPT_INTERFACE_VERSION 2

Definition at line 52 of file getopt.c.

7.22.1.6 #define NONOPTION_P (argv[optind][0] != '-' || argv[optind][1] == '\0')

7.22.1.7 #define SWAP_FLAGS(*ch1*, *ch2*)

Definition at line 291 of file getopt.c.

7.22.2 Enumeration Type Documentation

7.22.2.1 anonymous enum

Enumerator:

REQUIRE_ORDER

PERMUTE

RETURN_IN_ORDER

Definition at line 194 of file getopt.c.

```
{
  REQUIRE_ORDER, PERMUTE, RETURN_IN_ORDER
} ordering;
```

7.22.3 Function Documentation

7.22.3.1 int _getopt_internal (int *argc*, char ∗const ∗ *argv*, const char ∗ *optstring*, const struct option ∗ *longopts*, int ∗ *longind*, int *long_only*)

Definition at line 515 of file getopt.c.

```
{
  int print_errors = opterr;
  if (optstring[0] == ':')
    print_errors = 0;

  if (argc < 1)
    return -1;

  optarg = NULL;

  if (optind == 0 || !__getopt_initialized)
    {
      if (optind == 0)
        optind = 1;     /* Don't scan ARGV[0], the program name.  */
      optstring = _getopt_initialize (argc, argv, optstring);
      __getopt_initialized = 1;
    }

  /* Test whether ARGV[optind] points to a non-option argument.
     Either it does not have option syntax, or there is an environment flag
     from the shell indicating it is not an option.  The later information
     is only used when the used in the GNU libc.  */
#if defined _LIBC && defined USE_NONOPTION_FLAGS
# define NONOPTION_P (argv[optind][0] != '-' || argv[optind][1] == '\0'       \
                      || (optind < nonoption_flags_len                        \
                          && __getopt_nonoption_flags[optind] == '1'))
#else
# define NONOPTION_P (argv[optind][0] != '-' || argv[optind][1] == '\0')
#endif

  if (nextchar == NULL || *nextchar == '\0')
    {
      /* Advance to the next ARGV-element.  */

      /* Give FIRST_NONOPT & LAST_NONOPT rational values if OPTIND has been
         moved back by the user (who may also have changed the arguments).  */
      if (last_nonopt > optind)
        last_nonopt = optind;
      if (first_nonopt > optind)
        first_nonopt = optind;

      if (ordering == PERMUTE)
        {
```

```
      /* If we have just processed some options following some non-options,
         exchange them so that the options come first.  */

      if (first_nonopt != last_nonopt && last_nonopt != optind)
        exchange ((char **) argv);
      else if (last_nonopt != optind)
        first_nonopt = optind;

      /* Skip any additional non-options
         and extend the range of non-options previously skipped.  */

      while (optind < argc && NONOPTION_P)
        optind++;
      last_nonopt = optind;
    }

  /* The special ARGV-element `--' means premature end of options.
     Skip it like a null option,
     then exchange with previous non-options as if it were an option,
     then skip everything else like a non-option.  */

  if (optind != argc && !strcmp (argv[optind], "--"))
    {
      optind++;

      if (first_nonopt != last_nonopt && last_nonopt != optind)
        exchange ((char **) argv);
      else if (first_nonopt == last_nonopt)
        first_nonopt = optind;
      last_nonopt = argc;

      optind = argc;
    }

  /* If we have done all the ARGV-elements, stop the scan
     and back over any non-options that we skipped and permuted.  */

  if (optind == argc)
    {
      /* Set the next-arg-index to point at the non-options
         that we previously skipped, so the caller will digest them.  */
      if (first_nonopt != last_nonopt)
        optind = first_nonopt;
      return -1;
    }

  /* If we have come to a non-option and did not permute it,
     either stop the scan or describe it to the caller and pass it by.  */

  if (NONOPTION_P)
    {
      if (ordering == REQUIRE_ORDER)
        return -1;
      optarg = argv[optind++];
      return 1;
    }

  /* We have found another option-ARGV-element.
     Skip the initial punctuation.  */

  nextchar = (argv[optind] + 1
              + (longopts != NULL && argv[optind][1] == '-'));
```

```
    }

  /* Decode the current option-ARGV-element.  */

  /* Check whether the ARGV-element is a long option.

     If long_only and the ARGV-element has the form "-f", where f is
     a valid short option, don't consider it an abbreviated form of
     a long option that starts with f.  Otherwise there would be no
     way to give the -f short option.

     On the other hand, if there's a long option "fubar" and
     the ARGV-element is "-fu", do consider that an abbreviation of
     the long option, just like "--fu", and not "-f" with arg "u".

     This distinction seems to be the most useful approach.  */

  if (longopts != NULL
      && (argv[optind][1] == '-'
          || (long_only && (argv[optind][2] || !my_index (optstring, argv[optind]
      [1])))))
    {
      char *nameend;
      const struct option *p;
      const struct option *pfound = NULL;
      int exact = 0;
      int ambig = 0;
      int indfound = -1;
      int option_index;

      for (nameend = nextchar; *nameend && *nameend != '='; nameend++)
        /* Do nothing.  */ ;

      /* Test all long options for either exact match
         or abbreviated matches.  */
      for (p = longopts, option_index = 0; p->name; p++, option_index++)
        if (!strncmp (p->name, nextchar, nameend - nextchar))
          {
            if ((unsigned int) (nameend - nextchar)
                == (unsigned int) strlen (p->name))
              {
                /* Exact match found.  */
                pfound = p;
                indfound = option_index;
                exact = 1;
                break;
              }
            else if (pfound == NULL)
              {
                /* First nonexact match found.  */
                pfound = p;
                indfound = option_index;
              }
            else if (long_only
                     || pfound->has_arg != p->has_arg
                     || pfound->flag != p->flag
                     || pfound->val != p->val)
              /* Second or later nonexact match found.  */
              ambig = 1;
          }

      if (ambig && !exact)
```

```
        {
          if (print_errors)
            {
#if defined _LIBC && defined USE_IN_LIBIO
              char *buf;

              if (__asprintf (&buf, _("%s: option `%s' is ambiguous\n"),
                              argv[0], argv[optind]) >= 0)
                {

                  if (_IO_fwide (stderr, 0) > 0)
                    __fwprintf (stderr, L"%s", buf);
                  else
                    fputs (buf, stderr);

                  free (buf);
                }
#else
              fprintf (stderr, _("%s: option `%s' is ambiguous\n"),
                       argv[0], argv[optind]);
#endif
            }
          nextchar += strlen (nextchar);
          optind++;
          optopt = 0;
          return '?';
        }

      if (pfound != NULL)
        {
          option_index = indfound;
          optind++;
          if (*nameend)
            {
              /* Don't test has_arg with >, because some C compilers don't
                 allow it to be used on enums.  */
              if (pfound->has_arg)
                optarg = nameend + 1;
              else
                {
                  if (print_errors)
                    {
#if defined _LIBC && defined USE_IN_LIBIO
                      char *buf;
                      int n;
#endif

                      if (argv[optind - 1][1] == '-')
                        {
                          /* --option */
#if defined _LIBC && defined USE_IN_LIBIO
                          n = __asprintf (&buf, _("\
%s: option `--%s' doesn't allow an argument\n"),
                                                argv[0], pfound->name);
#else
                          fprintf (stderr, _("\
%s: option `--%s' doesn't allow an argument\n"),
                                   argv[0], pfound->name);
#endif
                        }
                      else
                        {
```

```
                        /* +option or -option */
#if defined _LIBC && defined USE_IN_LIBIO
                        n = __asprintf (&buf, _("\
%s: option `%c%s' doesn't allow an argument\n"),
                                        argv[0], argv[optind - 1][0],
                                        pfound->name);
#else
                        fprintf (stderr, _("\
%s: option `%c%s' doesn't allow an argument\n"),
                                 argv[0], argv[optind - 1][0], pfound->name);
#endif
                      }

#if defined _LIBC && defined USE_IN_LIBIO
                    if (n >= 0)
                      {
                        if (_IO_fwide (stderr, 0) > 0)
                          __fwprintf (stderr, L"%s", buf);
                        else
                          fputs (buf, stderr);

                        free (buf);
                      }
#endif
                  }

                nextchar += strlen (nextchar);

                optopt = pfound->val;
                return '?';
              }
            }
          else if (pfound->has_arg == 1)
            {
              if (optind < argc)
                optarg = argv[optind++];
              else
                {
                  if (print_errors)
                    {
#if defined _LIBC && defined USE_IN_LIBIO
                      char *buf;

                      if (__asprintf (&buf, _("\
%s: option `%s' requires an argument\n"),
                                      argv[0], argv[optind - 1]) >= 0)
                        {
                          if (_IO_fwide (stderr, 0) > 0)
                            __fwprintf (stderr, L"%s", buf);
                          else
                            fputs (buf, stderr);

                          free (buf);
                        }
#else
                      fprintf (stderr,
                               _("%s: option `%s' requires an argument\n"),
                               argv[0], argv[optind - 1]);
#endif
                    }
                  nextchar += strlen (nextchar);
                  optopt = pfound->val;
```

```
                  return optstring[0] == ':' ? ':' : '?';
                }
            }
          nextchar += strlen (nextchar);
          if (longind != NULL)
            *longind = option_index;
          if (pfound->flag)
            {
              *(pfound->flag) = pfound->val;
              return 0;
            }
          return pfound->val;
        }

      /* Can't find it as a long option.  If this is not getopt_long_only,
         or the option starts with '--' or is not a valid short
         option, then it's an error.
         Otherwise interpret it as a short option.  */
      if (!long_only || argv[optind][1] == '-'
          || my_index (optstring, *nextchar) == NULL)
        {
          if (print_errors)
            {
#if defined _LIBC && defined USE_IN_LIBIO
              char *buf;
              int n;
#endif

              if (argv[optind][1] == '-')
                {
                  /* --option */
#if defined _LIBC && defined USE_IN_LIBIO
                  n = __asprintf (&buf, _("%s: unrecognized option '--%s'\n"),
                                  argv[0], nextchar);
#else
                  fprintf (stderr, _("%s: unrecognized option '--%s'\n"),
                           argv[0], nextchar);
#endif
                }
              else
                {
                  /* +option or -option */
#if defined _LIBC && defined USE_IN_LIBIO
                  n = __asprintf (&buf, _("%s: unrecognized option '%c%s'\n"),
                                  argv[0], argv[optind][0], nextchar);
#else
                  fprintf (stderr, _("%s: unrecognized option '%c%s'\n"),
                           argv[0], argv[optind][0], nextchar);
#endif
                }

#if defined _LIBC && defined USE_IN_LIBIO
              if (n >= 0)
                {
                  if (_IO_fwide (stderr, 0) > 0)
                    __fwprintf (stderr, L"%s", buf);
                  else
                    fputs (buf, stderr);

                  free (buf);
                }
#endif
```

```
            }
          nextchar = (char *) "";
          optind++;
          optopt = 0;
          return '?';
        }
    }

  /* Look at and handle the next short option-character.  */

  {
    char c = *nextchar++;
    char *temp = my_index (optstring, c);

    /* Increment `optind' when we start to process its last character.  */
    if (*nextchar == '\0')
      ++optind;

    if (temp == NULL || c == ':')
      {
        if (print_errors)
          {
#if defined _LIBC && defined USE_IN_LIBIO
            char *buf;
            int n;
#endif

            if (posixly_correct)
              {
                /* 1003.2 specifies the format of this message.  */
#if defined _LIBC && defined USE_IN_LIBIO
                n = __asprintf (&buf, _("%s: illegal option -- %c\n"),
                                argv[0], c);
#else
                fprintf (stderr, _("%s: illegal option -- %c\n"), argv[0], c);
#endif
              }
            else
              {
#if defined _LIBC && defined USE_IN_LIBIO
                n = __asprintf (&buf, _("%s: invalid option -- %c\n"),
                                argv[0], c);
#else
                fprintf (stderr, _("%s: invalid option -- %c\n"), argv[0], c);
#endif
              }

#if defined _LIBC && defined USE_IN_LIBIO
            if (n >= 0)
              {
                if (_IO_fwide (stderr, 0) > 0)
                  __fwprintf (stderr, L"%s", buf);
                else
                  fputs (buf, stderr);

                free (buf);
              }
#endif
          }
        optopt = c;
        return '?';
      }
```

```
    /* Convenience. Treat POSIX -W foo same as long option --foo */
    if (temp[0] == 'W' && temp[1] == ';')
      {
        char *nameend;
        const struct option *p;
        const struct option *pfound = NULL;
        int exact = 0;
        int ambig = 0;
        int indfound = 0;
        int option_index;

        /* This is an option that requires an argument.  */
        if (*nextchar != '\0')
          {
            optarg = nextchar;
            /* If we end this ARGV-element by taking the rest as an arg,
               we must advance to the next element now.  */
            optind++;
          }
        else if (optind == argc)
          {
            if (print_errors)
              {
                /* 1003.2 specifies the format of this message.  */
#if defined _LIBC && defined USE_IN_LIBIO
                char *buf;

                if (__asprintf (&buf,
                                _("%s: option requires an argument -- %c\n"),
                                argv[0], c) >= 0)
                  {
                    if (_IO_fwide (stderr, 0) > 0)
                      __fwprintf (stderr, L"%s", buf);
                    else
                      fputs (buf, stderr);

                    free (buf);
                  }
#else
                fprintf (stderr, _("%s: option requires an argument -- %c\n"),
                         argv[0], c);
#endif
              }
            optopt = c;
            if (optstring[0] == ':')
              c = ':';
            else
              c = '?';
            return c;
          }
        else
          /* We already incremented `optind' once;
             increment it again when taking next ARGV-elt as argument.  */
          optarg = argv[optind++];

        /* optarg is now the argument, see if it's in the
           table of longopts.  */

        for (nextchar = nameend = optarg; *nameend && *nameend != '='; nameend++)

          /* Do nothing.  */ ;
```

```
        /* Test all long options for either exact match
           or abbreviated matches.  */
        for (p = longopts, option_index = 0; p->name; p++, option_index++)
          if (!strncmp (p->name, nextchar, nameend - nextchar))
            {
              if ((unsigned int) (nameend - nextchar) == strlen (p->name))
                {
                  /* Exact match found.  */
                  pfound = p;
                  indfound = option_index;
                  exact = 1;
                  break;
                }
              else if (pfound == NULL)
                {
                  /* First nonexact match found.  */
                  pfound = p;
                  indfound = option_index;
                }
              else
                /* Second or later nonexact match found.  */
                ambig = 1;
            }
        if (ambig && !exact)
          {
            if (print_errors)
              {
#if defined _LIBC && defined USE_IN_LIBIO
                char *buf;

                if (__asprintf (&buf, _("%s: option `-W %s' is ambiguous\n"),
                                argv[0], argv[optind]) >= 0)
                  {
                    if (_IO_fwide (stderr, 0) > 0)
                      __fwprintf (stderr, L"%s", buf);
                    else
                      fputs (buf, stderr);

                    free (buf);
                  }
#else
                fprintf (stderr, _("%s: option `-W %s' is ambiguous\n"),
                         argv[0], argv[optind]);
#endif
              }
            nextchar += strlen (nextchar);
            optind++;
            return '?';
          }
        if (pfound != NULL)
          {
            option_index = indfound;
            if (*nameend)
              {
                /* Don't test has_arg with >, because some C compilers don't
                   allow it to be used on enums.  */
                if (pfound->has_arg)
                  optarg = nameend + 1;
                else
                  {
                    if (print_errors)
                      {
```

```
#if defined _LIBC && defined USE_IN_LIBIO
                        char *buf;

                        if (__asprintf (&buf, _("\
%s: option '-W %s' doesn't allow an argument\n"),
                                        argv[0], pfound->name) >= 0)
                          {
                            if (_IO_fwide (stderr, 0) > 0)
                              __fwprintf (stderr, L"%s", buf);
                            else
                              fputs (buf, stderr);

                            free (buf);
                          }
#else
                        fprintf (stderr, _("\
%s: option '-W %s' doesn't allow an argument\n"),
                                 argv[0], pfound->name);
#endif
                      }

                    nextchar += strlen (nextchar);
                    return '?';
                  }
              }
            else if (pfound->has_arg == 1)
              {
                if (optind < argc)
                  optarg = argv[optind++];
                else
                  {
                    if (print_errors)
                      {
#if defined _LIBC && defined USE_IN_LIBIO
                        char *buf;

                        if (__asprintf (&buf, _("\
%s: option '%s' requires an argument\n"),
                                        argv[0], argv[optind - 1]) >= 0)
                          {
                            if (_IO_fwide (stderr, 0) > 0)
                              __fwprintf (stderr, L"%s", buf);
                            else
                              fputs (buf, stderr);

                            free (buf);
                          }
#else
                        fprintf (stderr,
                                 _("%s: option '%s' requires an argument\n"),
                                 argv[0], argv[optind - 1]);
#endif
                      }
                    nextchar += strlen (nextchar);
                    return optstring[0] == ':' ? ':' : '?';
                  }
              }
            nextchar += strlen (nextchar);
            if (longind != NULL)
              *longind = option_index;
            if (pfound->flag)
              {
```

```
                *(pfound->flag) = pfound->val;
                return 0;
              }
            return pfound->val;
          }
          nextchar = NULL;
          return 'W';   /* Let the application handle it.   */
      }
    if (temp[1] == ':')
      {
        if (temp[2] == ':')
          {
            /* This is an option that accepts an argument optionally.  */
            if (*nextchar != '\0')
              {
                optarg = nextchar;
                optind++;
              }
            else
              optarg = NULL;
            nextchar = NULL;
          }
        else
          {
            /* This is an option that requires an argument.  */
            if (*nextchar != '\0')
              {
                optarg = nextchar;
                /* If we end this ARGV-element by taking the rest as an arg,
                   we must advance to the next element now.  */
                optind++;
              }
            else if (optind == argc)
              {
                if (print_errors)
                  {
                    /* 1003.2 specifies the format of this message.  */
#if defined _LIBC && defined USE_IN_LIBIO
                    char *buf;

                    if (__asprintf (&buf, _("\
%s: option requires an argument -- %c\n"),
                                    argv[0], c) >= 0)
                      {
                        if (_IO_fwide (stderr, 0) > 0)
                          __fwprintf (stderr, L"%s", buf);
                        else
                          fputs (buf, stderr);

                        free (buf);
                      }
#else
                    fprintf (stderr,
                             _("%s: option requires an argument -- %c\n"),
                             argv[0], c);
#endif
                  }
                optopt = c;
                if (optstring[0] == ':')
                  c = ':';
                else
                  c = '?';
```

```
            }
          else
            /* We already incremented `optind' once;
               increment it again when taking next ARGV-elt as argument.  */
            optarg = argv[optind++];
          nextchar = NULL;
        }
    }
  return c;
 }
}
```

7.22.3.2 char* getenv ( )

7.22.3.3 int getopt (int *argc,* char *const * *argv,* const char * *optstring*)

Definition at line 1196 of file getopt.c.

```
{
  return _getopt_internal (argc, argv, optstring,
                           (const struct option *) 0,
                           (int *) 0,
                           0);
}
```

7.22.4 Variable Documentation

7.22.4.1 int __getopt_initialized attribute_hidden

Definition at line 143 of file getopt.c.

7.22.4.2 char* optarg

Definition at line 122 of file getopt.c.

7.22.4.3 int opterr = 1

Definition at line 157 of file getopt.c.

7.22.4.4 int optind = 1

Definition at line 137 of file getopt.c.

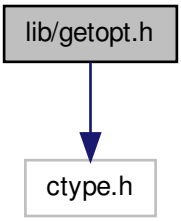
7.22.4.5 int optopt = '?'

Definition at line 163 of file getopt.c.

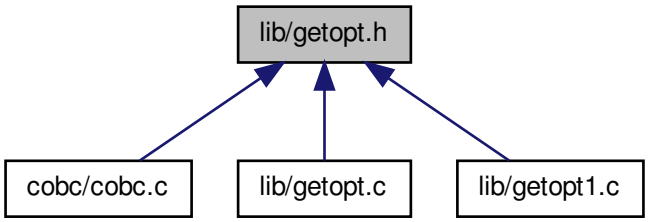
7.23 lib/getopt.h File Reference

```
#include <ctype.h>
```

Include dependency graph for getopt.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct option

Defines

- #define _GETOPT_H 1
- #define no_argument 0
- #define required_argument 1

- #define optional_argument 2

Functions

- int getopt ()
- int getopt_long ()
- int getopt_long_only ()
- int _getopt_internal ()

Variables

- char * optarg
- int optind
- int opterr
- int optopt

7.23.1 Define Documentation

7.23.1.1 #define _GETOPT_H 1

Definition at line 23 of file getopt.h.

7.23.1.2 #define no_argument 0

Definition at line 110 of file getopt.h.

7.23.1.3 #define optional_argument 2

Definition at line 112 of file getopt.h.

7.23.1.4 #define required_argument 1

Definition at line 111 of file getopt.h.

7.23.2 Function Documentation

7.23.2.1 int _getopt_internal ( )

7.23.2.2 int getopt ( )

7.23.2.3 int getopt_long ( )

7.23.2.4 int getopt_long_only ( )

7.23.3 Variable Documentation

7.23.3.1 char* optarg

Definition at line 122 of file getopt.c.

7.23.3.2 int opterr

Definition at line 157 of file getopt.c.

7.23.3.3 int optind

Definition at line 137 of file getopt.c.

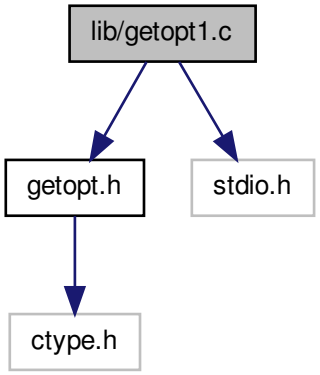
7.23.3.4 int optopt

Definition at line 163 of file getopt.c.

7.24 lib/getopt1.c File Reference

```
#include "getopt.h"
#include <stdio.h>
```

Include dependency graph for getopt1.c:



Defines

- #define const
- #define GETOPT_INTERFACE_VERSION 2
- #define NULL 0

Functions

- int getopt_long (int argc, char ∗const ∗argv, const char ∗options, const struct option ∗long_options, int ∗opt_index)
- int getopt_long_only (int argc, char ∗const ∗argv, const char ∗options, const struct option ∗long_options, int ∗opt_index)

7.24.1 Define Documentation

7.24.1.1 #define const

Definition at line 35 of file getopt1.c.

7.24.1.2 #define GETOPT_INTERFACE_VERSION 2

Definition at line 49 of file getopt1.c.

7.24.1.3 #define NULL 0

Definition at line 67 of file getopt1.c.

7.24.2 Function Documentation

7.24.2.1 int getopt_long (int *argc,* char ∗const ∗ *argv,* const char ∗ *options,* const struct option ∗ *long_options,* int ∗ *opt_index*)

Definition at line 71 of file getopt1.c.

```
{
  return _getopt_internal (argc, argv, options, long_options, opt_index, 0);
}
```

7.24.2.2 int getopt_long_only (int *argc,* char ∗const ∗ *argv,* const char ∗ *options,* const struct option ∗ *long_options,* int ∗ *opt_index*)

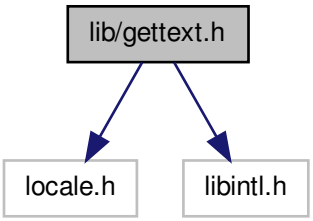
Definition at line 87 of file getopt1.c.

```
{
  return _getopt_internal (argc, argv, options, long_options, opt_index, 1);
}
```

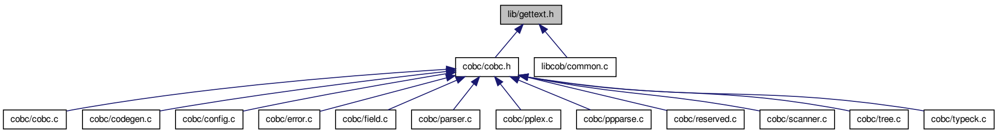
7.25 lib/gettext.h File Reference

```
#include <locale.h>
#include <libintl.h>
```

Include dependency graph for gettext.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define gettext_noop(String) String
- #define _(s) gettext(s)
- #define N_(s) gettext_noop(s)
- #define gettext_noop(String) String
- #define _(s) gettext(s)
- #define N_(s) gettext_noop(s)
- #define gettext_noop(String) String
- #define _(s) gettext(s)
- #define N_(s) gettext_noop(s)
- #define gettext_noop(String) String
- #define _(s) gettext(s)
- #define N_(s) gettext_noop(s)
- #define gettext_noop(String) String
- #define _(s) gettext(s)
- #define N_(s) gettext_noop(s)
- #define gettext_noop(String) String
- #define _(s) gettext(s)
- #define N_(s) gettext_noop(s)
- #define gettext_noop(String) String
- #define _(s) gettext(s)
- #define N_(s) gettext_noop(s)
- #define gettext(Msgid) ((const char ∗) (Msgid))
- #define dgettext(Domainname, Msgid) ((const char ∗) (Msgid))
- #define dcgettext(Domainname, Msgid, Category) ((const char ∗) (Msgid))
- #define ngettext(Msgid1, Msgid2, N) ((N) == 1 ? (const char ∗) (Msgid1) : (const char ∗) (Msgid2))
- #define dngettext(Domainname, Msgid1, Msgid2, N) ((N) == 1 ? (const char ∗) (Msgid1) : (const char ∗) (Msgid2))
- #define dcngettext(Domainname, Msgid1, Msgid2, N, Category) ((N) == 1 ? (const char ∗) (Msgid1) : (const char ∗) (Msgid2))
- #define textdomain(Domainname) ((const char ∗) (Domainname))
- #define bindtextdomain(Domainname, Dirname) ((const char ∗) (Dirname))
- #define bind_textdomain_codeset(Domainname, Codeset) ((const char ∗) (Codeset))
- #define gettext_noop(String) String
- #define _(s) gettext(s)
- #define N_(s) gettext_noop(s)

7.25.1 Define Documentation

7.25.1.1 #define _(*s*) gettext(s)

7.25.1.2 #define _(*s*) gettext(s)

7.25.1.3 #define _(*s*) gettext(s)

7.25.1.4 #define _(*s*) gettext(s)

7.25.1.5 #define _(*s*) gettext(s)

Definition at line 60 of file gettext.h.

7.25.1.6 #define _(*s*) gettext(s)

7.25.1.7 #define _(*s*) gettext(s)

7.25.1.8 #define _(*s*) gettext(s)

7.25.1.9 #define bind_textdomain_codeset(*Domainname, Codeset*) ((const char *) (Codeset))

Definition at line 47 of file gettext.h.

7.25.1.10 #define bindtextdomain(*Domainname, Dirname*) ((const char *) (Dirname))

Definition at line 46 of file gettext.h.

7.25.1.11 #define dcgettext(*Domainname, Msgid, Category*) ((const char *) (Msgid))

Definition at line 38 of file gettext.h.

7.25.1.12 #define dcngettext(*Domainname, Msgid1, Msgid2, N, Category*) ((N) == 1 ? (const char *) (Msgid1) : (const char *) (Msgid2))

Definition at line 43 of file gettext.h.

7.25.1.13 #define dgettext(*Domainname, Msgid*) ((const char *) (Msgid))

Definition at line 37 of file gettext.h.

7.25.1.14 #define dngettext(*Domainname, Msgid1, Msgid2, N*) ((N) == 1 ? (const char *) (Msgid1) : (const char *) (Msgid2))

Definition at line 41 of file gettext.h.

7.25.1.15 #define gettext(*Msgid*) ((const char *) (Msgid))

Definition at line 36 of file gettext.h.

7.25.1.16 #define gettext_noop(*String*) String

Definition at line 58 of file gettext.h.

7.25.1.17 #define gettext_noop(*String*) String

7.25.1.18 #define gettext_noop(*String*) String

7.25.1.19 #define gettext_noop(*String*) String

7.25.1.20 #define gettext_noop(*String*) String

7.25.1.21 #define gettext_noop(*String*) String

7.25.1.22 #define gettext_noop(*String*) String

7.25.1.23 #define gettext_noop(*String*) String

7.25.1.24 #define N_(*s*) gettext_noop(s)

7.25.1.25 #define N_(*s*) gettext_noop(s)

7.25.1.26 #define N_(*s*) gettext_noop(s)

7.25.1.27 #define N_(*s*) gettext_noop(s)

7.25.1.28 #define N_(*s*) gettext_noop(s)

Definition at line 61 of file gettext.h.

7.25.1.29 #define N_(*s*) gettext_noop(s)

7.25.1.30 #define N_(*s*) gettext_noop(s)

7.25.1.31 #define N_(*s*) gettext_noop(s)

7.25.1.32 #define ngettext(*Msgid1, Msgid2, N*) ((N) == 1 ? (const char *) (Msgid1) : (const char *) (Msgid2))

Definition at line 39 of file gettext.h.

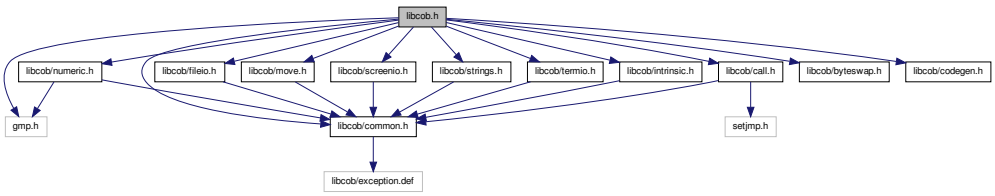
7.25.1.33 #define textdomain(*Domainname*) ((const char *) (Domainname))

Definition at line 45 of file gettext.h.

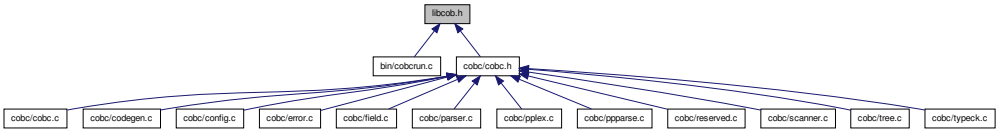
7.26 libcob.h File Reference

```
#include <gmp.h>
#include <libcob/byteswap.h>
#include <libcob/call.h>
#include <libcob/common.h>
#include <libcob/fileio.h>
#include <libcob/move.h>
#include <libcob/numeric.h>
#include <libcob/screenio.h>
#include <libcob/strings.h>
#include <libcob/termio.h>
#include <libcob/intrinsic.h>
#include <libcob/codegen.h>
```

Include dependency graph for libcob.h:

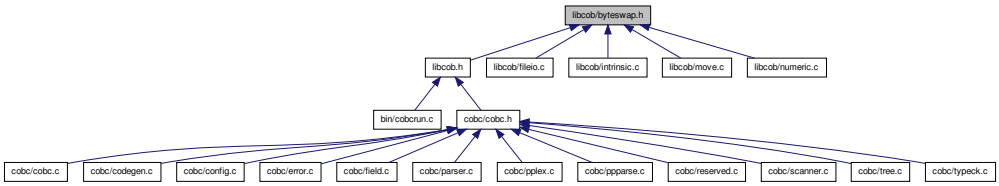


This graph shows which files directly or indirectly include this file:



7.27 libcob/byteswap.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define COB_BSWAP_16_CONSTANT(val)
- #define COB_BSWAP_32_CONSTANT(val)
- #define COB_BSWAP_64_CONSTANT(val)
- #define COB_BSWAP_16(val) (COB_BSWAP_16_CONSTANT (val))
- #define COB_BSWAP_32(val) (COB_BSWAP_32_CONSTANT (val))
- #define COB_BSWAP_64(val) (COB_BSWAP_64_CONSTANT (val))

7.27.1 Define Documentation

7.27.1.1 #define COB_BSWAP_16(val) (COB_BSWAP_16_CONSTANT (val))

Definition at line 183 of file byteswap.h.

7.27.1.2 #define COB_BSWAP_16_CONSTANT(val)

Value:

```
((unsigned short) (     \
    (unsigned short) ((unsigned short) (val) >> 8) |            \
    (unsigned short) ((unsigned short) (val) << 8)))
```

Definition at line 32 of file byteswap.h.

7.27.1.3 #define COB_BSWAP_32(val) (COB_BSWAP_32_CONSTANT (val))

Definition at line 184 of file byteswap.h.

7.27.1.4 #define COB_BSWAP_32_CONSTANT(val)

Value:

```
((unsigned int) (               \
    (((unsigned int) (val) & (unsigned int) 0x000000ffU) << 24) |       \
    (((unsigned int) (val) & (unsigned int) 0x0000ff00U) <<  8) |       \
    (((unsigned int) (val) & (unsigned int) 0x00ff0000U) >>  8) |       \
    (((unsigned int) (val) & (unsigned int) 0xff000000U) >> 24)))
```

Definition at line 36 of file byteswap.h.

7.27.1.5 #define COB_BSWAP_64(val) (COB_BSWAP_64_CONSTANT (val))

Definition at line 185 of file byteswap.h.

7.27.1.6 #define COB_BSWAP_64_CONSTANT(val)

Value:

```
((unsigned long long) ( \
    (((unsigned long long) (val) &                              \
      (unsigned long long) 0x00000000000000ffULL) << 56) |      \
    (((unsigned long long) (val) &                              \
      (unsigned long long) 0x000000000000ff00ULL) << 40) |      \
    (((unsigned long long) (val) &                              \
      (unsigned long long) 0x0000000000ff0000ULL) << 24) |      \
    (((unsigned long long) (val) &                              \
      (unsigned long long) 0x00000000ff000000ULL) <<  8) |      \
    (((unsigned long long) (val) &                              \
      (unsigned long long) 0x000000ff00000000ULL) >>  8) |      \
    (((unsigned long long) (val) &                              \
      (unsigned long long) 0x0000ff0000000000ULL) >> 24) |      \
    (((unsigned long long) (val) &                              \
      (unsigned long long) 0x00ff000000000000ULL) >> 40) |      \
    (((unsigned long long) (val) &                              \
      (unsigned long long) 0xff00000000000000ULL) >> 56)))
```

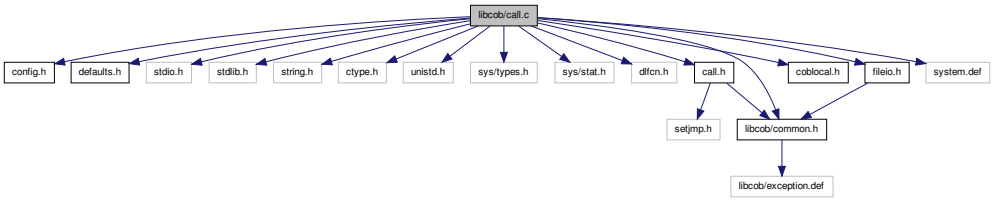
Definition at line 42 of file byteswap.h.

7.28 libcob/call.c File Reference

```
#include "config.h"
#include "defaults.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dlfcn.h>
#include "call.h"
#include "common.h"
#include "coblocal.h"
#include "fileio.h"
#include "system.def"
```

Include dependency graph for call.c:



Classes

- struct struct_handle
- struct call_hash
- struct system_table

Defines

- #define __USE_GNU 1
- #define lt_dlopen(x) dlopen(x, RTLD_LAZY | RTLD_GLOBAL)
- #define lt_dlsym(x, y) dlsym(x, y)
- #define lt_dlclose(x) dlclose(x)
- #define lt_dlerror() dlerror()
- #define lt_dlhandle void *
- #define PATHSEPC ':'
- #define PATHSEPS ":"
- #define COB_MAX_COBCALL_PARMS 16
- #define CALL_BUFF_SIZE 256
- #define CALL_FILEBUFF_SIZE 2048
- #define HASH_SIZE 131
- #define COB_SYSTEM_GEN(x, y, z) { x, (void *)z },

Functions

- const char * cob_resolve_error (void)
- void cob_call_error (void)
- void cob_set_cancel (const char *name, void *entry, void *cancel)
- void * cob_resolve (const char *name)
- void * cob_resolve_1 (const char *name)
- void * cob_call_resolve (const cob_field *f)
- void * cob_call_resolve_1 (const cob_field *f)
- void cobcancel (const char *name)
- void cob_field_cancel (const cob_field *f)
- void cob_init_call (void)
- int cobcall (const char *name, const int argc, void **argv)
- int cobfunc (const char *name, const int argc, void **argv)
- void * cobsavenv (struct cobjmp_buf *jbuf)
- void * cobsavenv2 (struct cobjmp_buf *jbuf, const int jsize)
- void coblongjmp (struct cobjmp_buf *jbuf)

7.28.1 Define Documentation

7.28.1.1 #define __USE_GNU 1

Definition at line 44 of file call.c.

7.28.1.2 #define CALL_BUFF_SIZE 256

Definition at line 100 of file call.c.

7.28.1.3 #define CALL_FILEBUFF_SIZE 2048

Definition at line 101 of file call.c.

7.28.1.4 #define COB_MAX_COBCALL_PARMS 16

Definition at line 99 of file call.c.

7.28.1.5 #define COB_SYSTEM_GEN(*x*, *y*, *z*) { x, (void *)z },

Definition at line 152 of file call.c.

7.28.1.6 #define HASH_SIZE 131

Definition at line 130 of file call.c.

7.28.1.7 #define lt_dlclose(x) dlclose(x)

Definition at line 48 of file call.c.

7.28.1.8 #define lt_dlerror( ) dlerror()

Definition at line 49 of file call.c.

7.28.1.9 #define lt_dlhandle void *

Definition at line 50 of file call.c.

7.28.1.10 #define lt_dlopen(x) dlopen(x, RTLD_LAZY | RTLD_GLOBAL)

Definition at line 46 of file call.c.

7.28.1.11 #define lt_dlsym(x, y) dlsym(x, y)

Definition at line 47 of file call.c.

7.28.1.12 #define PATHSEPC ':'

Definition at line 95 of file call.c.

7.28.1.13 #define PATHSEPS ":"

Definition at line 96 of file call.c.

7.28.2 Function Documentation

7.28.2.1 void cob_call_error (void)

Definition at line 286 of file call.c.

```
{
        const char      *s;

        s = cob_resolve_error ();
        if (!s) {
                s = "Unknown error";
        }
        cob_runtime_error ("%s", s);
        cob_stop_run (1);
}
```

7.28.2.2 void* cob_call_resolve (const cob_field * f)

Definition at line 454 of file call.c.

```
{
        char    *buff;

        buff = cob_get_buff (f->size + 1);
        cob_field_to_string (f, buff);
        return cob_resolve (buff);
}
```

7.28.2.3 void* cob_call_resolve_1 (const cob_field * f)

Definition at line 464 of file call.c.

```
{
        void    *p;

        p = cob_call_resolve (f);
        if (unlikely(!p)) {
                cob_call_error ();
        }
        return p;
}
```

7.28.2.4 void cob_field_cancel (const cob_field * f)

Definition at line 504 of file call.c.

```
{
        char    *name;

        name = cob_get_buff (f->size + 1);
        cob_field_to_string (f, name);
        cobcancel (name);
}
```

7.28.2.5 void cob_init_call (void)

Definition at line 514 of file call.c.

```
{
        char                            *buff;
        char                            *s;
        char                            *p;
        const struct system_table       *psyst;
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
        lt_dlhandle                     libhandle;
#endif
```

```
        size_t                          i;
        struct stat                     st;

#ifndef USE_LIBDL
        lt_dlinit ();
#endif

        /* big enough for anything from libdl/libltdl */
        resolve_error_buff = cob_malloc (CALL_BUFF_SIZE);

#ifndef COB_ALT_HASH
        call_table = cob_malloc (sizeof (struct call_hash *) * HASH_SIZE);
#endif

        call_filename_buff = cob_malloc (CALL_FILEBUFF_SIZE);
        call_entry_buff = cob_malloc (COB_SMALL_BUFF);
        call_entry2_buff = cob_malloc (COB_SMALL_BUFF);
        s = getenv ("COB_LOAD_CASE");
        if (s != NULL) {
                if (strcasecmp (s, "LOWER") == 0) {
                        name_convert = 1;
                } else if (strcasecmp (s, "UPPER") == 0) {
                        name_convert = 2;
                }
        }

        buff = cob_malloc (COB_MEDIUM_BUFF);
        s = getenv ("COB_LIBRARY_PATH");
        if (s == NULL) {
                snprintf (buff, COB_MEDIUM_MAX, ".%s%s",
                          PATHSEPS, COB_LIBRARY_PATH);
        } else {
                snprintf (buff, COB_MEDIUM_MAX, "%s%s.%s%s",
                          s, PATHSEPS, PATHSEPS, COB_LIBRARY_PATH);
        }
        cob_set_library_path (buff);

#ifndef COB_BORKED_DLOPEN
        mainhandle = lt_dlopen (NULL);
#endif

        s = getenv ("COB_PRE_LOAD");
        if (s != NULL) {
                p = cob_strdup (s);
                s = strtok (p, PATHSEPS);
                for (; s; s = strtok (NULL, PATHSEPS)) {
                        for (i = 0; i < resolve_size; ++i) {
                                buff[COB_MEDIUM_MAX] = 0;
                                snprintf (buff, COB_MEDIUM_MAX, "%s/%s.%s",
                                          resolve_path[i], s, COB_MODULE_EXT);
                                if (stat (buff, &st) == 0) {
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
                                        if ((libhandle = lt_dlopen (buff)) !=
      NULL) {
                                                cache_handle (libhandle);
#else
                                        if (lt_dlopen (buff) != NULL) {
#endif
                                                break;
                                        }
                                }
                        }
```

```
                }
                free (p);
        }
        free (buff);
        call_buffer = cob_malloc (CALL_BUFF_SIZE);
        call_lastsize = CALL_BUFF_SIZE;
        for (psyst = (struct system_table *)&system_tab[0]; psyst->syst_name; ++p
      syst) {
                insert (psyst->syst_name, psyst->syst_call, NULL);
        }
}
```

7.28.2.6 void* cob_resolve (const char * *name*)

Definition at line 317 of file call.c.

```
{
        unsigned char           *p;
        const unsigned char     *s;
        void                    *func;
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
        struct struct_handle    *chkhandle;
#endif
        lt_dlhandle             handle;
        size_t                  i;
        struct stat             st;

/* Checked in generated code
        if (!cob_initialized) {
                fputs ("cob_init() must be called before cob_resolve()", stderr);

                cob_stop_run (1);
        }
*/

        /* search the cache */
        cob_exception_code = 0;
        func = lookup (name);
        if (func) {
                return func;
        }

        /* encode program name */
        p = (unsigned char *)call_entry_buff;
        s = (const unsigned char *)name;
        if (unlikely(isdigit (*s))) {
                p += sprintf ((char *)p, "_%02X", *s++);
        }
        for (; *s; ++s) {
                if (likely(isalnum (*s) || *s == '_')) {
                        *p++ = *s;
                } else if (*s == '-') {
                        *p++ = '_';
                        *p++ = '_';
                } else {
                        p += sprintf ((char *)p, "_%02X", *s);
                }
        }
        *p = 0;
```

```
        /* search the main program */
        if (mainhandle != NULL && (func = lt_dlsym (mainhandle, call_entry_buff))
       != NULL) {
                insert (name, func, NULL);
                resolve_error = NULL;
                return func;
        }

        /* Search preloaded modules */
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
        for (chkhandle = pre_handle; chkhandle; chkhandle = chkhandle->next) {
                if ((func = lt_dlsym (chkhandle->preload_handle, call_entry_buff)
      ) != NULL) {
                        insert (name, func, NULL);
                        resolve_error = NULL;
                        return func;
                }
        }
#endif
#if     defined(USE_LIBDL) && defined (RTLD_DEFAULT)
        if ((func = lt_dlsym (RTLD_DEFAULT, call_entry_buff)) != NULL) {
                insert (name, func, NULL);
                resolve_error = NULL;
                return func;
        }
#endif

        s = (const unsigned char *)name;
        if (unlikely(name_convert != 0)) {
                s = (const unsigned char *)name;
                p = call_entry2_buff;
                for (; *s; ++s) {
                        if (name_convert == 1 && isupper (*s)) {
                                *p++ = tolower (*s);
                        } else if (name_convert == 2 && islower (*s)) {
                                *p++ = toupper (*s);
                        } else {
                                *p++ = *s;
                        }
                }
                *p = 0;
                s = (const unsigned char *)call_entry2_buff;
        }

        /* search external modules */
        for (i = 0; i < resolve_size; ++i) {
                call_filename_buff[CALL_FILEBUFF_SIZE - 1] = 0;
                if (resolve_path[i] == NULL) {
                        snprintf (call_filename_buff, CALL_FILEBUFF_SIZE - 1,
                                  "%s.%s", s, COB_MODULE_EXT);
                } else {
                        snprintf (call_filename_buff, CALL_FILEBUFF_SIZE - 1,
                                  "%s/%s.%s", resolve_path[i], s, COB_MODULE_EXT)
      ;
                }
                if (stat (call_filename_buff, &st) == 0) {
                        if ((handle = lt_dlopen (call_filename_buff)) != NULL) {
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
                                /* Candidate for future calls */
                                cache_handle (handle);
#endif
```

```
                                if ((func = lt_dlsym (handle, call_entry_buff)) !
      = NULL) {
                                        insert (name, func, NULL);
                                        resolve_error = NULL;
                                        return func;
                                }
                        }
                        resolve_error_buff[CALL_BUFF_SIZE - 1] = 0;
                        strncpy (resolve_error_buff, lt_dlerror (),
                                 CALL_BUFF_SIZE - 1);
                        resolve_error = resolve_error_buff;
                        cob_set_exception (COB_EC_PROGRAM_NOT_FOUND);
                        return NULL;
                }
        }
        resolve_error_buff[CALL_BUFF_SIZE - 1] = 0;
        snprintf (resolve_error_buff, CALL_BUFF_SIZE - 1,
                  "Cannot find module '%s'", name);
        resolve_error = resolve_error_buff;
        cob_set_exception (COB_EC_PROGRAM_NOT_FOUND);
        return NULL;
}
```

7.28.2.7 void* cob_resolve_1 (const char * *name*)

Definition at line 442 of file call.c.

```
{
        void    *p;

        p = cob_resolve (name);
        if (unlikely(!p)) {
                cob_call_error ();
        }
        return p;
}
```

7.28.2.8 const char* cob_resolve_error (void)

Definition at line 277 of file call.c.

```
{
        const char      *p = resolve_error;

        resolve_error = NULL;
        return p;
}
```

7.28.2.9 void cob_set_cancel (const char * *name*, void * *entry*, void * *cancel*)

Definition at line 299 of file call.c.

```
{
        struct call_hash        *p;

#ifdef  COB_ALT_HASH
        for (p = call_table; p; p = p->next) {
#else
        for (p = call_table[hash ((const unsigned char *)name)]; p; p = p->next)
      {
#endif
                if (strcmp (name, p->name) == 0) {
                        p->cancel = cancel;
                        return;
                }
        }
        insert (name, entry, cancel);
}
```

7.28.2.10 int cobcall (const char * *name*, const int *argc*, void ** *argv*)

Definition at line 596 of file call.c.

```
{
        int     i;
        union {
                void    *(*funcptr)();
                int     (*funcint)();
                void    *func_void;
        } unifunc;
        void    *pargv[16];

        if (unlikely(!cob_initialized)) {
                cob_runtime_error ("'cobcall' - Runtime has not been initialized"
      );
                cob_stop_run (1);
        }
        if (argc < 0 || argc > 16) {
                cob_runtime_error ("Invalid number of arguments to 'cobcall'");
                cob_stop_run (1);
        }
        if (unlikely(!name)) {
                cob_runtime_error ("NULL name parameter passed to 'cobcall'");
                cob_stop_run (1);
        }
        unifunc.func_void = cob_resolve_1 (name);
        memset (pargv, 0, sizeof(pargv));
        /* Set number of parameters */
        cob_call_params = argc;
        for (i = 0; i < argc; ++i) {
                pargv[i] = argv[i];
        }
        return unifunc.funcint (pargv[0], pargv[1], pargv[2], pargv[3],
                                pargv[4], pargv[5], pargv[6], pargv[7],
                                pargv[8], pargv[9], pargv[10], pargv[11],
                                pargv[12], pargv[13], pargv[14], pargv[15]);
}
```

7.28.2.11 void cobcancel (const char * *name*)

Definition at line 476 of file call.c.

```
{
        struct call_hash        *p;
        union {
                int     (*cancel_func)(int, ...);
                void    *cancel_void;
        } unicanc;

        if (unlikely(!name)) {
                cob_runtime_error ("NULL name parameter passed to 'cobcancel'");
                cob_stop_run (1);
        }
#ifdef  COB_ALT_HASH
        for (p = call_table; p; p = p->next) {
#else
        for (p = call_table[hash ((const unsigned char *)name)]; p; p = p->next)
      {
#endif
                if (strcmp (name, p->name) == 0) {
                        if (p->cancel && !p->flag_is_active) {
                                unicanc.cancel_void = p->cancel;
                                unicanc.cancel_func (-1, NULL, NULL, NULL, NULL,
                                                     NULL, NULL, NULL, NULL);
                        }
                }
        }
}
```

7.28.2.12 int cobfunc (const char * *name*, const int *argc*, void ** *argv*)

Definition at line 632 of file call.c.

```
{
        int     ret;

        if (unlikely(!cob_initialized)) {
                cob_runtime_error ("'cobfunc' - Runtime has not been initialized"
      );
                cob_stop_run (1);
        }
        ret = cobcall (name, argc, argv);
        cobcancel (name);
        return ret;
}
```

7.28.2.13 void coblongjmp (struct cobjmp_buf * *jbuf*)

Definition at line 671 of file call.c.

```
{
        if (unlikely(!jbuf)) {
```

```
                cob_runtime_error ("NULL name parameter passed to 'coblongjmp'");

                cob_stop_run (1);
        }
        if (!cobjmp_primed) {
                cob_runtime_error ("Call to 'coblongjmp' with no prior 'cobsetjmp
      '");
                cob_stop_run (1);
        }
        cobjmp_primed = 0;
        longjmp (jbuf->cbj_jmp_buf, 1);
}
```

7.28.2.14 void* cobsavenv (struct cobjmp_buf * *jbuf*)

Definition at line 646 of file call.c.

```
{
        if (unlikely(!jbuf)) {
                cob_runtime_error ("NULL name parameter passed to 'cobsavenv'");
                cob_stop_run (1);
        }
        if (cobjmp_primed) {
                cob_runtime_error ("Multiple call to 'cobsetjmp'");
                cob_stop_run (1);
        }
        cobjmp_primed = 1;
        return jbuf->cbj_jmp_buf;
}
```

7.28.2.15 void* cobsavenv2 (struct cobjmp_buf * *jbuf,* const int *jsize*)

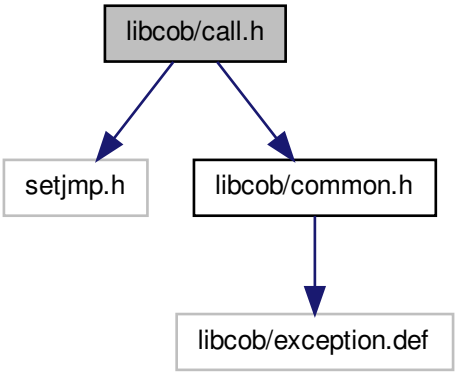
Definition at line 661 of file call.c.

```
{
        int     jtemp;

        /* Shut up compiler */
        jtemp = jsize;
        return cobsavenv (jbuf);
}
```

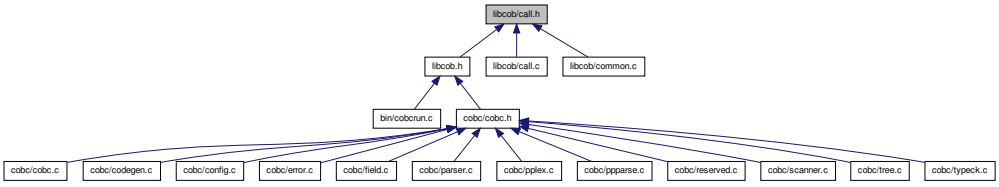
7.29 libcob/call.h File Reference

```
#include <setjmp.h>

#include <libcob/common.h>
```

Include dependency graph for call.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct cobjmp_buf

Defines

- #define cobsetjmp(x) setjmp (cobsavenv (x))

Functions

- void cob_set_cancel (const char *, void *, void *)
- void * cob_resolve (const char *)

- void * cob_resolve_1 (const char *)
- const char * cob_resolve_error (void)
- void * cob_call_resolve (const cob_field *)
- void * cob_call_resolve_1 (const cob_field *)
- void cob_field_cancel (const cob_field *)
- void cobcancel (const char *)
- int cobcall (const char *, const int, void **)
- int cobfunc (const char *, const int, void **)
- void * cobsavenv (struct cobjmp_buf *)
- void * cobsavenv2 (struct cobjmp_buf *, const int)
- void coblongjmp (struct cobjmp_buf *)
- void cob_call_error (void)

7.29.1 Define Documentation

7.29.1.1 #define cobsetjmp(x) setjmp (cobsavenv (x))

Definition at line 49 of file call.h.

7.29.2 Function Documentation

7.29.2.1 void cob_call_error (void)

Definition at line 286 of file call.c.

```
{
        const char      *s;

        s = cob_resolve_error ();
        if (!s) {
                s = "Unknown error";
        }
        cob_runtime_error ("%s", s);
        cob_stop_run (1);
}
```

7.29.2.2 void* cob_call_resolve (const cob_field *)

Definition at line 454 of file call.c.

```
{
        char    *buff;

        buff = cob_get_buff (f->size + 1);
        cob_field_to_string (f, buff);
        return cob_resolve (buff);
}
```

7.29.2.3 void* cob_call_resolve_1 (const cob_field *)

Definition at line 464 of file call.c.

```
{
        void    *p;

        p = cob_call_resolve (f);
        if (unlikely(!p)) {
                cob_call_error ();
        }
        return p;
}
```

7.29.2.4 void cob_field_cancel (const cob_field *)

Definition at line 504 of file call.c.

```
{
        char    *name;

        name = cob_get_buff (f->size + 1);
        cob_field_to_string (f, name);
        cobcancel (name);
}
```

7.29.2.5 void* cob_resolve (const char *)

Definition at line 317 of file call.c.

```
{
        unsigned char           *p;
        const unsigned char     *s;
        void                    *func;
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
        struct struct_handle    *chkhandle;
#endif
        lt_dlhandle             handle;
        size_t                  i;
        struct stat             st;

/* Checked in generated code
        if (!cob_initialized) {
                fputs ("cob_init() must be called before cob_resolve()", stderr);

                cob_stop_run (1);
        }
*/

        /* search the cache */
        cob_exception_code = 0;
        func = lookup (name);
        if (func) {
                return func;
```

```
        }

        /* encode program name */
        p = (unsigned char *)call_entry_buff;
        s = (const unsigned char *)name;
        if (unlikely(isdigit (*s))) {
                p += sprintf ((char *)p, "_%02X", *s++);
        }
        for (; *s; ++s) {
                if (likely(isalnum (*s) || *s == '_')) {
                        *p++ = *s;
                } else if (*s == '-') {
                        *p++ = '_';
                        *p++ = '_';
                } else {
                        p += sprintf ((char *)p, "_%02X", *s);
                }
        }
        *p = 0;

        /* search the main program */
        if (mainhandle != NULL && (func = lt_dlsym (mainhandle, call_entry_buff))
      != NULL) {
                insert (name, func, NULL);
                resolve_error = NULL;
                return func;
        }

        /* Search preloaded modules */
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
        for (chkhandle = pre_handle; chkhandle; chkhandle = chkhandle->next) {
                if ((func = lt_dlsym (chkhandle->preload_handle, call_entry_buff)
     ) != NULL) {
                        insert (name, func, NULL);
                        resolve_error = NULL;
                        return func;
                }
        }
#endif
#if     defined(USE_LIBDL) && defined (RTLD_DEFAULT)
        if ((func = lt_dlsym (RTLD_DEFAULT, call_entry_buff)) != NULL) {
                insert (name, func, NULL);
                resolve_error = NULL;
                return func;
        }
#endif

        s = (const unsigned char *)name;
        if (unlikely(name_convert != 0)) {
                s = (const unsigned char *)name;
                p = call_entry2_buff;
                for (; *s; ++s) {
                        if (name_convert == 1 && isupper (*s)) {
                                *p++ = tolower (*s);
                        } else if (name_convert == 2 && islower (*s)) {
                                *p++ = toupper (*s);
                        } else {
                                *p++ = *s;
                        }
                }
                *p = 0;
                s = (const unsigned char *)call_entry2_buff;
```

```
        }

        /* search external modules */
        for (i = 0; i < resolve_size; ++i) {
                call_filename_buff[CALL_FILEBUFF_SIZE - 1] = 0;
                if (resolve_path[i] == NULL) {
                        snprintf (call_filename_buff, CALL_FILEBUFF_SIZE - 1,
                                  "%s.%s", s, COB_MODULE_EXT);
                } else {
                        snprintf (call_filename_buff, CALL_FILEBUFF_SIZE - 1,
                                  "%s/%s.%s", resolve_path[i], s, COB_MODULE_EXT)
      ;
                }
                if (stat (call_filename_buff, &st) == 0) {
                        if ((handle = lt_dlopen (call_filename_buff)) != NULL) {
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
                                /* Candidate for future calls */
                                cache_handle (handle);
#endif
                                if ((func = lt_dlsym (handle, call_entry_buff)) !
      = NULL) {
                                        insert (name, func, NULL);
                                        resolve_error = NULL;
                                        return func;
                                }
                        }
                        resolve_error_buff[CALL_BUFF_SIZE - 1] = 0;
                        strncpy (resolve_error_buff, lt_dlerror (),
                                 CALL_BUFF_SIZE - 1);
                        resolve_error = resolve_error_buff;
                        cob_set_exception (COB_EC_PROGRAM_NOT_FOUND);
                        return NULL;
                }
        }
        resolve_error_buff[CALL_BUFF_SIZE - 1] = 0;
        snprintf (resolve_error_buff, CALL_BUFF_SIZE - 1,
                  "Cannot find module '%s'", name);
        resolve_error = resolve_error_buff;
        cob_set_exception (COB_EC_PROGRAM_NOT_FOUND);
        return NULL;
}
```

7.29.2.6 void* cob_resolve_1 (const char *)

Definition at line 442 of file call.c.

```
{
        void    *p;

        p = cob_resolve (name);
        if (unlikely(!p)) {
                cob_call_error ();
        }
        return p;
}
```

7.29.2.7 const char* cob_resolve_error (void)

Definition at line 277 of file call.c.

```
{
        const char      *p = resolve_error;

        resolve_error = NULL;
        return p;
}
```

7.29.2.8 void cob_set_cancel (const char * , void * , void *)

Definition at line 299 of file call.c.

```
{
        struct call_hash        *p;

#ifdef  COB_ALT_HASH
        for (p = call_table; p; p = p->next) {
#else
        for (p = call_table[hash ((const unsigned char *)name)]; p; p = p->next)
      {
#endif
                if (strcmp (name, p->name) == 0) {
                        p->cancel = cancel;
                        return;
                }
        }
        insert (name, entry, cancel);
}
```

7.29.2.9 int cobcall (const char * , const int , void **)

Definition at line 596 of file call.c.

```
{
        int     i;
        union {
                void    *(*funcptr)();
                int     (*funcint)();
                void    *func_void;
        } unifunc;
        void    *pargv[16];

        if (unlikely(!cob_initialized)) {
                cob_runtime_error ("'cobcall' - Runtime has not been initialized"
      );
                cob_stop_run (1);
        }
        if (argc < 0 || argc > 16) {
                cob_runtime_error ("Invalid number of arguments to 'cobcall'");
                cob_stop_run (1);
        }
```

```
        if (unlikely(!name)) {
                cob_runtime_error ("NULL name parameter passed to 'cobcall'");
                cob_stop_run (1);
        }
        unifunc.func_void = cob_resolve_1 (name);
        memset (pargv, 0, sizeof(pargv));
        /* Set number of parameters */
        cob_call_params = argc;
        for (i = 0; i < argc; ++i) {
                pargv[i] = argv[i];
        }
        return unifunc.funcint (pargv[0], pargv[1], pargv[2], pargv[3],
                                pargv[4], pargv[5], pargv[6], pargv[7],
                                pargv[8], pargv[9], pargv[10], pargv[11],
                                pargv[12], pargv[13], pargv[14], pargv[15]);
}
```

7.29.2.10 void cobcancel (const char *)

Definition at line 476 of file call.c.

```
{
        struct call_hash        *p;
        union {
                int     (*cancel_func)(int, ...);
                void    *cancel_void;
        } unicanc;

        if (unlikely(!name)) {
                cob_runtime_error ("NULL name parameter passed to 'cobcancel'");
                cob_stop_run (1);
        }
#ifdef  COB_ALT_HASH
        for (p = call_table; p; p = p->next) {
#else
        for (p = call_table[hash ((const unsigned char *)name)]; p; p = p->next)
      {
#endif
                if (strcmp (name, p->name) == 0) {
                        if (p->cancel && !p->flag_is_active) {
                                unicanc.cancel_void = p->cancel;
                                unicanc.cancel_func (-1, NULL, NULL, NULL, NULL,
                                                     NULL, NULL, NULL, NULL);
                        }
                }
        }
}
```

7.29.2.11 int cobfunc (const char * , const int , void **)

Definition at line 632 of file call.c.

```
{
        int     ret;
```

```
        if (unlikely(!cob_initialized)) {
                cob_runtime_error ("'cobfunc' - Runtime has not been initialized"
      );
                cob_stop_run (1);
        }
        ret = cobcall (name, argc, argv);
        cobcancel (name);
        return ret;
}
```

7.29.2.12 void coblongjmp (struct cobjmp_buf *)

Definition at line 671 of file call.c.

```
{
        if (unlikely(!jbuf)) {
                cob_runtime_error ("NULL name parameter passed to 'coblongjmp'");

                cob_stop_run (1);
        }
        if (!cobjmp_primed) {
                cob_runtime_error ("Call to 'coblongjmp' with no prior 'cobsetjmp
      '");
                cob_stop_run (1);
        }
        cobjmp_primed = 0;
        longjmp (jbuf->cbj_jmp_buf, 1);
}
```

7.29.2.13 void* cobsavenv (struct cobjmp_buf *)

Definition at line 646 of file call.c.

```
{
        if (unlikely(!jbuf)) {
                cob_runtime_error ("NULL name parameter passed to 'cobsavenv'");
                cob_stop_run (1);
        }
        if (cobjmp_primed) {
                cob_runtime_error ("Multiple call to 'cobsetjmp'");
                cob_stop_run (1);
        }
        cobjmp_primed = 1;
        return jbuf->cbj_jmp_buf;
}
```

7.29.2.14 void* cobsavenv2 (struct cobjmp_buf *, const int)

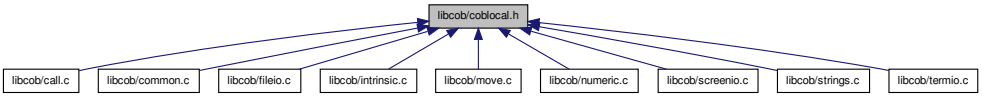
Definition at line 661 of file call.c.

```
{
        int     jtemp;
```

```
        /* Shut up compiler */
        jtemp = jsize;
        return cobsavenv (jbuf);
}
```

7.30 libcob/coblocal.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define COB_HIDDEN
- #define COB_ATTR_INIT(v, w, x, y, z)
- #define COB_CHK_PARMS(x, z)

Functions

- COB_HIDDEN void cob_memcpy (cob_field *, unsigned char *, const int)
- COB_HIDDEN void cob_exit_fileio (void)
- COB_HIDDEN void cob_field_to_string (const cob_field *, char *)
- COB_HIDDEN void cob_init_numeric (void)
- COB_HIDDEN void cob_init_termio (void)
- COB_HIDDEN void cob_init_fileio (void)
- COB_HIDDEN void cob_init_call (void)
- COB_HIDDEN void cob_init_intrinsic (void)
- COB_HIDDEN void cob_init_strings (void)
- COB_HIDDEN void cob_init_move (void)
- COB_HIDDEN void cob_screen_terminate (void)
- COB_HIDDEN void cob_screen_set_mode (size_t)
- COB_HIDDEN int cob_real_get_sign (cob_field *)
- COB_HIDDEN void cob_real_put_sign (cob_field *, const int)
- COB_HIDDEN long long cob_get_long_long (cob_field *)

Variables

- COB_HIDDEN int cob_screen_initialized
- COB_HIDDEN int cob_got_exception
- COB_HIDDEN unsigned int cob_orig_line
- COB_HIDDEN const char * cob_orig_statement
- COB_HIDDEN const char * cob_orig_program_id
- COB_HIDDEN const char * cob_orig_section
- COB_HIDDEN const char * cob_orig_paragraph

7.30.1 Define Documentation

7.30.1.1 #define COB_ATTR_INIT(*v, w, x, y, z*)

Value:

```
do { \
        attr.type = v; \
        attr.digits = w; \
        attr.scale = x; \
        attr.flags = y; \
        attr.pic = z; \
        } while (0)
```

Definition at line 33 of file coblocal.h.

7.30.1.2 #define COB_CHK_PARMS(*x, z*)

Definition at line 50 of file coblocal.h.

7.30.1.3 #define COB_HIDDEN

Definition at line 30 of file coblocal.h.

7.30.2 Function Documentation

7.30.2.1 COB_HIDDEN void cob_exit_fileio (void)

Definition at line 4426 of file fileio.c.

```
{
        struct file_list        *l;

        for (l = file_cache; l; l = l->next) {
                if (l->file->open_mode != COB_OPEN_CLOSED &&
                     l->file->open_mode != COB_OPEN_LOCKED) {
                        cob_field_to_string (l->file->assign, runtime_buffer);
                        cob_close (l->file, 0, NULL);
                        fprintf (stderr, "WARNING - Implicit CLOSE of %s (\"%s\")
```

```
      \n",
                                l->file->select_name, runtime_buffer);
                        fflush (stderr);
                }
        }
#ifdef  USE_DB41
        free (record_lock_object);
        if (bdb_env) {
                bdb_env->lock_id_free (bdb_env, bdb_lock_id);
                bdb_env->close (bdb_env, 0);
        }
#endif
#if     defined(WITH_INDEX_EXTFH) || defined(WITH_SEQRA_EXTFH)
        extfh_cob_exit_fileio ();
#endif
}
```

7.30.2.2 COB_HIDDEN void cob_field_to_string (const cob_field * , char *)

7.30.2.3 COB_HIDDEN long long cob_get_long_long (cob_field *)

Definition at line 1329 of file move.c.

```
{
        long long       n;
        cob_field       temp;
        cob_field_attr  attr;

        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                return cob_display_get_long_long (f);
        case COB_TYPE_NUMERIC_BINARY:
                return cob_binary_mget_int64 (f);
        case COB_TYPE_NUMERIC_PACKED:
                return cob_packed_get_long_long (f);
        default:
                COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0,
                               COB_FLAG_HAVE_SIGN, NULL);
                temp.size = 8;
                temp.data = (unsigned char *)&n;
                temp.attr = &attr;
                cob_move (f, &temp);
                return n;
        }
}
```

7.30.2.4 COB_HIDDEN void cob_init_call (void)

Definition at line 514 of file call.c.

```
{
        char                            *buff;
        char                            *s;
        char                            *p;
        const struct system_table       *psyst;
```

```
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
        lt_dlhandle                     libhandle;
#endif
        size_t                          i;
        struct stat                     st;

#ifndef USE_LIBDL
        lt_dlinit ();
#endif

        /* big enough for anything from libdl/libltdl */
        resolve_error_buff = cob_malloc (CALL_BUFF_SIZE);

#ifndef COB_ALT_HASH
        call_table = cob_malloc (sizeof (struct call_hash *) * HASH_SIZE);
#endif

        call_filename_buff = cob_malloc (CALL_FILEBUFF_SIZE);
        call_entry_buff = cob_malloc (COB_SMALL_BUFF);
        call_entry2_buff = cob_malloc (COB_SMALL_BUFF);
        s = getenv ("COB_LOAD_CASE");
        if (s != NULL) {
                if (strcasecmp (s, "LOWER") == 0) {
                        name_convert = 1;
                } else if (strcasecmp (s, "UPPER") == 0) {
                        name_convert = 2;
                }
        }

        buff = cob_malloc (COB_MEDIUM_BUFF);
        s = getenv ("COB_LIBRARY_PATH");
        if (s == NULL) {
                snprintf (buff, COB_MEDIUM_MAX, ".%s%s",
                          PATHSEPS, COB_LIBRARY_PATH);
        } else {
                snprintf (buff, COB_MEDIUM_MAX, "%s%s.%s%s",
                          s, PATHSEPS, PATHSEPS, COB_LIBRARY_PATH);
        }
        cob_set_library_path (buff);

#ifndef COB_BORKED_DLOPEN
        mainhandle = lt_dlopen (NULL);
#endif

        s = getenv ("COB_PRE_LOAD");
        if (s != NULL) {
                p = cob_strdup (s);
                s = strtok (p, PATHSEPS);
                for (; s; s = strtok (NULL, PATHSEPS)) {
                        for (i = 0; i < resolve_size; ++i) {
                                buff[COB_MEDIUM_MAX] = 0;
                                snprintf (buff, COB_MEDIUM_MAX, "%s/%s.%s",
                                          resolve_path[i], s, COB_MODULE_EXT);
                                if (stat (buff, &st) == 0) {
#if     defined (_WIN32) || !defined (RTLD_DEFAULT)
                                        if ((libhandle = lt_dlopen (buff)) !=
      NULL) {
                                                cache_handle (libhandle);
#else
                                        if (lt_dlopen (buff) != NULL) {
#endif
                                                break;
```

```
                                        }
                                }
                        }
                }
                free (p);
        }
        free (buff);
        call_buffer = cob_malloc (CALL_BUFF_SIZE);
        call_lastsize = CALL_BUFF_SIZE;
        for (psyst = (struct system_table *)&system_tab[0]; psyst->syst_name; ++p
      syst) {
                insert (psyst->syst_name, psyst->syst_call, NULL);
        }
}
```

7.30.2.5 COB_HIDDEN void cob_init_fileio (void)

Definition at line 4379 of file fileio.c.

```
{
        char    *s;
        int     n;

        if ((s = getenv ("COB_SYNC")) != NULL) {
                if (*s == 'Y' || *s == 'y') {
                        cob_do_sync = 1;
                }
                if (*s == 'P' || *s == 'p') {
                        cob_do_sync = 2;
                }
        }
        if ((s = getenv ("COB_SORT_MEMORY")) != NULL) {
                n = atoi (s);
                if (n >= 1024*1024) {
                        cob_sort_memory = n;
                }
        }
        cob_file_path = getenv ("COB_FILE_PATH");
        if (cob_file_path) {
                if (!*cob_file_path || *cob_file_path == ' ') {
                        cob_file_path = NULL;
                }
        }
        cob_ls_nulls = getenv ("COB_LS_NULLS");
        cob_ls_fixed = getenv ("COB_LS_FIXED");

        file_open_env = cob_malloc (COB_SMALL_BUFF);
        file_open_name = cob_malloc (COB_SMALL_BUFF);
        file_open_buff = cob_malloc (COB_SMALL_BUFF);

#ifdef  USE_DB41
        bdb_home = getenv ("DB_HOME");
        join_environment ();
        record_lock_object = cob_malloc (1024);
        bdb_buff = cob_malloc (COB_SMALL_BUFF);
        rlo_size = 1024;
#endif

#if     defined(WITH_INDEX_EXTFH) || defined(WITH_SEQRA_EXTFH)
```

```
        extfh_cob_init_fileio (&sequential_funcs, &lineseq_funcs, &relative_funcs
      , &cob_file_write_opt);
#endif

}
```

7.30.2.6 COB_HIDDEN void cob_init_intrinsic (void)

Definition at line 3269 of file intrinsic.c.

```
{
        size_t          i;

        cob_decimal_init (&d1);
        cob_decimal_init (&d2);
        cob_decimal_init (&d3);
        cob_decimal_init (&d4);
        cob_decimal_init (&d5);
        /* mpz_init2 (mp, 256); */
        memset ((char *)&calc_field[0], 0, sizeof (calc_field));
        memset ((char *)&calc_attr[0], 0, sizeof (calc_attr));
        for (i = 0; i < DEPTH_LEVEL; ++i) {
                calc_field[i].data = cob_malloc (256);
                calc_field[i].size = 256;
                calc_size[i] = 256;
        }
        locale_buff = cob_malloc (COB_SMALL_BUFF);
}
```

7.30.2.7 COB_HIDDEN void cob_init_move (void)

Definition at line 1354 of file move.c.

```
{
        lastdata = cob_malloc (COB_SMALL_BUFF);
        lastsize = COB_SMALL_BUFF;
}
```

7.30.2.8 COB_HIDDEN void cob_init_numeric (void)

Definition at line 1396 of file numeric.c.

```
{
        size_t  i;

        cob_decimal_init (&cob_d1);
        cob_decimal_init (&cob_d2);
        cob_decimal_init (&cob_d3);
        cob_decimal_init (&cob_d4);
        mpz_init2 (cob_mpzt, 256);
        mpz_init2 (cob_mexp, 512);
        for (i = 0; i < COB_MAX_BINARY; i++) {
```

```
                mpz_init (cob_mpze10[i]);
                mpz_ui_pow_ui (cob_mpze10[i], 10, i);
        }
        num_buff_ptr = cob_malloc (2048);
        memset (packed_value, 0, sizeof(packed_value));
}
```

7.30.2.9 COB_HIDDEN void cob_init_strings (void)

Definition at line 595 of file strings.c.

```
{
        inspect_mark = cob_malloc (COB_MEDIUM_BUFF);
        lastsize = COB_MEDIUM_BUFF;
        alpha_attr.type = COB_TYPE_ALPHANUMERIC;
        alpha_attr.digits = 0;
        alpha_attr.scale = 0;
        alpha_attr.flags = 0;
        alpha_attr.pic = NULL;
        alpha_fld.size = 0;
        alpha_fld.data = NULL;
        alpha_fld.attr = &alpha_attr;
}
```

7.30.2.10 COB_HIDDEN void cob_init_termio (void)

Definition at line 279 of file termio.c.

```
{
        term_buff = cob_malloc (COB_MEDIUM_BUFF);
}
```

7.30.2.11 COB_HIDDEN void cob_memcpy (cob_field *, unsigned char *, const int)

7.30.2.12 COB_HIDDEN int cob_real_get_sign (cob_field *)

7.30.2.13 COB_HIDDEN void cob_real_put_sign (cob_field *, const int)

7.30.2.14 COB_HIDDEN void cob_screen_set_mode (size_t)

Definition at line 1058 of file screenio.c.

```
{
        if (!smode) {
                refresh ();
                def_prog_mode ();
                endwin ();
        } else {
                reset_prog_mode ();
                refresh ();
        }
}
```

7.30.2.15 COB_HIDDEN void cob_screen_terminate (void)

Definition at line 297 of file screenio.c.

```
{
        if (cob_screen_initialized) {
                cob_screen_initialized = 0;
                endwin ();
        }
}
```

7.30.3 Variable Documentation

7.30.3.1 COB_HIDDEN int cob_got_exception

Definition at line 166 of file common.c.

7.30.3.2 COB_HIDDEN unsigned int cob_orig_line

Definition at line 172 of file common.c.

7.30.3.3 COB_HIDDEN const char* cob_orig_paragraph

Definition at line 171 of file common.c.

7.30.3.4 COB_HIDDEN const char* cob_orig_program_id

Definition at line 169 of file common.c.

7.30.3.5 COB_HIDDEN const char* cob_orig_section

Definition at line 170 of file common.c.

7.30.3.6 COB_HIDDEN const char* cob_orig_statement

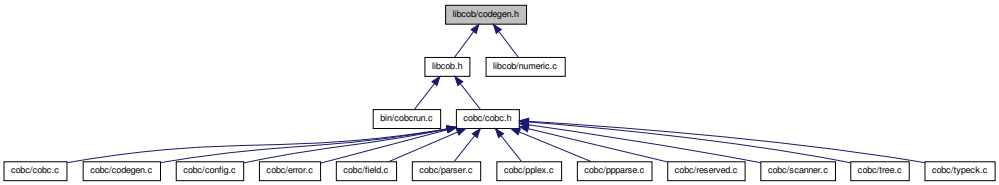
Definition at line 168 of file common.c.

7.30.3.7 COB_HIDDEN int cob_screen_initialized

Definition at line 51 of file screenio.c.

7.31 libcob/codegen.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define COB_STATIC

Functions

- int cob_get_numdisp (const unsigned char *, const size_t)
- int cob_cmp_packed_int (const cob_field *, const int)
- int cob_get_packed_int (const cob_field *)
- int cob_add_packed_int (cob_field *, const int)
- int cob_cmp_u8_binary (const unsigned char *, const int)
- int cob_cmp_s8_binary (const unsigned char *, const int)
- int cob_cmp_u16_binary (const unsigned char *, const int)
- int cob_cmp_s16_binary (const unsigned char *, const int)
- int cob_cmp_u24_binary (const unsigned char *, const int)
- int cob_cmp_s24_binary (const unsigned char *, const int)
- int cob_cmp_u32_binary (const unsigned char *, const int)
- int cob_cmp_s32_binary (const unsigned char *, const int)
- int cob_cmp_u40_binary (const unsigned char *, const int)
- int cob_cmp_s40_binary (const unsigned char *, const int)
- int cob_cmp_u48_binary (const unsigned char *, const int)
- int cob_cmp_s48_binary (const unsigned char *, const int)
- int cob_cmp_u56_binary (const unsigned char *, const int)
- int cob_cmp_s56_binary (const unsigned char *, const int)
- int cob_cmp_u64_binary (const unsigned char *, const int)
- int cob_cmp_s64_binary (const unsigned char *, const int)
- int cob_cmp_align_u16_binary (const unsigned char *, const int)
- int cob_cmp_align_s16_binary (const unsigned char *, const int)
- int cob_cmp_align_u32_binary (const unsigned char *, const int)
- int cob_cmp_align_s32_binary (const unsigned char *, const int)
- int cob_cmp_align_u64_binary (const unsigned char *, const int)
- int cob_cmp_align_s64_binary (const unsigned char *, const int)

- void cob_add_u8_binary (unsigned char *, const int)
- void cob_add_s8_binary (unsigned char *, const int)
- void cob_add_u16_binary (unsigned char *, const int)
- void cob_add_s16_binary (unsigned char *, const int)
- void cob_add_u24_binary (unsigned char *, const int)
- void cob_add_s24_binary (unsigned char *, const int)
- void cob_add_u32_binary (unsigned char *, const int)
- void cob_add_s32_binary (unsigned char *, const int)
- void cob_add_u40_binary (unsigned char *, const int)
- void cob_add_s40_binary (unsigned char *, const int)
- void cob_add_u48_binary (unsigned char *, const int)
- void cob_add_s48_binary (unsigned char *, const int)
- void cob_add_u56_binary (unsigned char *, const int)
- void cob_add_s56_binary (unsigned char *, const int)
- void cob_add_u64_binary (unsigned char *, const int)
- void cob_add_s64_binary (unsigned char *, const int)
- void cob_add_align_u16_binary (unsigned char *, const int)
- void cob_add_align_s16_binary (unsigned char *, const int)
- void cob_add_align_u32_binary (unsigned char *, const int)
- void cob_add_align_s32_binary (unsigned char *, const int)
- void cob_add_align_u64_binary (unsigned char *, const int)
- void cob_add_align_s64_binary (unsigned char *, const int)
- void cob_sub_u8_binary (unsigned char *, const int)
- void cob_sub_s8_binary (unsigned char *, const int)
- void cob_sub_u16_binary (unsigned char *, const int)
- void cob_sub_s16_binary (unsigned char *, const int)
- void cob_sub_u24_binary (unsigned char *, const int)
- void cob_sub_s24_binary (unsigned char *, const int)
- void cob_sub_u32_binary (unsigned char *, const int)
- void cob_sub_s32_binary (unsigned char *, const int)
- void cob_sub_u40_binary (unsigned char *, const int)
- void cob_sub_s40_binary (unsigned char *, const int)
- void cob_sub_u48_binary (unsigned char *, const int)
- void cob_sub_s48_binary (unsigned char *, const int)
- void cob_sub_u56_binary (unsigned char *, const int)
- void cob_sub_s56_binary (unsigned char *, const int)
- void cob_sub_u64_binary (unsigned char *, const int)
- void cob_sub_s64_binary (unsigned char *, const int)
- void cob_sub_align_u16_binary (unsigned char *, const int)
- void cob_sub_align_s16_binary (unsigned char *, const int)
- void cob_sub_align_u32_binary (unsigned char *, const int)
- void cob_sub_align_s32_binary (unsigned char *, const int)
- void cob_sub_align_u64_binary (unsigned char *, const int)
- void cob_sub_align_s64_binary (unsigned char *, const int)
- int cob_cmpswp_u16_binary (const unsigned char *, const int)
- int cob_cmpswp_s16_binary (const unsigned char *, const int)

- int cob_cmpswp_u24_binary (const unsigned char *, const int)
- int cob_cmpswp_s24_binary (const unsigned char *, const int)
- int cob_cmpswp_u32_binary (const unsigned char *, const int)
- int cob_cmpswp_s32_binary (const unsigned char *, const int)
- int cob_cmpswp_u40_binary (const unsigned char *, const int)
- int cob_cmpswp_s40_binary (const unsigned char *, const int)
- int cob_cmpswp_u48_binary (const unsigned char *, const int)
- int cob_cmpswp_s48_binary (const unsigned char *, const int)
- int cob_cmpswp_u56_binary (const unsigned char *, const int)
- int cob_cmpswp_s56_binary (const unsigned char *, const int)
- int cob_cmpswp_u64_binary (const unsigned char *, const int)
- int cob_cmpswp_s64_binary (const unsigned char *, const int)
- int cob_cmpswp_align_u16_binary (const unsigned char *, const int)
- int cob_cmpswp_align_s16_binary (const unsigned char *, const int)
- int cob_cmpswp_align_u32_binary (const unsigned char *, const int)
- int cob_cmpswp_align_s32_binary (const unsigned char *, const int)
- int cob_cmpswp_align_u64_binary (const unsigned char *, const int)
- int cob_cmpswp_align_s64_binary (const unsigned char *, const int)
- void cob_addswp_u16_binary (unsigned char *, const int)
- void cob_addswp_s16_binary (unsigned char *, const int)
- void cob_addswp_u24_binary (unsigned char *, const int)
- void cob_addswp_s24_binary (unsigned char *, const int)
- void cob_addswp_u32_binary (unsigned char *, const int)
- void cob_addswp_s32_binary (unsigned char *, const int)
- void cob_addswp_u40_binary (unsigned char *, const int)
- void cob_addswp_s40_binary (unsigned char *, const int)
- void cob_addswp_u48_binary (unsigned char *, const int)
- void cob_addswp_s48_binary (unsigned char *, const int)
- void cob_addswp_u56_binary (unsigned char *, const int)
- void cob_addswp_s56_binary (unsigned char *, const int)
- void cob_addswp_u64_binary (unsigned char *, const int)
- void cob_addswp_s64_binary (unsigned char *, const int)
- void cob_subswp_u16_binary (unsigned char *, const int)
- void cob_subswp_s16_binary (unsigned char *, const int)
- void cob_subswp_u24_binary (unsigned char *, const int)
- void cob_subswp_s24_binary (unsigned char *, const int)
- void cob_subswp_u32_binary (unsigned char *, const int)
- void cob_subswp_s32_binary (unsigned char *, const int)
- void cob_subswp_u40_binary (unsigned char *, const int)
- void cob_subswp_s40_binary (unsigned char *, const int)
- void cob_subswp_u48_binary (unsigned char *, const int)
- void cob_subswp_s48_binary (unsigned char *, const int)
- void cob_subswp_u56_binary (unsigned char *, const int)
- void cob_subswp_s56_binary (unsigned char *, const int)
- void cob_subswp_u64_binary (unsigned char *, const int)
- void cob_subswp_s64_binary (unsigned char *, const int)

- void cob_setswp_u16_binary (unsigned char *, const int)
- void cob_setswp_s16_binary (unsigned char *, const int)
- void cob_setswp_u24_binary (unsigned char *, const int)
- void cob_setswp_s24_binary (unsigned char *, const int)
- void cob_setswp_u32_binary (unsigned char *, const int)
- void cob_setswp_s32_binary (unsigned char *, const int)
- void cob_setswp_u40_binary (unsigned char *, const int)
- void cob_setswp_s40_binary (unsigned char *, const int)
- void cob_setswp_u48_binary (unsigned char *, const int)
- void cob_setswp_s48_binary (unsigned char *, const int)
- void cob_setswp_u56_binary (unsigned char *, const int)
- void cob_setswp_s56_binary (unsigned char *, const int)
- void cob_setswp_u64_binary (unsigned char *, const int)
- void cob_setswp_s64_binary (unsigned char *, const int)

7.31.1 Define Documentation

7.31.1.1 #define COB_STATIC

Definition at line 26 of file codegen.h.

7.31.2 Function Documentation

7.31.2.1 void cob_add_align_s16_binary (unsigned char * , const int)

7.31.2.2 void cob_add_align_s32_binary (unsigned char * , const int)

7.31.2.3 void cob_add_align_s64_binary (unsigned char * , const int)

7.31.2.4 void cob_add_align_u16_binary (unsigned char * , const int)

7.31.2.5 void cob_add_align_u32_binary (unsigned char * , const int)

7.31.2.6 void cob_add_align_u64_binary (unsigned char * , const int)

7.31.2.7 int cob_add_packed_int (cob_field * , const int)

7.31.2.8 void cob_add_s16_binary (unsigned char * , const int)

7.31.2.9 void cob_add_s24_binary (unsigned char * , const int)

7.31.2.10 void cob_add_s32_binary (unsigned char * , const int)

7.31.2.11 void cob_add_s40_binary (unsigned char * , const int)

7.31.2.12 void cob_add_s48_binary (unsigned char * , const int)

7.31.2.13 void cob_add_s56_binary (unsigned char * , const int)

7.31.2.14 void cob_add_s64_binary (unsigned char * , const int)

7.31.2.15 void cob_add_s8_binary (unsigned char * , const int)

7.31.2.16 void cob_add_u16_binary (unsigned char * , const int)

7.31.2.17 void cob_add_u24_binary (unsigned char * , const int)

7.31.2.18 void cob_add_u32_binary (unsigned char * , const int)

7.31.2.19 void cob_add_u40_binary (unsigned char * , const int)

7.31.2.20 void cob_add_u48_binary (unsigned char * , const int)

7.31.2.21 void cob_add_u56_binary (unsigned char * , const int)

7.31.2.22 void cob_add_u64_binary (unsigned char * , const int)

7.31.2.23 void cob_add_u8_binary (unsigned char * , const int)

7.31.2.24 void cob_addswp_s16_binary (unsigned char * , const int)

7.31.2.25 void cob_addswp_s24_binary (unsigned char * , const int)

7.31.2.26 void cob_addswp_s32_binary (unsigned char * , const int)

7.31.2.27 void cob_addswp_s40_binary (unsigned char * , const int)

7.31.2.28 void cob_addswp_s48_binary (unsigned char * , const int)

7.31.2.29 void cob_addswp_s56_binary (unsigned char * , const int)

7.31.2.30 void cob_addswp_s64_binary (unsigned char * , const int)

7.31.2.31 void cob_addswp_u16_binary (unsigned char * , const int)

7.31.2.32 void cob_addswp_u24_binary (unsigned char * , const int)

7.31.2.33 void cob_addswp_u32_binary (unsigned char * , const int)

7.31.2.34 void cob_addswp_u40_binary (unsigned char * , const int)

7.31.2.35 void cob_addswp_u48_binary (unsigned char * , const int)

7.31.2.36 void cob_addswp_u56_binary (unsigned char *, const int)

7.31.2.37 void cob_addswp_u64_binary (unsigned char *, const int)

7.31.2.38 int cob_cmp_align_s16_binary (const unsigned char *, const int)

7.31.2.39 int cob_cmp_align_s32_binary (const unsigned char *, const int)

7.31.2.40 int cob_cmp_align_s64_binary (const unsigned char *, const int)

7.31.2.41 int cob_cmp_align_u16_binary (const unsigned char *, const int)

7.31.2.42 int cob_cmp_align_u32_binary (const unsigned char *, const int)

7.31.2.43 int cob_cmp_align_u64_binary (const unsigned char *, const int)

7.31.2.44 int cob_cmp_packed_int (const cob_field *, const int)

7.31.2.45 int cob_cmp_s16_binary (const unsigned char *, const int)

7.31.2.46 int cob_cmp_s24_binary (const unsigned char *, const int)

7.31.2.47 int cob_cmp_s32_binary (const unsigned char *, const int)

7.31.2.48 int cob_cmp_s40_binary (const unsigned char *, const int)

7.31.2.49 int cob_cmp_s48_binary (const unsigned char *, const int)

7.31.2.50 int cob_cmp_s56_binary (const unsigned char *, const int)

7.31.2.51 int cob_cmp_s64_binary (const unsigned char *, const int)

7.31.2.52 int cob_cmp_s8_binary (const unsigned char *, const int)

7.31.2.53 int cob_cmp_u16_binary (const unsigned char *, const int)

7.31.2.54 int cob_cmp_u24_binary (const unsigned char *, const int)

7.31.2.55 int cob_cmp_u32_binary (const unsigned char *, const int)

7.31.2.56 int cob_cmp_u40_binary (const unsigned char *, const int)

7.31.2.57 int cob_cmp_u48_binary (const unsigned char *, const int)

7.31.2.58 int cob_cmp_u56_binary (const unsigned char *, const int)

7.31.2.59 int cob_cmp_u64_binary (const unsigned char *, const int)

7.31.2.60 int cob_cmp_u8_binary (const unsigned char *, const int)

7.31.2.61 int cob_cmpswp_align_s16_binary (const unsigned char *, const int)

7.31.2.62 int cob_cmpswp_align_s32_binary (const unsigned char *, const int)

7.31.2.63 int cob_cmpswp_align_s64_binary (const unsigned char *, const int)

7.31.2.64 int cob_cmpswp_align_u16_binary (const unsigned char *, const int)

7.31.2.65 int cob_cmpswp_align_u32_binary (const unsigned char *, const int)

7.31.2.66 int cob_cmpswp_align_u64_binary (const unsigned char *, const int)

7.31.2.67 int cob_cmpswp_s16_binary (const unsigned char *, const int)

7.31.2.68 int cob_cmpswp_s24_binary (const unsigned char *, const int)

7.31.2.69 int cob_cmpswp_s32_binary (const unsigned char *, const int)

7.31.2.70 int cob_cmpswp_s40_binary (const unsigned char *, const int)

7.31.2.71 int cob_cmpswp_s48_binary (const unsigned char *, const int)

7.31.2.72 int cob_cmpswp_s56_binary (const unsigned char *, const int)

7.31.2.73 int cob_cmpswp_s64_binary (const unsigned char *, const int)

7.31.2.74 int cob_cmpswp_u16_binary (const unsigned char *, const int)

7.31.2.75 int cob_cmpswp_u24_binary (const unsigned char *, const int)

7.31.2.76 int cob_cmpswp_u32_binary (const unsigned char *, const int)

7.31.2.77 int cob_cmpswp_u40_binary (const unsigned char *, const int)

7.31.2.78 int cob_cmpswp_u48_binary (const unsigned char *, const int)

7.31.2.79 int cob_cmpswp_u56_binary (const unsigned char *, const int)

7.31.2.80 int cob_cmpswp_u64_binary (const unsigned char *, const int)

7.31.2.81 int cob_get_numdisp (const unsigned char *, const size_t)

7.31.2.82 int cob_get_packed_int (const cob_field *)

7.31.2.83 void cob_setswp_s16_binary (unsigned char *, const int)

7.31.2.84 void cob_setswp_s24_binary (unsigned char *, const int)

7.31.2.85 void cob_setswp_s32_binary (unsigned char *, const int)

7.31.2.86 void cob_setswp_s40_binary (unsigned char *, const int)

7.31.2.87 void cob_setswp_s48_binary (unsigned char *, const int)

7.31.2.88 void cob_setswp_s56_binary (unsigned char *, const int)

7.31.2.89 void cob_setswp_s64_binary (unsigned char *, const int)

7.31.2.90 void cob_setswp_u16_binary (unsigned char *, const int)

7.31.2.91 void cob_setswp_u24_binary (unsigned char *, const int)

7.31.2.92 void cob_setswp_u32_binary (unsigned char *, const int)

7.31.2.93 void cob_setswp_u40_binary (unsigned char *, const int)

7.31.2.94 void cob_setswp_u48_binary (unsigned char *, const int)

7.31.2.95 void cob_setswp_u56_binary (unsigned char *, const int)

7.31.2.96 void cob_setswp_u64_binary (unsigned char *, const int)

7.31.2.97 void cob_sub_align_s16_binary (unsigned char *, const int)

7.31.2.98 void cob_sub_align_s32_binary (unsigned char *, const int)

7.31.2.99 void cob_sub_align_s64_binary (unsigned char *, const int)

7.31.2.100 void cob_sub_align_u16_binary (unsigned char *, const int)

7.31.2.101 void cob_sub_align_u32_binary (unsigned char *, const int)

7.31.2.102 void cob_sub_align_u64_binary (unsigned char *, const int)

7.31.2.103 void cob_sub_s16_binary (unsigned char *, const int)

7.31.2.104 void cob_sub_s24_binary (unsigned char *, const int)

7.31.2.105 void cob_sub_s32_binary (unsigned char *, const int)

7.31.2.106 void cob_sub_s40_binary (unsigned char *, const int)

7.31.2.107 void cob_sub_s48_binary (unsigned char *, const int)

7.31.2.108 void cob_sub_s56_binary (unsigned char * , const int)

7.31.2.109 void cob_sub_s64_binary (unsigned char * , const int)

7.31.2.110 void cob_sub_s8_binary (unsigned char * , const int)

7.31.2.111 void cob_sub_u16_binary (unsigned char * , const int)

7.31.2.112 void cob_sub_u24_binary (unsigned char * , const int)

7.31.2.113 void cob_sub_u32_binary (unsigned char * , const int)

7.31.2.114 void cob_sub_u40_binary (unsigned char * , const int)

7.31.2.115 void cob_sub_u48_binary (unsigned char * , const int)

7.31.2.116 void cob_sub_u56_binary (unsigned char * , const int)

7.31.2.117 void cob_sub_u64_binary (unsigned char * , const int)

7.31.2.118 void cob_sub_u8_binary (unsigned char * , const int)

7.31.2.119 void cob_subswp_s16_binary (unsigned char * , const int)

7.31.2.120 void cob_subswp_s24_binary (unsigned char * , const int)

7.31.2.121 void cob_subswp_s32_binary (unsigned char * , const int)

7.31.2.122 void cob_subswp_s40_binary (unsigned char * , const int)

7.31.2.123 void cob_subswp_s48_binary (unsigned char * , const int)

7.31.2.124 void cob_subswp_s56_binary (unsigned char * , const int)

7.31.2.125 void cob_subswp_s64_binary (unsigned char * , const int)

7.31.2.126 void cob_subswp_u16_binary (unsigned char * , const int)

7.31.2.127 void cob_subswp_u24_binary (unsigned char * , const int)

7.31.2.128 void cob_subswp_u32_binary (unsigned char * , const int)

7.31.2.129 void cob_subswp_u40_binary (unsigned char * , const int)

7.31.2.130 void cob_subswp_u48_binary (unsigned char * , const int)

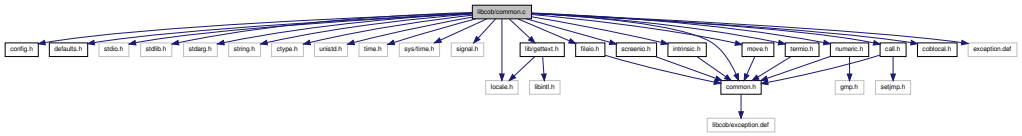
7.31.2.131 void cob_subswp_u56_binary (unsigned char * , const int)

7.31.2.132 void cob_subswp_u64_binary (unsigned char *, const int)

7.32 libcob/common.c File Reference

```
#include "config.h"
#include "defaults.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <signal.h>
#include <locale.h>
#include "common.h"
#include "coblocal.h"
#include "move.h"
#include "numeric.h"
#include "termio.h"
#include "fileio.h"
#include "call.h"
#include "screenio.h"
#include "intrinsic.h"
#include "lib/gettext.h"
#include "exception.def"
```

Include dependency graph for common.c:



Classes

- struct cob_exception
- struct cob_alloc_cache
- struct **exit_handlerlist**
- struct **handlerlist**

Defines

- #define COB_ERRBUF_SIZE 256
- #define COB_EXCEPTION(code, tag, name, critical) name,
- #define COB_EXCEPTION(code, tag, name, critical) 0x##code,
- #define EXCEPTION_TAB_SIZE sizeof(cob_exception_tab_code) / sizeof(int)

Typedefs

- typedef void(* cob_sighandler_t)(int)

Variables

- struct cob_module * cob_current_module = NULL
- int cob_initialized = 0
- int cob_exception_code = 0
- int cob_call_params = 0
- int cob_save_call_params = 0
- int cob_initial_external = 0
- int cob_got_exception = 0
- const char * cob_orig_statement = NULL
- const char * cob_orig_program_id = NULL
- const char * cob_orig_section = NULL
- const char * cob_orig_paragraph = NULL
- unsigned int cob_orig_line = 0
- cob_field cob_zero = { 1, (ucharptr)"0", &all_attr }
- cob_field cob_space = { 1, (ucharptr)" ", &all_attr }
- cob_field cob_high = { 1, (ucharptr)"\xff", &all_attr }
- cob_field cob_low = { 1, (ucharptr)"\0", &all_attr }
- cob_field cob_quote = { 1, (ucharptr)"\"", &all_attr }
- cob_field cob_one = { 1, (ucharptr)"1", &one_attr }

7.32.1 Define Documentation

7.32.1.1 #define COB_ERRBUF_SIZE 256

Definition at line 75 of file common.c.

7.32.1.2 #define COB_EXCEPTION(*code, tag,* **name**, *critical*) **name**,

Definition at line 122 of file common.c.

7.32.1.3 #define COB_EXCEPTION(*code, tag,* **name**, *critical*) 0x##code,

Definition at line 122 of file common.c.

7.32.1.4 #define EXCEPTION_TAB_SIZE sizeof(cob_exception_tab_code) / sizeof(int)

Definition at line 131 of file common.c.

7.32.2 Typedef Documentation

7.32.2.1 typedef void(* cob_sighandler_t)(int)

Definition at line 103 of file common.c.

7.32.3 Variable Documentation

7.32.3.1 int cob_call_params = 0

Definition at line 163 of file common.c.

7.32.3.2 struct cob_module* cob_current_module = NULL

Definition at line 158 of file common.c.

7.32.3.3 int cob_exception_code = 0

Definition at line 161 of file common.c.

7.32.3.4 int cob_got_exception = 0

Definition at line 166 of file common.c.

7.32.3.5 cob_field cob_high = { 1, (ucharptr)"\xff", &all_attr }

Definition at line 176 of file common.c.

7.32.3.6 int cob_initial_external = 0

Definition at line 165 of file common.c.

7.32.3.7 int cob_initialized = 0

Definition at line 160 of file common.c.

7.32.3.8 cob_field cob_low = { 1, (ucharptr)"\0", &all_attr }

Definition at line 177 of file common.c.

7.32.3.9 cob_field cob_one = { 1, (ucharptr)"1", &one_attr }

Definition at line 179 of file common.c.

7.32.3.10 unsigned int cob_orig_line = 0

Definition at line 172 of file common.c.

7.32.3.11 const char* cob_orig_paragraph = NULL

Definition at line 171 of file common.c.

7.32.3.12 const char* cob_orig_program_id = NULL

Definition at line 169 of file common.c.

7.32.3.13 const char* cob_orig_section = NULL

Definition at line 170 of file common.c.

7.32.3.14 const char* cob_orig_statement = NULL

Definition at line 168 of file common.c.

7.32.3.15 cob_field cob_quote = { 1, (ucharptr)"\"", &all_attr }

Definition at line 178 of file common.c.

7.32.3.16 int cob_save_call_params = 0

Definition at line 164 of file common.c.

7.32.3.17 cob_field cob_space = { 1, (ucharptr)" ", &all_attr }

Definition at line 175 of file common.c.

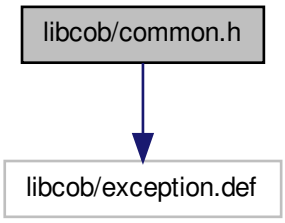
7.32.3.18 cob_field cob_zero = { 1, (ucharptr)"0", &all_attr }

Definition at line 174 of file common.c.

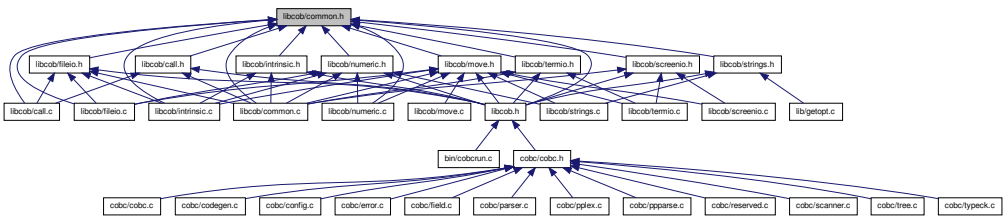
7.33 libcob/common.h File Reference

```
#include <libcob/exception.def>
```

Include dependency graph for common.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct cob_external
- struct cob_field_attr
- struct cob_field
- struct cob_module

Defines

- #define DLL_EXPIMP
- #define COB_INLINE
- #define likely(x) (x)
- #define unlikely(x) (x)
- #define COB_NOINLINE
- #define COB_MINI_BUFF 256
- #define COB_SMALL_BUFF 1024
- #define COB_NORMAL_BUFF 2048
- #define COB_MEDIUM_BUFF 8192
- #define COB_LARGE_BUFF 16384
- #define COB_MINI_MAX (COB_MINI_BUFF - 1)
- #define COB_SMALL_MAX (COB_SMALL_BUFF - 1)
- #define COB_NORMAL_MAX (COB_NORMAL_BUFF - 1)
- #define COB_MEDIUM_MAX (COB_MEDIUM_BUFF - 1)
- #define COB_LARGE_MAX (COB_LARGE_BUFF - 1)
- #define COB_STACK_SIZE 255
- #define COB_MAX_FIELD_PARAMS 64
- #define COB_TYPE_UNKNOWN 0x00
- #define COB_TYPE_GROUP 0x01
- #define COB_TYPE_BOOLEAN 0x02
- #define COB_TYPE_NUMERIC 0x10
- #define COB_TYPE_NUMERIC_DISPLAY 0x10
- #define COB_TYPE_NUMERIC_BINARY 0x11
- #define COB_TYPE_NUMERIC_PACKED 0x12
- #define COB_TYPE_NUMERIC_FLOAT 0x13
- #define COB_TYPE_NUMERIC_DOUBLE 0x14
- #define COB_TYPE_NUMERIC_EDITED 0x24
- #define COB_TYPE_ALPHANUMERIC 0x21
- #define COB_TYPE_ALPHANUMERIC_ALL 0x22
- #define COB_TYPE_ALPHANUMERIC_EDITED 0x23
- #define COB_TYPE_NATIONAL 0x40
- #define COB_TYPE_NATIONAL_EDITED 0x41
- #define COB_FLAG_HAVE_SIGN 0x01
- #define COB_FLAG_SIGN_SEPARATE 0x02
- #define COB_FLAG_SIGN_LEADING 0x04
- #define COB_FLAG_BLANK_ZERO 0x08
- #define COB_FLAG_JUSTIFIED 0x10

- #define COB_FLAG_BINARY_SWAP 0x20
- #define COB_FLAG_REAL_BINARY 0x40
- #define COB_FLAG_IS_POINTER 0x80
- #define COB_FIELD_HAVE_SIGN(f) ((f)->attr->flags & COB_FLAG_HAVE_SIGN)
- #define COB_FIELD_SIGN_SEPARATE(f) ((f)->attr->flags & COB_FLAG_SIGN_SEPARATE)
- #define COB_FIELD_SIGN_LEADING(f) ((f)->attr->flags & COB_FLAG_SIGN_LEADING)
- #define COB_FIELD_BLANK_ZERO(f) ((f)->attr->flags & COB_FLAG_BLANK_ZERO)
- #define COB_FIELD_JUSTIFIED(f) ((f)->attr->flags & COB_FLAG_JUSTIFIED)
- #define COB_FIELD_BINARY_SWAP(f) ((f)->attr->flags & COB_FLAG_BINARY_SWAP)
- #define COB_FIELD_REAL_BINARY(f) ((f)->attr->flags & COB_FLAG_REAL_BINARY)
- #define COB_FIELD_IS_POINTER(f) ((f)->attr->flags & COB_FLAG_IS_POINTER)
- #define cob_get_sign(f) (COB_FIELD_HAVE_SIGN (f) ? cob_real_get_sign (f) : 0)
- #define cob_put_sign(f, s) if (COB_FIELD_HAVE_SIGN (f)) cob_real_put_sign (f, s)
- #define COB_FIELD_TYPE(f) ((f)->attr->type)
- #define COB_FIELD_DIGITS(f) ((f)->attr->digits)
- #define COB_FIELD_SCALE(f) ((f)->attr->scale)
- #define COB_FIELD_FLAGS(f) ((f)->attr->flags)
- #define COB_FIELD_PIC(f) ((f)->attr->pic)
- #define COB_FIELD_DATA(f)
- #define COB_FIELD_SIZE(f) ((f)->size - (COB_FIELD_SIGN_SEPARATE (f) ? 1 : 0))
- #define COB_FIELD_IS_NUMERIC(f) (COB_FIELD_TYPE (f) & COB_TYPE_NUMERIC)
- #define GET_SIGN_ASCII(x) x -= 0x40
- #define PUT_SIGN_ASCII(x) x += 0x40
- #define COB_DISPLAY_SIGN_ASCII 0
- #define COB_DISPLAY_SIGN_EBCDIC 1
- #define COB_EXCEPTION(code, tag, name, critical) tag,
- #define COB_FERROR_INITIALIZED 0
- #define COB_FERROR_CODEGEN 1
- #define COB_FERROR_CHAINING 2
- #define COB_FERROR_STACK 3
- #define cob_d2i(x) ((x) - '0')
- #define cob_i2d(x) ((x) + '0')

Typedefs

- typedef unsigned char * ucharptr

Enumerations

- enum cob_exception_id { COB_EC_ZERO, COB_EC_MAX }

Functions

- void cob_init (int, char **)
- void cob_module_enter (struct cob_module *)
- void cob_module_leave (struct cob_module *)
- void cobexit (int)
- void cob_stop_run (const int)
- void cob_fatal_error (const unsigned int)
- void cob_runtime_error (const char *,...)
- void * cob_malloc (const size_t)
- const char * cob_get_exception_name (const int)
- void cob_set_exception (const int)
- void cob_check_version (const char *, const char *, const int)
- void cob_accept_date (cob_field *)
- void cob_accept_date_yyyymmdd (cob_field *)
- void cob_accept_day (cob_field *)
- void cob_accept_day_yyyyddd (cob_field *)
- void cob_accept_day_of_week (cob_field *)
- void cob_accept_time (cob_field *)
- void cob_display_command_line (cob_field *)
- void cob_accept_command_line (cob_field *)
- void cob_set_environment (cob_field *, cob_field *)
- void cob_display_environment (cob_field *)
- void cob_get_environment (cob_field *, cob_field *)
- void cob_accept_environment (cob_field *)
- void cob_display_env_value (cob_field *)
- void cob_display_arg_number (cob_field *)
- void cob_accept_arg_number (cob_field *)
- void cob_accept_arg_value (cob_field *)
- void cob_chain_setup (void *, const size_t, const size_t)
- void cob_allocate (unsigned char **, cob_field *, cob_field *)
- void cob_free_alloc (unsigned char **, unsigned char *)
- int cobinit (void)
- int cobtidy (void)
- void * cobcommandline (int, int *, char ***, char ***, char **)
- char * cobgetenv (const char *)
- int cobputenv (char *)
- int CBL_ERROR_PROC (unsigned char *, unsigned char *)
- int CBL_EXIT_PROC (unsigned char *, unsigned char *)
- int SYSTEM (const unsigned char *)
- int CBL_AND (unsigned char *, unsigned char *, const int)
- int CBL_OR (unsigned char *, unsigned char *, const int)

- int CBL_NOR (unsigned char *, unsigned char *, const int)
- int CBL_XOR (unsigned char *, unsigned char *, const int)
- int CBL_IMP (unsigned char *, unsigned char *, const int)
- int CBL_NIMP (unsigned char *, unsigned char *, const int)
- int CBL_EQ (unsigned char *, unsigned char *, const int)
- int CBL_NOT (unsigned char *, const int)
- int CBL_XF4 (unsigned char *, unsigned char *)
- int CBL_XF5 (unsigned char *, unsigned char *)
- int CBL_X91 (unsigned char *, const unsigned char *, unsigned char *)
- int CBL_TOUPPER (unsigned char *, const int)
- int CBL_TOLOWER (unsigned char *, const int)
- int CBL_OC_NANOSLEEP (unsigned char *)
- int cob_return_args (unsigned char *)
- int cob_parameter_size (unsigned char *)
- int cob_acuw_sleep (unsigned char *)
- int cob_acuw_justify (unsigned char *,...)
- unsigned char * cob_external_addr (const char *, const int)
- unsigned char * cob_get_pointer (const unsigned char *)
- void * cob_get_prog_pointer (const unsigned char *)
- void cob_set_location (const char *, const char *, const unsigned int, const char *, const char *, const char *)
- void cob_ready_trace (void)
- void cob_reset_trace (void)
- int cob_get_switch (const int)
- void cob_set_switch (const int, const int)
- int cob_cmp (cob_field *, cob_field *)
- int cob_is_omitted (const cob_field *)
- int cob_is_numeric (cob_field *)
- int cob_is_alpha (const cob_field *)
- int cob_is_upper (const cob_field *)
- int cob_is_lower (const cob_field *)
- void cob_table_sort_init (const int, const unsigned char *)
- void cob_table_sort_init_key (const int, cob_field *, size_t)
- void cob_table_sort (cob_field *, const int)
- void cob_check_numeric (cob_field *, const char *)
- void cob_check_based (const unsigned char *, const char *)
- void cob_check_odo (const int, const int, const int, const char *)
- void cob_check_subscript (const int, const int, const int, const char *)
- void cob_check_ref_mod (const int, const int, const int, const char *)
- int cob_numeric_cmp (cob_field *, cob_field *)

Variables

- DLL_EXPIMP int cob_initialized
- DLL_EXPIMP int cob_exception_code
- DLL_EXPIMP struct cob_module * cob_current_module
- DLL_EXPIMP int cob_call_params
- DLL_EXPIMP int cob_save_call_params
- DLL_EXPIMP int cob_initial_external
- DLL_EXPIMP cob_field cob_zero
- DLL_EXPIMP cob_field cob_space
- DLL_EXPIMP cob_field cob_high
- DLL_EXPIMP cob_field cob_low
- DLL_EXPIMP cob_field cob_quote
- DLL_EXPIMP cob_field cob_one

7.33.1 Define Documentation

7.33.1.1 #define cob_d2i(*x*) ((x) - '0')

Definition at line 263 of file common.h.

7.33.1.2 #define COB_DISPLAY_SIGN_ASCII 0

Definition at line 194 of file common.h.

7.33.1.3 #define COB_DISPLAY_SIGN_EBCDIC 1

Definition at line 195 of file common.h.

7.33.1.4 #define COB_EXCEPTION(*code, tag,* name, *critical*) tag,

Definition at line 223 of file common.h.

7.33.1.5 #define COB_FERROR_CHAINING 2

Definition at line 239 of file common.h.

7.33.1.6 #define COB_FERROR_CODEGEN 1

Definition at line 238 of file common.h.

7.33.1.7 #define COB_FERROR_INITIALIZED 0

Definition at line 237 of file common.h.

7.33.1.8 #define COB_FERROR_STACK 3

Definition at line 240 of file common.h.

7.33.1.9 #define COB_FIELD_BINARY_SWAP(f) ((f)->attr->flags & COB_FLAG_BINARY_SWAP)

Definition at line 146 of file common.h.

7.33.1.10 #define COB_FIELD_BLANK_ZERO(f) ((f)->attr->flags & COB_FLAG_BLANK_ZERO)

Definition at line 144 of file common.h.

7.33.1.11 #define COB_FIELD_DATA(f)

Value:

```
((f)->data +                                                            \
   ((COB_FIELD_SIGN_SEPARATE (f) && COB_FIELD_SIGN_LEADING (f)) ? 1 : 0))
```

Definition at line 176 of file common.h.

7.33.1.12 #define COB_FIELD_DIGITS(f) ((f)->attr->digits)

Definition at line 172 of file common.h.

7.33.1.13 #define COB_FIELD_FLAGS(f) ((f)->attr->flags)

Definition at line 174 of file common.h.

7.33.1.14 #define COB_FIELD_HAVE_SIGN(f) ((f)->attr->flags & COB_FLAG_HAVE_SIGN)

Definition at line 141 of file common.h.

7.33.1.15 #define COB_FIELD_IS_NUMERIC(f) (COB_FIELD_TYPE (f) & COB_TYPE_NUMERIC)

Definition at line 182 of file common.h.

7.33.1.16 #define COB_FIELD_IS_POINTER(f) ((f)->attr->flags & COB_FLAG_IS_POINTER)

Definition at line 148 of file common.h.

7.33.1.17 #define COB_FIELD_JUSTIFIED(f) ((f)->attr->flags & COB_FLAG_JUSTIFIED)

Definition at line 145 of file common.h.

7.33.1.18 #define COB_FIELD_PIC(f) ((f)->attr->pic)

Definition at line 175 of file common.h.

7.33.1.19 #define COB_FIELD_REAL_BINARY(f) ((f)->attr->flags & COB_FLAG_REAL_BINARY)

Definition at line 147 of file common.h.

7.33.1.20 #define COB_FIELD_SCALE(f) ((f)->attr->scale)

Definition at line 173 of file common.h.

7.33.1.21 #define COB_FIELD_SIGN_LEADING(f) ((f)->attr->flags & COB_FLAG_SIGN_LEADING)

Definition at line 143 of file common.h.

7.33.1.22 #define COB_FIELD_SIGN_SEPARATE(f) ((f)->attr->flags & COB_FLAG_SIGN_SEPARATE)

Definition at line 142 of file common.h.

7.33.1.23 #define COB_FIELD_SIZE(f) ((f)->size - (COB_FIELD_SIGN_SEPARATE (f) ? 1 : 0))

Definition at line 179 of file common.h.

7.33.1.24 #define COB_FIELD_TYPE(f) ((f)->attr->type)

Definition at line 171 of file common.h.

7.33.1.25 #define COB_FLAG_BINARY_SWAP 0x20

Definition at line 137 of file common.h.

7.33.1.26 #define COB_FLAG_BLANK_ZERO 0x08

Definition at line 135 of file common.h.

7.33.1.27 #define COB_FLAG_HAVE_SIGN 0x01

Definition at line 132 of file common.h.

7.33.1.28 #define COB_FLAG_IS_POINTER 0x80

Definition at line 139 of file common.h.

7.33.1.29 #define COB_FLAG_JUSTIFIED 0x10

Definition at line 136 of file common.h.

7.33.1.30 #define COB_FLAG_REAL_BINARY 0x40

Definition at line 138 of file common.h.

7.33.1.31 #define COB_FLAG_SIGN_LEADING 0x04

Definition at line 134 of file common.h.

7.33.1.32 #define COB_FLAG_SIGN_SEPARATE 0x02

Definition at line 133 of file common.h.

7.33.1.33 #define cob_get_sign(*f*) (COB_FIELD_HAVE_SIGN (f) ? cob_real_get_sign (f) : 0)

Definition at line 150 of file common.h.

7.33.1.34 #define cob_i2d(*x*) ((x) + '0')

Definition at line 266 of file common.h.

7.33.1.35 #define COB_INLINE

Definition at line 54 of file common.h.

7.33.1.36 #define COB_LARGE_BUFF 16384

Definition at line 83 of file common.h.

7.33.1.37 #define COB_LARGE_MAX (COB_LARGE_BUFF - 1)

Definition at line 88 of file common.h.

7.33.1.38 #define COB_MAX_FIELD_PARAMS 64

Definition at line 92 of file common.h.

7.33.1.39 #define COB_MEDIUM_BUFF 8192

Definition at line 82 of file common.h.

7.33.1.40 #define COB_MEDIUM_MAX (COB_MEDIUM_BUFF - 1)

Definition at line 87 of file common.h.

7.33.1.41 #define COB_MINI_BUFF 256

Definition at line 79 of file common.h.

7.33.1.42 #define COB_MINI_MAX (COB_MINI_BUFF - 1)

Definition at line 84 of file common.h.

7.33.1.43 #define COB_NOINLINE

Definition at line 70 of file common.h.

7.33.1.44 #define COB_NORMAL_BUFF 2048

Definition at line 81 of file common.h.

7.33.1.45 #define COB_NORMAL_MAX (COB_NORMAL_BUFF - 1)

Definition at line 86 of file common.h.

7.33.1.46 #define cob_put_sign(*f, s*) if (COB_FIELD_HAVE_SIGN (f)) cob_real_put_sign (f, s)

Definition at line 151 of file common.h.

7.33.1.47 #define COB_SMALL_BUFF 1024

Definition at line 80 of file common.h.

7.33.1.48 #define COB_SMALL_MAX (COB_SMALL_BUFF - 1)

Definition at line 85 of file common.h.

7.33.1.49 #define COB_STACK_SIZE 255

Definition at line 90 of file common.h.

7.33.1.50 #define COB_TYPE_ALPHANUMERIC 0x21

Definition at line 123 of file common.h.

7.33.1.51 #define COB_TYPE_ALPHANUMERIC_ALL 0x22

Definition at line 124 of file common.h.

7.33.1.52 #define COB_TYPE_ALPHANUMERIC_EDITED 0x23

Definition at line 125 of file common.h.

7.33.1.53 #define COB_TYPE_BOOLEAN 0x02

Definition at line 113 of file common.h.

7.33.1.54 #define COB_TYPE_GROUP 0x01

Definition at line 112 of file common.h.

7.33.1.55 #define COB_TYPE_NATIONAL 0x40

Definition at line 127 of file common.h.

7.33.1.56 #define COB_TYPE_NATIONAL_EDITED 0x41

Definition at line 128 of file common.h.

7.33.1.57 #define COB_TYPE_NUMERIC 0x10

Definition at line 115 of file common.h.

7.33.1.58 #define COB_TYPE_NUMERIC_BINARY 0x11

Definition at line 117 of file common.h.

7.33.1.59 #define COB_TYPE_NUMERIC_DISPLAY 0x10

Definition at line 116 of file common.h.

7.33.1.60 #define COB_TYPE_NUMERIC_DOUBLE 0x14

Definition at line 120 of file common.h.

7.33.1.61 #define COB_TYPE_NUMERIC_EDITED 0x24

Definition at line 121 of file common.h.

7.33.1.62 #define COB_TYPE_NUMERIC_FLOAT 0x13

Definition at line 119 of file common.h.

7.33.1.63 #define COB_TYPE_NUMERIC_PACKED 0x12

Definition at line 118 of file common.h.

7.33.1.64 #define COB_TYPE_UNKNOWN 0x00

Definition at line 111 of file common.h.

7.33.1.65 #define DLL_EXPIMP

Definition at line 46 of file common.h.

7.33.1.66 #define GET_SIGN_ASCII(*x*) x -= 0x40

Definition at line 191 of file common.h.

7.33.1.67 #define likely(*x*) (x)

Definition at line 68 of file common.h.

7.33.1.68 #define PUT_SIGN_ASCII(x) x += 0x40

Definition at line 192 of file common.h.

7.33.1.69 #define unlikely(x) (x)

Definition at line 69 of file common.h.

7.33.2 Typedef Documentation

7.33.2.1 typedef unsigned char* ucharptr

Definition at line 77 of file common.h.

7.33.3 Enumeration Type Documentation

7.33.3.1 enum cob_exception_id

Enumerator:

COB_EC_ZERO

COB_EC_MAX

Definition at line 225 of file common.h.

```
                        {
        COB_EC_ZERO,
#include <libcob/exception.def>
        COB_EC_MAX
};
```

7.33.4 Function Documentation

7.33.4.1 int CBL_AND (unsigned char * , unsigned char * , const int)

7.33.4.2 int CBL_EQ (unsigned char * , unsigned char * , const int)

7.33.4.3 int CBL_ERROR_PROC (unsigned char * , unsigned char *)

7.33.4.4 int CBL_EXIT_PROC (unsigned char * , unsigned char *)

7.33.4.5 int CBL_IMP (unsigned char * , unsigned char * , const int)

7.33.4.6 int CBL_NIMP (unsigned char * , unsigned char * , const int)

7.33.4.7 int CBL_NOR (unsigned char * , unsigned char * , const int)

7.33.4.8 int CBL_NOT (unsigned char * , const int)

7.33.4.9 int CBL_OC_NANOSLEEP (unsigned char *)

7.33.4.10 int CBL_OR (unsigned char * , unsigned char * , const int)

7.33.4.11 int CBL_TOLOWER (unsigned char * , const int)

7.33.4.12 int CBL_TOUPPER (unsigned char * , const int)

7.33.4.13 int CBL_X91 (unsigned char * , const unsigned char * , unsigned char *)

7.33.4.14 int CBL_XF4 (unsigned char * , unsigned char *)

7.33.4.15 int CBL_XF5 (unsigned char * , unsigned char *)

7.33.4.16 int CBL_XOR (unsigned char * , unsigned char * , const int)

7.33.4.17 void cob_accept_arg_number (cob_field *)

7.33.4.18 void cob_accept_arg_value (cob_field *)

7.33.4.19 void cob_accept_command_line (cob_field *)

7.33.4.20 void cob_accept_date (cob_field *)

7.33.4.21 void cob_accept_date_yyyymmdd (cob_field *)

7.33.4.22 void cob_accept_day (cob_field *)

7.33.4.23 void cob_accept_day_of_week (cob_field *)

7.33.4.24 void cob_accept_day_yyyyddd (cob_field *)

7.33.4.25 void cob_accept_environment (cob_field *)

7.33.4.26 void cob_accept_time (cob_field *)

7.33.4.27 int cob_acuw_justify (unsigned char * , ...)

7.33.4.28 int cob_acuw_sleep (unsigned char *)

7.33.4.29 void cob_allocate (unsigned char ** , cob_field * , cob_field *)

7.33.4.30 void cob_chain_setup (void * , const size_t , const size_t)

7.33.4.31 void cob_check_based (const unsigned char * , const char *)

7.33.4.32 void cob_check_numeric (cob_field * , const char *)

7.33.4.33 void cob_check_odo (const int , const int , const int , const char *)

7.33.4.34 void cob_check_ref_mod (const int , const int , const int , const char *)

7.33.4.35 void cob_check_subscript (const int , const int , const int , const char *)

7.33.4.36 void cob_check_version (const char * , const char * , const int)

7.33.4.37 int cob_cmp (cob_field * , cob_field *)

7.33.4.38 void cob_display_arg_number (cob_field *)

7.33.4.39 void cob_display_command_line (cob_field *)

7.33.4.40 void cob_display_env_value (cob_field *)

7.33.4.41 void cob_display_environment (cob_field *)

7.33.4.42 unsigned char* cob_external_addr (const char * , const int)

7.33.4.43 void cob_fatal_error (const unsigned *int*)

7.33.4.44 void cob_free_alloc (unsigned char ** , unsigned char *)

7.33.4.45 void cob_get_environment (cob_field * , cob_field *)

7.33.4.46 const char* cob_get_exception_name (const int)

7.33.4.47 unsigned char* cob_get_pointer (const unsigned char *)

7.33.4.48 void* cob_get_prog_pointer (const unsigned char *)

7.33.4.49 int cob_get_switch (const int)

7.33.4.50 void cob_init (int , char **)

7.33.4.51 int cob_is_alpha (const cob_field *)

7.33.4.52 int cob_is_lower (const cob_field *)

7.33.4.53 int cob_is_numeric (cob_field *)

7.33.4.54 int cob_is_omitted (const cob_field *)

7.33.4.55 int cob_is_upper (const cob_field *)

7.33.4.56 void* cob_malloc (const size_t)

7.33.4.57 void cob_module_enter (struct cob_module *)

7.33.4.58 void cob_module_leave (struct cob_module *)

7.33.4.59 int cob_numeric_cmp (cob_field *, cob_field *)

Definition at line 1324 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        cob_decimal_set_field (&cob_d2, f2);
        return cob_decimal_cmp (&cob_d1, &cob_d2);
}
```

7.33.4.60 int cob_parameter_size (unsigned char *)

7.33.4.61 void cob_ready_trace (void)

7.33.4.62 void cob_reset_trace (void)

7.33.4.63 int cob_return_args (unsigned char *)

7.33.4.64 void cob_runtime_error (const char *, ...)

7.33.4.65 void cob_set_environment (cob_field *, cob_field *)

7.33.4.66 void cob_set_exception (const int)

7.33.4.67 void cob_set_location (const char *, const char *, const unsigned *int,* const char *, const char *, const char *)

7.33.4.68 void cob_set_switch (const int , const int)

7.33.4.69 void cob_stop_run (const int)

7.33.4.70 void cob_table_sort (cob_field *, const int)

7.33.4.71 void cob_table_sort_init (const int , const unsigned char *)

7.33.4.72 void cob_table_sort_init_key (const int , cob_field *, size_t)

7.33.4.73 void* cobcommandline (int , int *, char ***, char ***, char **)

7.33.4.74 void cobexit (int)

7.33.4.75 char* cobgetenv (const char *)

7.33.4.76 int cobinit (void)

7.33.4.77 int cobputenv (char *)

7.33.4.78 int cobtidy (void)

7.33.4.79 int SYSTEM (const unsigned char *)

7.33.5 Variable Documentation

7.33.5.1 DLL_EXPIMP int cob_call_params

Definition at line 163 of file common.c.

7.33.5.2 DLL_EXPIMP struct cob_module* cob_current_module

Definition at line 158 of file common.c.

7.33.5.3 DLL_EXPIMP int cob_exception_code

Definition at line 161 of file common.c.

7.33.5.4 DLL_EXPIMP cob_field cob_high

Definition at line 176 of file common.c.

7.33.5.5 DLL_EXPIMP int cob_initial_external

Definition at line 165 of file common.c.

7.33.5.6 DLL_EXPIMP int cob_initialized

Definition at line 160 of file common.c.

7.33.5.7 DLL_EXPIMP cob_field cob_low

Definition at line 177 of file common.c.

7.33.5.8 DLL_EXPIMP cob_field cob_one

Definition at line 179 of file common.c.

7.33.5.9 DLL_EXPIMP cob_field cob_quote

Definition at line 178 of file common.c.

7.33.5.10 DLL_EXPIMP int cob_save_call_params

Definition at line 164 of file common.c.

7.33.5.11 DLL_EXPIMP cob_field cob_space

Definition at line 175 of file common.c.

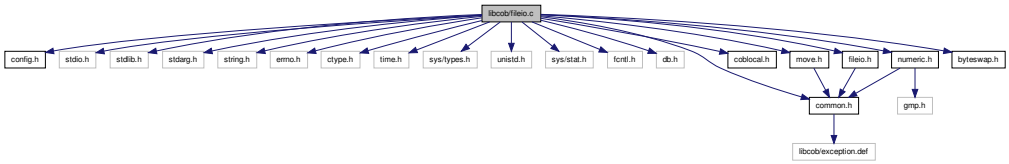
7.33.5.12 DLL_EXPIMP cob_field cob_zero

Definition at line 174 of file common.c.

7.34 libcob/fileio.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
#include <time.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <db.h>
#include "common.h"
#include "coblocal.h"
#include "move.h"
#include "numeric.h"
#include "fileio.h"
#include "byteswap.h"
```

Include dependency graph for fileio.c:



Classes

- struct cobitem
- struct memory_struct
- struct file_struct
- struct cobsort
- struct **file_list**
- struct indexed_file

Defines

- #define _LFS64_LARGEFILE 1
- #define _LFS64_STDIO 1
- #define _FILE_OFFSET_BITS 64
- #define _LARGEFILE64_SOURCE 1
- #define SEEK_INIT(f) fseek ((FILE *)f->file, (off_t)0, SEEK_CUR)
- #define O_BINARY 0
- #define O_LARGEFILE 0
- #define INITIAL_FLAGS 0
- #define COBSORTEND 1
- #define COBSORTABORT 2
- #define COBSORTFILEERR 3
- #define COBSORTNOTOPEN 4
- #define RETURN_STATUS(x) do { save_status (f, x, fnstatus); return; } while (0)
- #define NUM_PREFIX sizeof(prefix) / sizeof(char *)
- #define DB_PUT(db, flags) db->put (db, NULL, &p->key, &p->data, flags)
- #define DB_GET(db, flags) db->get (db, NULL, &p->key, &p->data, flags)
- #define DB_SEQ(db, flags) db->c_get (db, &p->key, &p->data, flags)
- #define DB_DEL(db, key, flags) db->del (db, NULL, key, flags)
- #define DB_CLOSE(db) db->close (db, 0)
- #define DB_SYNC(db) db->sync (db, 0)
- #define cob_dbtsize_t u_int32_t
- #define DBT_SET(key, fld)

Functions

- void cob_unlock_file (cob_file *f, cob_field *fnstatus)
- void cob_open (cob_file *f, const int mode, const int sharing, cob_field *fnstatus)
- void cob_close (cob_file *f, const int opt, cob_field *fnstatus)
- void cob_start (cob_file *f, const int cond, cob_field *key, cob_field *fnstatus)
- void cob_read (cob_file *f, cob_field *key, cob_field *fnstatus, int read_opts)
- void cob_write (cob_file *f, cob_field *rec, const int opt, cob_field *fnstatus)
- void cob_rewrite (cob_file *f, cob_field *rec, const int opt, cob_field *fnstatus)
- void cob_delete (cob_file *f, cob_field *fnstatus)
- void cob_commit (void)
- void cob_rollback (void)
- void cob_default_error_handle (void)
- void cob_init_fileio (void)
- void cob_exit_fileio (void)
- int CBL_OPEN_FILE (unsigned char *file_name, unsigned char *file_access, unsigned char *file_lock, unsigned char *file_dev, unsigned char *file_handle)
- int CBL_CREATE_FILE (unsigned char *file_name, unsigned char *file_access, unsigned char *file_lock, unsigned char *file_dev, unsigned char *file_handle)
- int CBL_READ_FILE (unsigned char *file_handle, unsigned char *file_offset, unsigned char *file_len, unsigned char *flags, unsigned char *buf)
- int CBL_WRITE_FILE (unsigned char *file_handle, unsigned char *file_offset, unsigned char *file_len, unsigned char *flags, unsigned char *buf)
- int CBL_CLOSE_FILE (unsigned char *file_handle)
- int CBL_FLUSH_FILE (unsigned char *file_handle)
- int CBL_DELETE_FILE (unsigned char *file_name)
- int CBL_COPY_FILE (unsigned char *fname1, unsigned char *fname2)
- int CBL_CHECK_FILE_EXIST (unsigned char *file_name, unsigned char *file_info)
- int CBL_RENAME_FILE (unsigned char *fname1, unsigned char *fname2)
- int CBL_GET_CURRENT_DIR (const int flags, const int dir_length, unsigned char *dir)
- int CBL_CREATE_DIR (unsigned char *dir)
- int CBL_CHANGE_DIR (unsigned char *dir)
- int CBL_DELETE_DIR (unsigned char *dir)
- int cob_acuw_mkdir (unsigned char *dir)
- int cob_acuw_chdir (unsigned char *dir, unsigned char *status)
- int cob_acuw_copyfile (unsigned char *fname1, unsigned char *fname2, unsigned char *file_type)
- int cob_acuw_file_info (unsigned char *file_name, unsigned char *file_info)
- int cob_acuw_file_delete (unsigned char *file_name, unsigned char *file_type)
- void cob_file_sort_using (cob_file *sort_file, cob_file *data_file)
- void cob_file_sort_giving (cob_file *sort_file, const size_t varcnt,...)
- void cob_file_sort_init (cob_file *f, const int nkeys, const unsigned char *collating_sequence, void *sort_return, cob_field *fnstatus)
- void cob_file_sort_init_key (cob_file *f, const int flag, cob_field *field, size_t offset)
- void cob_file_sort_close (cob_file *f)
- void cob_file_release (cob_file *f)
- void cob_file_return (cob_file *f)

Variables

- cob_file * cob_error_file

7.34.1 Define Documentation

7.34.1.1 #define _FILE_OFFSET_BITS 64

Definition at line 25 of file fileio.c.

7.34.1.2 #define _LARGEFILE64_SOURCE 1

Definition at line 26 of file fileio.c.

7.34.1.3 #define _LFS64_LARGEFILE 1

Definition at line 23 of file fileio.c.

7.34.1.4 #define _LFS64_STDIO 1

Definition at line 24 of file fileio.c.

7.34.1.5 #define cob_dbtsize_t u_int32_t

Definition at line 288 of file fileio.c.

7.34.1.6 #define COBSORTABORT 2

Definition at line 154 of file fileio.c.

7.34.1.7 #define COBSORTEND 1

Definition at line 153 of file fileio.c.

7.34.1.8 #define COBSORTFILEERR 3

Definition at line 155 of file fileio.c.

7.34.1.9 #define COBSORTNOTOPEN 4

Definition at line 156 of file fileio.c.

7.34.1.10 #define DB_CLOSE(*db*) db->close (db, 0)

Definition at line 286 of file fileio.c.

7.34.1.11 #define DB_DEL(*db, key, flags*) db->del (db, NULL, key, flags)

Definition at line 285 of file fileio.c.

7.34.1.12 #define DB_GET(*db, flags*) db->get (db, NULL, &p->key, &p->data, flags)

Definition at line 283 of file fileio.c.

7.34.1.13 #define DB_PUT(*db, flags*) db->put (db, NULL, &p->key, &p->data, flags)

Definition at line 282 of file fileio.c.

7.34.1.14 #define DB_SEQ(*db, flags*) db->c_get (db, &p->key, &p->data, flags)

Definition at line 284 of file fileio.c.

7.34.1.15 #define DB_SYNC(*db*) db->sync (db, 0)

Definition at line 287 of file fileio.c.

7.34.1.16 #define DBT_SET(*key, fld*)

Value:

```
key.data = fld->data;                           \
  key.size = (cob_dbtsize_t) fld->size
```

Definition at line 303 of file fileio.c.

7.34.1.17 #define INITIAL_FLAGS 0

Definition at line 148 of file fileio.c.

7.34.1.18 #define NUM_PREFIX sizeof(prefix) / sizeof(char *)

Definition at line 250 of file fileio.c.

7.34.1.19 #define O_BINARY 0

Definition at line 138 of file fileio.c.

7.34.1.20 #define O_LARGEFILE 0

Definition at line 142 of file fileio.c.

7.34.1.21 #define RETURN_STATUS(*x*) do { save_status (f, x, fnstatus); return; } while (0)

Definition at line 234 of file fileio.c.

7.34.1.22 #define SEEK_INIT(*f*) fseek ((FILE *)f->file, (off_t)0, SEEK_CUR)

Definition at line 132 of file fileio.c.

7.34.2 Function Documentation

7.34.2.1 int CBL_CHANGE_DIR (unsigned char * *dir*)

Definition at line 4857 of file fileio.c.

```
{
        char    *fn;
        int     ret;

        COB_CHK_PARMS (CBL_CHANGE_DIR, 1);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        ret = chdir (fn);
        free (fn);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.34.2.2 int CBL_CHECK_FILE_EXIST (unsigned char * *file_name,* unsigned char * *file_info*)

Definition at line 4722 of file fileio.c.

```
{
        char            *fn;
        struct tm       *tm;
        long long       sz;
        struct stat     st;
        short           y;
        char            d, m, hh, mm, ss;

        COB_CHK_PARMS (CBL_CHECK_FILE_EXIST, 2);

        if (!cob_current_module->cob_procedure_parameters[0]) {
```

```
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        if (stat (fn, &st) < 0) {
                free (fn);
                return 35;
        }
        free (fn);
        sz = st.st_size;
        tm = localtime (&st.st_mtime);
        d = (char) tm->tm_mday;
        m = (char) tm->tm_mon + 1;
        y = tm->tm_year + 1900;
        hh = (char) tm->tm_hour;
        mm = (char) tm->tm_min;
        ss = (char) tm->tm_sec;

#ifndef WORDS_BIGENDIAN
        sz = COB_BSWAP_64 (sz);
        y = COB_BSWAP_16 (y);
#endif
        memcpy (file_info, &sz, 8);
        file_info[8] = d;
        file_info[9] = m;
        memcpy (file_info+10, &y, 2);
        file_info[12] = hh;
        file_info[13] = mm;
        file_info[14] = ss;
        file_info[15] = 0;
        return 0;
}
```

7.34.2.3 int CBL_CLOSE_FILE (unsigned char * *file_handle*)

Definition at line 4631 of file fileio.c.

```
{
        int     fd;

        COB_CHK_PARMS (CBL_CLOSE_FILE, 1);

        memcpy (&fd, file_handle, 4);
        return close (fd);
}
```

7.34.2.4 int CBL_COPY_FILE (unsigned char * *fname1*, unsigned char * *fname2*)

Definition at line 4670 of file fileio.c.

```
{
        char    *fn1;
        char    *fn2;
#ifdef  O_BINARY
        int     flag = O_BINARY;
#else
```

```
        int     flag = 0;
#endif
        int     ret;
        int     i;
        int     fd1, fd2;

        COB_CHK_PARMS (CBL_COPY_FILE, 2);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        if (!cob_current_module->cob_procedure_parameters[1]) {
                return -1;
        }
        fn1 = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);

        flag |= O_RDONLY;
        fd1 = open (fn1, flag, 0);
        if (fd1 < 0) {
                free (fn1);
                return -1;
        }
        free (fn1);
        fn2 = cob_str_from_fld (cob_current_module->cob_procedure_parameters[1]);

        flag &= ~O_RDONLY;
        flag |= O_CREAT | O_TRUNC | O_WRONLY;
        fd2 = open (fn2, flag, 0660);
        if (fd2 < 0) {
                close (fd1);
                free (fn2);
                return -1;
        }
        free (fn2);
        ret = 0;
        while ((i = read (fd1, fn1, sizeof(fn1))) > 0) {
                if (write (fd2, fn1, (size_t)i) < 0) {
                        ret = -1;
                        break;
                }
        }
        close (fd1);
        close (fd2);
        return ret;
}
```

7.34.2.5 int CBL_CREATE_DIR (unsigned char * *dir*)

Definition at line 4833 of file fileio.c.

```
{
        char    *fn;
        int     ret;

        COB_CHK_PARMS (CBL_CREATE_DIR, 1);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
```

```
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
#ifdef  _WIN32
        ret = mkdir (fn);
#else
        ret = mkdir (fn, 0770);
#endif
        free (fn);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.34.2.6 int CBL_CREATE_FILE (unsigned char * *file_name*, unsigned char * *file_access*, unsigned char * *file_lock*, unsigned char * *file_dev*, unsigned char * *file_handle*)

Definition at line 4548 of file fileio.c.

```
{
        COB_CHK_PARMS (CBL_CREATE_FILE, 5);

        return open_cbl_file (file_name, file_access, file_handle, O_CREAT | O_TR
      UNC);
}
```

7.34.2.7 int CBL_DELETE_DIR (unsigned char * *dir*)

Definition at line 4877 of file fileio.c.

```
{
        char    *fn;
        int     ret;

        COB_CHK_PARMS (CBL_DELETE_DIR, 1);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        ret = rmdir (fn);
        free (fn);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.34.2.8 int CBL_DELETE_FILE (unsigned char * *file_name*)

Definition at line 4650 of file fileio.c.

```
{
        char    *fn;
        int     ret;

        COB_CHK_PARMS (CBL_DELETE_FILE, 1);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        ret = unlink (fn);
        free (fn);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.34.2.9 int CBL_FLUSH_FILE (unsigned char * *file_handle*)

Definition at line 4642 of file fileio.c.

```
{
        COB_CHK_PARMS (CBL_FLUSH_FILE, 1);

        return 0;
}
```

7.34.2.10 int CBL_GET_CURRENT_DIR (const int *flags*, const int *dir_length*, unsigned char * *dir*)

Definition at line 4793 of file fileio.c.

```
{
        char    *dirname;
        int     dir_size;
        int     has_space;

        COB_CHK_PARMS (CBL_GET_CURRENT_DIR, 3);

        if (dir_length < 1) {
                return 128;
        }
        if (flags) {
                return 129;
        }
        memset (dir, ' ', (size_t)dir_length);
        dirname = getcwd (NULL, 0);
        if (dirname == NULL) {
                return 128;
        }
        dir_size = (int) strlen (dirname);
        has_space = 0;
        if (strchr (dirname, ' ')) {
                has_space = 2;
```

```
        }
        if (dir_size + has_space > dir_length) {
                free (dirname);
                return 128;
        }
        if (has_space) {
                *dir = '"';
                memcpy (&dir[1], dirname, (size_t)dir_size);
                dir[dir_size + 1] = '"';
        } else {
                memcpy (dir, dirname, (size_t)dir_size);
        }
        free (dirname);
        return 0;
}
```

7.34.2.11 int CBL_OPEN_FILE (unsigned char * *file_name*, unsigned char * *file_access*, unsigned char * *file_lock*, unsigned char * *file_dev*, unsigned char * *file_handle*)

Definition at line 4538 of file fileio.c.

```
{
        COB_CHK_PARMS (CBL_OPEN_FILE, 5);

        return open_cbl_file (file_name, file_access, file_handle, 0);
}
```

7.34.2.12 int CBL_READ_FILE (unsigned char * *file_handle*, unsigned char * *file_offset*, unsigned char * *file_len*, unsigned char * *flags*, unsigned char * *buf*)

Definition at line 4558 of file fileio.c.

```
{
        long long       off;
        int             fd;
        int             len;
        int             rc = 0;
        struct stat     st;

        COB_CHK_PARMS (CBL_READ_FILE, 5);

        memcpy (&fd, file_handle, 4);
        memcpy (&off, file_offset, 8);
        memcpy (&len, file_len, 4);
#ifndef WORDS_BIGENDIAN
        off = COB_BSWAP_64 (off);
        len = COB_BSWAP_32 (len);
#endif
        if (lseek (fd, (off_t)off, SEEK_SET) < 0) {
                return -1;
        }
        if (len > 0) {
                rc = read (fd, buf, (size_t)len);
                if (rc < 0) {
                        rc = -1;
```

```
                } else if (rc == 0) {
                        rc = 10;
                } else {
                        rc = 0;
                }
        }
        if ((*flags & 0x80) != 0) {
                if (fstat (fd, &st) < 0) {
                        return -1;
                }
                off = st.st_size;
#ifndef WORDS_BIGENDIAN
                off = COB_BSWAP_64 (off);
#endif
                memcpy (file_offset, &off, 8);
        }
        return rc;
}
```

7.34.2.13 int CBL_RENAME_FILE (unsigned char * *fname1*, unsigned char * *fname2*)

Definition at line 4767 of file fileio.c.

```
{
        char    *fn1;
        char    *fn2;
        int     ret;

        COB_CHK_PARMS (CBL_RENAME_FILE, 2);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        if (!cob_current_module->cob_procedure_parameters[1]) {
                return -1;
        }
        fn1 = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);

        fn2 = cob_str_from_fld (cob_current_module->cob_procedure_parameters[1]);

        ret = rename (fn1, fn2);
        free (fn1);
        free (fn2);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.34.2.14 int CBL_WRITE_FILE (unsigned char * *file_handle*, unsigned char * *file_offset*, unsigned char * *file_len*, unsigned char * *flags*, unsigned char * *buf*)

Definition at line 4603 of file fileio.c.

```
{
```

```
        long long       off;
        int             fd;
        int             len;
        int             rc;

        COB_CHK_PARMS (CBL_WRITE_FILE, 5);

        memcpy (&fd, file_handle, 4);
        memcpy (&off, file_offset, 8);
        memcpy (&len, file_len, 4);
#ifndef WORDS_BIGENDIAN
        off = COB_BSWAP_64 (off);
        len = COB_BSWAP_32 (len);
#endif
        if (lseek (fd, (off_t)off, SEEK_SET) < 0) {
                return -1;
        }
        rc = write (fd, buf, (size_t)len);
        if (rc < 0) {
                return 30;
        }
        return 0;
}
```

7.34.2.15 int cob_acuw_chdir (unsigned char * *dir,* unsigned char * *status*)

Definition at line 4911 of file fileio.c.

```
{
        int             ret;

        COB_CHK_PARMS (C$CHDIR, 2);

        ret = CBL_CHANGE_DIR (dir);
        if (ret < 0) {
                ret = 128;
        }
        cob_set_int (cob_current_module->cob_procedure_parameters[1], ret);
        return ret;
}
```

7.34.2.16 int cob_acuw_copyfile (unsigned char * *fname1,* unsigned char * *fname2,* unsigned char * *file_type*)

Definition at line 4926 of file fileio.c.

```
{
        int             ret = 128;

        /* RXW - Type is not yet evaluated */

        COB_CHK_PARMS (C$COPY, 3);

        if (cob_call_params < 3) {
                return 128;
```

```
        }
        ret = CBL_COPY_FILE (fname1, fname2);
        if (ret < 0) {
                ret = 128;
        }
        return ret;
}
```

7.34.2.17 int cob_acuw_file_delete (unsigned char * *file_name*, unsigned char * *file_type*)

Definition at line 4993 of file fileio.c.

```
{
        int     ret;

        /* RXW - Type is not yet evaluated */
        COB_CHK_PARMS (C$DELETE, 2);

        if (cob_call_params < 2 || !cob_current_module->cob_procedure_parameters[
      0]) {
                return 128;
        }
        ret = CBL_DELETE_FILE (file_name);
        if (ret < 0) {
                ret = 128;
        }
        return ret;
}
```

7.34.2.18 int cob_acuw_file_info (unsigned char * *file_name*, unsigned char * *file_info*)

Definition at line 4945 of file fileio.c.

```
{
        char                    *fn;
        struct tm               *tm;
        unsigned long long      sz;
        unsigned int            dt;
        short                   y;
        short                   d, m, hh, mm, ss;
        struct stat             st;

        COB_CHK_PARMS (C$FILEINFO, 2);

        if (cob_call_params < 2 || !cob_current_module->cob_procedure_parameters[
      0]) {
                return 128;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        if (stat (fn, &st) < 0) {
                free (fn);
                return 35;
        }
        free (fn);
        sz = st.st_size;
```

```
        tm = localtime (&st.st_mtime);
        d = tm->tm_mday;
        m = tm->tm_mon + 1;
        y = tm->tm_year + 1900;
        hh = tm->tm_hour;
        mm = tm->tm_min;
        ss = tm->tm_sec;

#ifndef WORDS_BIGENDIAN
        sz = COB_BSWAP_64 (sz);
#endif
        memcpy (file_info, &sz, 8);
        dt = (y * 10000) + (m * 100) + d;
#ifndef WORDS_BIGENDIAN
        dt = COB_BSWAP_32 (dt);
#endif
        memcpy (file_info + 8, &dt, 4);
        dt = (hh * 1000000) + (mm * 10000) + (ss * 100);
#ifndef WORDS_BIGENDIAN
        dt = COB_BSWAP_32 (dt);
#endif
        memcpy (file_info + 12, &dt, 4);
        return 0;
}
```

7.34.2.19 int cob_acuw_mkdir (unsigned char * *dir*)

Definition at line 4897 of file fileio.c.

```
{
        int             ret;

        COB_CHK_PARMS (C$MAKEDIR, 1);

        ret = CBL_CREATE_DIR (dir);
        if (ret < 0) {
                ret = 128;
        }
        return ret;
}
```

7.34.2.20 void cob_close (cob_file * *f,* const int *opt,* cob_field * *fnstatus*)

Definition at line 3987 of file fileio.c.

```
{
        int     ret;

        f->flag_read_done = 0;

        if (f->special) {
                f->open_mode = COB_OPEN_CLOSED;
                RETURN_STATUS (COB_STATUS_00_SUCCESS);
        }
        if (f->open_mode == COB_OPEN_CLOSED) {
```

```
                RETURN_STATUS (COB_STATUS_42_NOT_OPEN);
        }

        if (f->flag_nonexistent) {
                ret = COB_STATUS_00_SUCCESS;
        } else {
#ifdef  WITH_SEQRA_EXTFH
                if (f->organization != COB_ORG_INDEXED) {
                        ret = extfh_cob_file_close (f, opt);
                } else {
#endif
                        ret = fileio_funcs[(int)f->organization]->close (f, opt);

#ifdef  WITH_SEQRA_EXTFH
                }
#endif
        }

        if (ret == COB_STATUS_00_SUCCESS) {
                switch (opt) {
                case COB_CLOSE_LOCK:
                        f->open_mode = COB_OPEN_LOCKED;
                        break;
                default:
                        f->open_mode = COB_OPEN_CLOSED;
                        break;
                }
        }

        RETURN_STATUS (ret);
}
```

7.34.2.21 void cob_commit (void)

Definition at line 4273 of file fileio.c.

```
{
        struct file_list        *l;

        for (l = file_cache; l; l = l->next) {
                cob_file_unlock (l->file);
        }
}
```

7.34.2.22 void cob_default_error_handle (void)

Definition at line 4293 of file fileio.c.

```
{
        const char      *msg;
        unsigned char   *file_status;
        char            *filename;
        int             status;

        file_status = cob_error_file->file_status;
```

```
status = cob_d2i(file_status[0]) * 10 + cob_d2i(file_status[1]);
switch (status) {
case COB_STATUS_10_END_OF_FILE:
        msg = "End of file";
        break;
case COB_STATUS_14_OUT_OF_KEY_RANGE:
        msg = "Key out of range";
        break;
case COB_STATUS_21_KEY_INVALID:
        msg = "Key order not ascending";
        break;
case COB_STATUS_22_KEY_EXISTS:
        msg = "Record key already exists";
        break;
case COB_STATUS_23_KEY_NOT_EXISTS:
        msg = "Record key does not exist";
        break;
case COB_STATUS_30_PERMANENT_ERROR:
        msg = "Permanent file error";
        break;
case COB_STATUS_35_NOT_EXISTS:
        msg = "File does not exist";
        break;
case COB_STATUS_37_PERMISSION_DENIED:
        msg = "Permission denied";
        break;
case COB_STATUS_41_ALREADY_OPEN:
        msg = "File already open";
        break;
case COB_STATUS_42_NOT_OPEN:
        msg = "File not open";
        break;
case COB_STATUS_43_READ_NOT_DONE:
        msg = "READ must be executed first";
        break;
case COB_STATUS_44_RECORD_OVERFLOW:
        msg = "Record overflow";
        break;
case COB_STATUS_46_READ_ERROR:
        msg = "Failed to read";
        break;
case COB_STATUS_47_INPUT_DENIED:
        msg = "READ/START not allowed";
        break;
case COB_STATUS_48_OUTPUT_DENIED:
        msg = "WRITE not allowed";
        break;
case COB_STATUS_49_I_O_DENIED:
        msg = "DELETE/REWRITE not allowed";
        break;
case COB_STATUS_51_RECORD_LOCKED:
        msg = "Record locked by another file connector";
        break;
case COB_STATUS_52_EOP:
        msg = "A page overflow condition occurred";
        break;
case COB_STATUS_57_I_O_LINAGE:
        msg = "LINAGE values invalid";
        break;
case COB_STATUS_61_FILE_SHARING:
        msg = "File sharing conflict";
        break;
```

```
        case COB_STATUS_91_NOT_AVAILABLE:
                msg = "Runtime library is not configured for this operation";
                break;
        default:
                msg = "Unknown file error";
                break;
        }

        filename = cob_malloc (COB_MEDIUM_BUFF);
        cob_field_to_string (cob_error_file->assign, filename);
        cob_runtime_error ("%s (STATUS = %02d) File : '%s'", msg,
                                status, filename);
        free (filename);
}
```

7.34.2.23 void cob_delete (cob_file * *f*, cob_field * *fnstatus*)

Definition at line 4247 of file fileio.c.

```
{
        int     ret;
        int     read_done = f->flag_read_done;

        f->flag_read_done = 0;

        if (unlikely(f->open_mode == COB_OPEN_CLOSED ||
            f->open_mode != COB_OPEN_I_O)) {
                RETURN_STATUS (COB_STATUS_49_I_O_DENIED);
        }

        if (f->access_mode == COB_ACCESS_SEQUENTIAL && !read_done) {
                RETURN_STATUS (COB_STATUS_43_READ_NOT_DONE);
        }

        ret = fileio_funcs[(int)f->organization]->fdelete (f);

        if (unlikely(cob_do_sync && ret == 0)) {
                cob_sync (f, cob_do_sync);
        }

        RETURN_STATUS (ret);
}
```

7.34.2.24 void cob_exit_fileio (void)

Definition at line 4426 of file fileio.c.

```
{
        struct file_list        *l;

        for (l = file_cache; l; l = l->next) {
                if (l->file->open_mode != COB_OPEN_CLOSED &&
                     l->file->open_mode != COB_OPEN_LOCKED) {
                        cob_field_to_string (l->file->assign, runtime_buffer);
                        cob_close (l->file, 0, NULL);
```

```
                        fprintf (stderr, "WARNING - Implicit CLOSE of %s (\"%s\")
      \n",
                                l->file->select_name, runtime_buffer);
                        fflush (stderr);
                }
        }
#ifdef  USE_DB41
        free (record_lock_object);
        if (bdb_env) {
                bdb_env->lock_id_free (bdb_env, bdb_lock_id);
                bdb_env->close (bdb_env, 0);
        }
#endif
#if     defined(WITH_INDEX_EXTFH) || defined(WITH_SEQRA_EXTFH)
        extfh_cob_exit_fileio ();
#endif
}
```

7.34.2.25 void cob_file_release (cob_file * f)

Definition at line 5654 of file fileio.c.

```
{
        struct cobsort  *hp;
        cob_field       *fnstatus = NULL;
        int             ret;

        hp = f->file;
        if (likely(hp)) {
                fnstatus = hp->fnstatus;
        }
        ret = cob_file_sort_submit (f, f->record->data);
        switch (ret) {
        case 0:
                RETURN_STATUS (COB_STATUS_00_SUCCESS);
                break;
        default:
                if (likely(hp)) {
                        *(int *)(hp->sort_return) = 16;
                }
                RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
                break;
        }
}
```

7.34.2.26 void cob_file_return (cob_file * f)

Definition at line 5679 of file fileio.c.

```
{
        struct cobsort  *hp;
        cob_field       *fnstatus = NULL;
        int             ret;

        hp = f->file;
```

```
        if (likely(hp)) {
                fnstatus = hp->fnstatus;
        }
        ret = cob_file_sort_retrieve (f, f->record->data);
        switch (ret) {
        case 0:
                RETURN_STATUS (COB_STATUS_00_SUCCESS);
                break;
        case COBSORTEND:
                RETURN_STATUS (COB_STATUS_10_END_OF_FILE);
                break;
        default:
                if (likely(hp)) {
                        *(int *)(hp->sort_return) = 16;
                }
                RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
                break;
        }
}
```

7.34.2.27 void cob_file_sort_close (cob_file * *f*)

Definition at line 5631 of file fileio.c.

```
{
        struct cobsort  *hp;
        cob_field       *fnstatus = NULL;
        size_t          i;

        hp = f->file;
        if (likely(hp)) {
                fnstatus = hp->fnstatus;
                cob_free_list (hp->empty);
                for (i = 0; i < 4; i++) {
                        cob_free_list (hp->queue[i].first);
                        if (hp->file[i].fp != NULL) {
                                fclose (hp->file[i].fp);
                        }
                }
                free (hp);
        }
        f->file = NULL;
        RETURN_STATUS (COB_STATUS_00_SUCCESS);
}
```

7.34.2.28 void cob_file_sort_giving (cob_file * *sort_file*, const size_t *varcnt*, ...)

Definition at line 5544 of file fileio.c.

```
{
        cob_file        **fbase;
        struct cobsort  *hp;
        size_t          i;
        int             ret;
        int             opt;
```

```
        va_list         args;

        fbase = cob_malloc (varcnt * sizeof(cob_file *));
        va_start (args, varcnt);
        for (i = 0; i < varcnt; i++) {
                fbase[i] = va_arg (args, cob_file *);
        }
        va_end (args);
        for (i = 0; i < varcnt; i++) {
                cob_open (fbase[i], COB_OPEN_OUTPUT, 0, NULL);
        }
        while (1) {
                ret = cob_file_sort_retrieve (sort_file, sort_file->record->data)
     ;
                if (ret) {
                        if (ret == COBSORTEND) {
                                sort_file->file_status[0] = '1';
                                sort_file->file_status[1] = '0';
                        } else {
                                hp = sort_file->file;
                                *(int *)(hp->sort_return) = 16;
                                sort_file->file_status[0] = '3';
                                sort_file->file_status[1] = '0';
                        }
                        break;
                }
                for (i = 0; i < varcnt; i++) {
                        if (fbase[i]->special ||
                            fbase[i]->organization == COB_ORG_LINE_SEQUENTIAL) {
                                opt = COB_WRITE_BEFORE | COB_WRITE_LINES | 1;
                        } else {
                                opt = 0;
                        }
                        cob_copy_check (fbase[i], sort_file);
                        cob_write (fbase[i], fbase[i]->record, opt, NULL);
                }
        }
        for (i = 0; i < varcnt; i++) {
                cob_close (fbase[i], COB_CLOSE_NORMAL, NULL);
        }
        free (fbase);
}
```

7.34.2.29 void cob_file_sort_init (cob_file * *f*, const int *nkeys*, const unsigned char * *collating_sequence*, void * *sort_return*, cob_field * *fnstatus*)

Definition at line 5594 of file fileio.c.

```
{
        struct cobsort  *p;

        p = cob_malloc (sizeof (struct cobsort));
        p->fnstatus = fnstatus;
        p->size = f->record_max;
        p->r_size = f->record_max + sizeof(size_t);
        p->w_size = f->record_max + sizeof(size_t) + 1;
        p->pointer = f;
        p->sort_return = sort_return;
        *(int *)sort_return = 0;
```

```
        p->memory = (size_t)cob_sort_memory / (p->size + sizeof(struct cobitem));

        f->file = p;
        f->keys = cob_malloc (sizeof (struct cob_file_key) * nkeys);
        f->nkeys = 0;
        if (collating_sequence) {
                f->sort_collating = collating_sequence;
        } else {
                f->sort_collating = cob_current_module->collating_sequence;
        }
        RETURN_STATUS (COB_STATUS_00_SUCCESS);
}
```

7.34.2.30 void cob_file_sort_init_key (cob_file * *f*, const int *flag*, cob_field * *field*, size_t *offset*)

Definition at line 5621 of file fileio.c.

```
{
        f->keys[f->nkeys].flag = flag;
        f->keys[f->nkeys].field = field;
        f->keys[f->nkeys].offset = offset;
        f->nkeys++;
}
```

7.34.2.31 void cob_file_sort_using (cob_file * *sort_file*, cob_file * *data_file*)

Definition at line 5524 of file fileio.c.

```
{
        int             ret;

        cob_open (data_file, COB_OPEN_INPUT, 0, NULL);
        while (1) {
                cob_read (data_file, NULL, NULL, COB_READ_NEXT);
                if (data_file->file_status[0] != '0') {
                        break;
                }
                cob_copy_check (sort_file, data_file);
                ret = cob_file_sort_submit (sort_file, sort_file->record->data);
                if (ret) {
                        break;
                }
        }
        cob_close (data_file, COB_CLOSE_NORMAL, NULL);
}
```

7.34.2.32 void cob_init_fileio (void)

Definition at line 4379 of file fileio.c.

```
{
        char    *s;
        int     n;

        if ((s = getenv ("COB_SYNC")) != NULL) {
                if (*s == 'Y' || *s == 'y') {
                        cob_do_sync = 1;
                }
                if (*s == 'P' || *s == 'p') {
                        cob_do_sync = 2;
                }
        }
        if ((s = getenv ("COB_SORT_MEMORY")) != NULL) {
                n = atoi (s);
                if (n >= 1024*1024) {
                        cob_sort_memory = n;
                }
        }
        cob_file_path = getenv ("COB_FILE_PATH");
        if (cob_file_path) {
                if (!*cob_file_path || *cob_file_path == ' ') {
                        cob_file_path = NULL;
                }
        }
        cob_ls_nulls = getenv ("COB_LS_NULLS");
        cob_ls_fixed = getenv ("COB_LS_FIXED");

        file_open_env = cob_malloc (COB_SMALL_BUFF);
        file_open_name = cob_malloc (COB_SMALL_BUFF);
        file_open_buff = cob_malloc (COB_SMALL_BUFF);

#ifdef  USE_DB41
        bdb_home = getenv ("DB_HOME");
        join_environment ();
        record_lock_object = cob_malloc (1024);
        bdb_buff = cob_malloc (COB_SMALL_BUFF);
        rlo_size = 1024;
#endif

#if     defined(WITH_INDEX_EXTFH) || defined(WITH_SEQRA_EXTFH)
        extfh_cob_init_fileio (&sequential_funcs, &lineseq_funcs, &relative_funcs
      , &cob_file_write_opt);
#endif

}
```

7.34.2.33 void cob_open (cob_file * *f*, const int *mode*, const int *sharing*, cob_field * *fnstatus*)

Definition at line 3734 of file fileio.c.

```
{
        char            *p;
        char            *src;
        char            *dst;
        size_t          i;
        size_t          simple;
        int             was_not_exist = 0;
        struct stat     st;
```

```
        f->flag_read_done = 0;

        /* file was previously closed with lock */
        if (f->open_mode == COB_OPEN_LOCKED) {
                RETURN_STATUS (COB_STATUS_38_CLOSED_WITH_LOCK);
        }

        /* file is already open */
        if (f->open_mode != COB_OPEN_CLOSED) {
                RETURN_STATUS (COB_STATUS_41_ALREADY_OPEN);
        }

        f->last_open_mode = mode;
        f->flag_nonexistent = 0;
        f->flag_end_of_file = 0;
        f->flag_begin_of_file = 0;
        f->flag_first_read = 2;

        if (f->special) {
                if (f->special == 1) {
                        if (mode != COB_OPEN_INPUT) {
                                RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
                        }
                        f->file = stdin;
                        f->open_mode = mode;
                        RETURN_STATUS (COB_STATUS_00_SUCCESS);
                } else {
                        if (mode != COB_OPEN_OUTPUT) {
                                RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
                        }
                        f->file = stdout;
                        f->open_mode = mode;
                        RETURN_STATUS (COB_STATUS_00_SUCCESS);
                }
        }

        /* obtain the file name */
        cob_field_to_string (f->assign, file_open_name);

#ifdef  WITH_INDEX_EXTFH
        if (f->organization == COB_ORG_INDEXED) {
                int     ret;

                ret = extfh_indexed_locate (f, file_open_name);
                switch (ret) {
                case COB_NOT_CONFIGURED:
                        /* EXTFH requires OC to process the filename */
                        break;
                case COB_STATUS_00_SUCCESS:
                        /* EXTFH recognized the file */
                        goto file_available;
                default:
                        /* EXTFH detected an error */
                        RETURN_STATUS (ret);
                }
        }
#endif  /* WITH_INDEX_EXTFH */

#ifdef  WITH_SEQRA_EXTFH
        if (f->organization != COB_ORG_INDEXED) {
                int     ret;
```

```
                ret = extfh_seqra_locate (f, file_open_name);
                switch (ret) {
                case COB_NOT_CONFIGURED:
                        /* EXTFH requires OC to process the filename */
                        break;
                case COB_STATUS_00_SUCCESS:
                        /* EXTFH recognized the file */
                        goto file_available;
                default:
                        /* EXTFH detected an error */
                        RETURN_STATUS (ret);
                }
        }
#endif  /* WITH_SEQRA_EXTFH */

        if (cob_current_module->flag_filename_mapping) {
                src = file_open_name;
                dst = file_open_buff;
                simple = 1;
                /* expand envoronment variables */
                /* ex. "$TMPDIR/foo" -> "/tmp/foo" */
                while (*src) {
                        if (!isalnum (*src) && *src != '_' && *src != '-') {
                                simple = 0;
                        }
                        if (*src == '$') {
                                for (i = 1; ; i++) {
                                        if (!isalnum (src[i]) && src[i] != '_' &&
       *src != '-') {
                                                break;
                                        }
                                }
                                memcpy (file_open_env, src + 1, i - 1);
                                file_open_env[i - 1] = 0;
                                if ((p = getenv (file_open_env)) != NULL) {
                                        strcpy (dst, p);
                                        dst += strlen (p);
                                }
                                src += i;
                        } else {
                                *dst++ = *src++;
                        }
                }
                *dst = 0;
                strncpy (file_open_name, file_open_buff, COB_SMALL_MAX);

                /* resolve by environment variables */
                /* ex. "TMPFILE" -> DD_TMPFILE, dd_TMPFILE, or TMPFILE */
                if (simple) {
                        for (i = 0; i < NUM_PREFIX; i++) {
                                snprintf (file_open_buff, COB_SMALL_MAX, "%s%s",
                                          prefix[i], file_open_name);
                                if ((p = getenv (file_open_buff)) != NULL) {
                                        strncpy (file_open_name, p,
      COB_SMALL_MAX);
                                        break;
                                }
                        }
                        if (i == NUM_PREFIX && cob_file_path) {
                                snprintf (file_open_buff, COB_SMALL_MAX, "%s/%s",
```

```
                                        cob_file_path, file_open_name);
                                strncpy (file_open_name, file_open_buff,
                                        COB_SMALL_MAX);
                        }
                }
        }
        /* check if the file exists */
#ifdef  USE_DB41
        if (f->organization == COB_ORG_INDEXED) {
                if ((bdb_env && bdb_nofile (file_open_name)) ||
                     (!bdb_env && stat (file_open_name, &st) == -1 && errno == EN
      OENT)) {
                        was_not_exist = 1;
                        if (mode != COB_OPEN_OUTPUT && f->flag_optional == 0) {
                                RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                        }
                }
        } else if (stat (file_open_name, &st) == -1 && errno == ENOENT) {
#else   /* USE_DB41 */

#if     defined(WITH_CISAM) || defined(WITH_DISAM) || defined(WITH_VBISAM)
        if (f->organization == COB_ORG_INDEXED) {
                strncpy (file_open_buff, file_open_name, COB_SMALL_MAX);
                strcat (file_open_buff, ".idx");
                if (stat (file_open_buff, &st) == -1 && errno == ENOENT) {
                        was_not_exist = 1;
                        if (mode != COB_OPEN_OUTPUT && f->flag_optional == 0) {
                                RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                        }
                }
                strncpy (file_open_buff, file_open_name, COB_SMALL_MAX);
                strcat (file_open_buff, ".dat");
                if (stat (file_open_buff, &st) == -1 && errno == ENOENT) {
                        was_not_exist = 1;
                        if (mode != COB_OPEN_OUTPUT && f->flag_optional == 0) {
                                RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                        }
                }
        } else if (stat (file_open_name, &st) == -1 && errno == ENOENT) {
#else   /* WITH_CISAM || WITH_DISAM || WITH_VBISAM */
        if (stat (file_open_name, &st) == -1 && errno == ENOENT) {
#endif  /* WITH_CISAM || WITH_DISAM || WITH_VBISAM */

#endif  /* USE_DB41 */

                was_not_exist = 1;
                if (mode != COB_OPEN_OUTPUT && f->flag_optional == 0) {
                        RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                }
        }

#if     defined(WITH_INDEX_EXTFH) || defined(WITH_SEQRA_EXTFH)
file_available:
#endif  /* WITH_INDEX_EXTFH || WITH_SEQRA_EXTFH */

        cob_cache_file (f);

        /* open the file */
#ifdef  WITH_SEQRA_EXTFH
        if (f->organization != COB_ORG_INDEXED) {
                int     ret;
```

```
                ret = extfh_cob_file_open (f, file_open_name, mode, sharing);
                switch (ret) {
                case COB_STATUS_00_SUCCESS:
                        f->open_mode = mode;
                        break;
                case COB_STATUS_35_NOT_EXISTS:
                        if (f->flag_optional) {
                                f->open_mode = mode;
                                f->flag_nonexistent = 1;
                                f->flag_end_of_file = 1;
                                f->flag_begin_of_file = 1;
                                RETURN_STATUS (COB_STATUS_05_SUCCESS_OPTIONAL);
                        }
                        break;
                }
                RETURN_STATUS (ret);
        }
#endif
#if     defined(WITH_CISAM) || defined(WITH_DISAM) || defined(WITH_VBISAM)
        if (f->organization == COB_ORG_INDEXED) {
                /* Do this here to avoid mangling of the status in the 'switch' b
      elow */
                RETURN_STATUS (fileio_funcs[(int)f->organization]->open (f, file_
      open_name, mode, sharing));
        }
#endif
        switch (fileio_funcs[(int)f->organization]->open (f, file_open_name, mode
      , sharing)) {
        case 0:
                f->open_mode = mode;
                if (f->flag_optional && was_not_exist) {
                        RETURN_STATUS (COB_STATUS_05_SUCCESS_OPTIONAL);
                } else {
                        RETURN_STATUS (COB_STATUS_00_SUCCESS);
                }
        case ENOENT:
                if (mode == COB_OPEN_EXTEND || mode == COB_OPEN_OUTPUT) {
                        RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
                }
                if (f->flag_optional) {
                        f->open_mode = mode;
                        f->flag_nonexistent = 1;
                        f->flag_end_of_file = 1;
                        f->flag_begin_of_file = 1;
                        RETURN_STATUS (COB_STATUS_05_SUCCESS_OPTIONAL);
                } else {
                        RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                }
        case EACCES:
        case EISDIR:
        case EROFS:
                RETURN_STATUS (COB_STATUS_37_PERMISSION_DENIED);
        case EAGAIN:
        case COB_STATUS_61_FILE_SHARING:
                RETURN_STATUS (COB_STATUS_61_FILE_SHARING);
        case COB_STATUS_91_NOT_AVAILABLE:
                RETURN_STATUS (COB_STATUS_91_NOT_AVAILABLE);
        case COB_LINAGE_INVALID:
                RETURN_STATUS (COB_STATUS_57_I_O_LINAGE);
        default:
                RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
        }
```

```
}
```

7.34.2.34 void cob_read (cob_file * f, cob_field * key, cob_field * fnstatus, int read_opts)

Definition at line 4081 of file fileio.c.

```
{
        int     ret;

        f->flag_read_done = 0;

        if (unlikely(f->flag_nonexistent)) {
                if (f->flag_first_read == 0) {
                        RETURN_STATUS (COB_STATUS_23_KEY_NOT_EXISTS);
                }
                f->flag_first_read = 0;
                RETURN_STATUS (COB_STATUS_10_END_OF_FILE);
        }

        /* sequential read at the end of file is an error */
        if (key == NULL) {
                if (f->flag_end_of_file && !(read_opts & COB_READ_PREVIOUS)) {
                        RETURN_STATUS (COB_STATUS_46_READ_ERROR);
                }
                if (f->flag_begin_of_file && (read_opts & COB_READ_PREVIOUS)) {
                        RETURN_STATUS (COB_STATUS_46_READ_ERROR);
                }
        }

        if (unlikely(f->open_mode == COB_OPEN_CLOSED
            || f->open_mode == COB_OPEN_OUTPUT
            || f->open_mode == COB_OPEN_EXTEND)) {
                RETURN_STATUS (COB_STATUS_47_INPUT_DENIED);
        }

#ifdef  USE_DB41
        if (f->organization == COB_ORG_INDEXED && bdb_env != NULL) {
                if (f->open_mode != COB_OPEN_I_O  ||
                    (f->lock_mode & COB_LOCK_EXCLUSIVE)) {
                        read_opts &= ~COB_READ_LOCK;
                } else if ((f->lock_mode & COB_LOCK_AUTOMATIC) &&
                   !(read_opts & COB_READ_NO_LOCK)) {
                        read_opts |= COB_READ_LOCK;
                }
        } else {
                read_opts &= ~COB_READ_LOCK;
        }
#endif
        if (key) {
                ret = fileio_funcs[(int)f->organization]->read (f, key, read_opts
      );
        } else {
                ret = fileio_funcs[(int)f->organization]->read_next (f, read_opts
      );
        }

        switch (ret) {
        case COB_STATUS_00_SUCCESS:
```

```
                f->flag_first_read = 0;
                f->flag_read_done = 1;
                f->flag_end_of_file = 0;
                f->flag_begin_of_file = 0;
                if (f->record_size && f->organization != COB_ORG_LINE_SEQUENTIAL)
        {
                        cob_set_int (f->record_size, (int) f->record->size);
                }
                break;
         case COB_STATUS_10_END_OF_FILE:
                if (read_opts & COB_READ_PREVIOUS) {
                        f->flag_begin_of_file = 1;
                } else {
                        f->flag_end_of_file = 1;
                }
                break;
         }

         RETURN_STATUS (ret);
}
```

7.34.2.35 void cob_rewrite (cob_file * *f*, cob_field * *rec*, const int *opt*, cob_field * *fnstatus*)

Definition at line 4201 of file fileio.c.

```
{
         int     ret;
         int     read_done = f->flag_read_done;

         f->flag_read_done = 0;

         if (unlikely(f->open_mode == COB_OPEN_CLOSED ||
             f->open_mode != COB_OPEN_I_O)) {
                 RETURN_STATUS (COB_STATUS_49_I_O_DENIED);
         }

         if (f->access_mode == COB_ACCESS_SEQUENTIAL && !read_done) {
                 RETURN_STATUS (COB_STATUS_43_READ_NOT_DONE);
         }

         if (f->organization == COB_ORG_SEQUENTIAL) {
                 if (f->record->size != rec->size) {
                         RETURN_STATUS (COB_STATUS_44_RECORD_OVERFLOW);
                 }

                 if (f->record_size) {
                         if (f->record->size != (size_t)cob_get_int (f->
       record_size)) {
                                 RETURN_STATUS (COB_STATUS_44_RECORD_OVERFLOW);
                         }
                 }
         }

/* RXW
#ifdef  USE_DB41
         if (f->organization != COB_ORG_INDEXED || bdb_env == NULL) {
                 opt &= ~COB_WRITE_LOCK;
         }
```

```
#endif
*/

        ret = fileio_funcs[(int)f->organization]->rewrite (f, opt);

        if (unlikely(cob_do_sync && ret == 0)) {
                cob_sync (f, cob_do_sync);
        }

        RETURN_STATUS (ret);
}
```

7.34.2.36 void cob_rollback (void)

Definition at line 4283 of file fileio.c.

```
{
        struct file_list        *l;

        for (l = file_cache; l; l = l->next) {
                cob_file_unlock (l->file);
        }
}
```

7.34.2.37 void cob_start (cob_file * *f,* const int *cond,* cob_field * *key,* cob_field * *fnstatus*)

Definition at line 4052 of file fileio.c.

```
{
        int     ret;

        f->flag_read_done = 0;
        f->flag_first_read = 0;

        if (f->flag_nonexistent) {
                RETURN_STATUS (COB_STATUS_23_KEY_NOT_EXISTS);
        }

        if (f->open_mode == COB_OPEN_CLOSED
            || f->open_mode == COB_OPEN_OUTPUT
            || f->open_mode == COB_OPEN_EXTEND
            || f->access_mode == COB_ACCESS_RANDOM) {
                RETURN_STATUS (COB_STATUS_47_INPUT_DENIED);
        }

        ret = fileio_funcs[(int)f->organization]->start (f, cond, key);
        if (ret == COB_STATUS_00_SUCCESS) {
                f->flag_end_of_file = 0;
                f->flag_begin_of_file = 0;
                f->flag_first_read = 1;
        }

        RETURN_STATUS (ret);
}
```

7.34.2.38 void cob_unlock_file (cob_file * f, cob_field * fnstatus)

Definition at line 3727 of file fileio.c.

```
{
        cob_file_unlock (f);
        RETURN_STATUS (COB_STATUS_00_SUCCESS);
}
```

7.34.2.39 void cob_write (cob_file * f, cob_field * rec, const int opt, cob_field * fnstatus)

Definition at line 4153 of file fileio.c.

```
{
        int     ret;

        f->flag_read_done = 0;

        if (f->access_mode == COB_ACCESS_SEQUENTIAL) {
                if (f->open_mode == COB_OPEN_CLOSED
                    || f->open_mode == COB_OPEN_INPUT
                    || f->open_mode == COB_OPEN_I_O) {
                        RETURN_STATUS (COB_STATUS_48_OUTPUT_DENIED);
                }
        } else {
                if (f->open_mode == COB_OPEN_CLOSED
                    || f->open_mode == COB_OPEN_INPUT
                    || f->open_mode == COB_OPEN_EXTEND) {
                        RETURN_STATUS (COB_STATUS_48_OUTPUT_DENIED);
                }
        }

        if (f->record_size) {
                f->record->size = cob_get_int (f->record_size);
        } else {
                f->record->size = rec->size;
        }

        if (f->record->size < f->record_min || f->record_max < f->record->size) {

                RETURN_STATUS (COB_STATUS_44_RECORD_OVERFLOW);
        }

/* RXW
#ifdef  USE_DB41
        if (f->organization != COB_ORG_INDEXED || bdb_env == NULL) {
                opt &= ~COB_WRITE_LOCK;
        }
#endif
*/

        ret = fileio_funcs[(int)f->organization]->write (f, opt);

        if (unlikely(cob_do_sync && ret == 0)) {
                cob_sync (f, cob_do_sync);
        }
```

```
        RETURN_STATUS (ret);
}
```

7.34.3 Variable Documentation

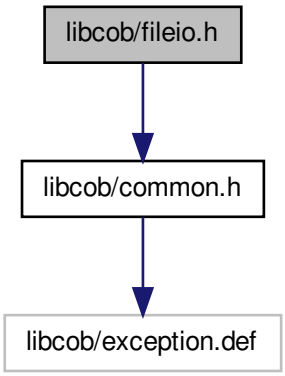
7.34.3.1 cob_file* cob_error_file

Definition at line 197 of file fileio.c.

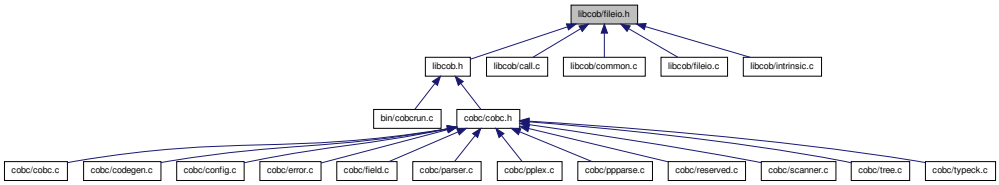
7.35 libcob/fileio.h File Reference

```
#include <libcob/common.h>
```

Include dependency graph for fileio.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct cob_file_key
- struct linage_struct
- struct cob_file
- struct cob_fileio_funcs

Defines

- #define COB_FILE_VERSION 0
- #define COB_EQ 1
- #define COB_LT 2
- #define COB_LE 3
- #define COB_GT 4
- #define COB_GE 5
- #define COB_NE 6
- #define COB_ASCENDING 0
- #define COB_DESCENDING 1
- #define COB_FILE_MODE 0644
- #define COB_ORG_SEQUENTIAL 0
- #define COB_ORG_LINE_SEQUENTIAL 1
- #define COB_ORG_RELATIVE 2
- #define COB_ORG_INDEXED 3
- #define COB_ORG_SORT 4
- #define COB_ORG_MAX 5
- #define COB_ACCESS_SEQUENTIAL 1
- #define COB_ACCESS_DYNAMIC 2
- #define COB_ACCESS_RANDOM 3
- #define COB_SELECT_FILE_STATUS 0x01
- #define COB_SELECT_EXTERNAL 0x02
- #define COB_SELECT_LINAGE 0x04
- #define COB_SELECT_SPLITKEY 0x08
- #define COB_LOCK_EXCLUSIVE 1
- #define COB_LOCK_MANUAL 2
- #define COB_LOCK_AUTOMATIC 4
- #define COB_LOCK_MULTIPLE 8
- #define COB_LOCK_MASK 0x7
- #define COB_OPEN_CLOSED 0
- #define COB_OPEN_INPUT 1
- #define COB_OPEN_OUTPUT 2
- #define COB_OPEN_I_O 3
- #define COB_OPEN_EXTEND 4
- #define COB_OPEN_LOCKED 5
- #define COB_CLOSE_NORMAL 0
- #define COB_CLOSE_LOCK 1
- #define COB_CLOSE_NO_REWIND 2

- #define COB_CLOSE_UNIT 3
- #define COB_CLOSE_UNIT_REMOVAL 4
- #define COB_WRITE_MASK 0x0000ffff
- #define COB_WRITE_LINES 0x00010000
- #define COB_WRITE_PAGE 0x00020000
- #define COB_WRITE_CHANNEL 0x00040000
- #define COB_WRITE_AFTER 0x00100000
- #define COB_WRITE_BEFORE 0x00200000
- #define COB_WRITE_EOP 0x00400000
- #define COB_WRITE_LOCK 0x00800000
- #define COB_READ_NEXT 0x01
- #define COB_READ_PREVIOUS 0x02
- #define COB_READ_FIRST 0x04
- #define COB_READ_LAST 0x08
- #define COB_READ_LOCK 0x10
- #define COB_READ_NO_LOCK 0x20
- #define COB_READ_KEPT_LOCK 0x40
- #define COB_READ_WAIT_LOCK 0x80
- #define COB_READ_IGNORE_LOCK 0x100
- #define COB_STATUS_00_SUCCESS 00
- #define COB_STATUS_02_SUCCESS_DUPLICATE 02
- #define COB_STATUS_04_SUCCESS_INCOMPLETE 04
- #define COB_STATUS_05_SUCCESS_OPTIONAL 05
- #define COB_STATUS_07_SUCCESS_NO_UNIT 07
- #define COB_STATUS_10_END_OF_FILE 10
- #define COB_STATUS_14_OUT_OF_KEY_RANGE 14
- #define COB_STATUS_21_KEY_INVALID 21
- #define COB_STATUS_22_KEY_EXISTS 22
- #define COB_STATUS_23_KEY_NOT_EXISTS 23
- #define COB_STATUS_30_PERMANENT_ERROR 30
- #define COB_STATUS_31_INCONSISTENT_FILENAME 31
- #define COB_STATUS_34_BOUNDARY_VIOLATION 34
- #define COB_STATUS_35_NOT_EXISTS 35
- #define COB_STATUS_37_PERMISSION_DENIED 37
- #define COB_STATUS_38_CLOSED_WITH_LOCK 38
- #define COB_STATUS_39_CONFLICT_ATTRIBUTE 39
- #define COB_STATUS_41_ALREADY_OPEN 41
- #define COB_STATUS_42_NOT_OPEN 42
- #define COB_STATUS_43_READ_NOT_DONE 43
- #define COB_STATUS_44_RECORD_OVERFLOW 44
- #define COB_STATUS_46_READ_ERROR 46
- #define COB_STATUS_47_INPUT_DENIED 47
- #define COB_STATUS_48_OUTPUT_DENIED 48
- #define COB_STATUS_49_I_O_DENIED 49
- #define COB_STATUS_51_RECORD_LOCKED 51
- #define COB_STATUS_52_EOP 52

- #define COB_STATUS_57_I_O_LINAGE 57
- #define COB_STATUS_61_FILE_SHARING 61
- #define COB_STATUS_91_NOT_AVAILABLE 91
- #define COB_LINAGE_INVALID 16384
- #define COB_NOT_CONFIGURED 32768

Functions

- void cob_default_error_handle (void)
- void cob_open (cob_file *, const int, const int, cob_field *)
- void cob_close (cob_file *, const int, cob_field *)
- void cob_read (cob_file *, cob_field *, cob_field *, int)
- void cob_write (cob_file *, cob_field *, const int, cob_field *)
- void cob_rewrite (cob_file *, cob_field *, const int, cob_field *)
- void cob_delete (cob_file *, cob_field *)
- void cob_start (cob_file *, const int, cob_field *, cob_field *)
- void cob_unlock_file (cob_file *, cob_field *)
- void cob_commit (void)
- void cob_rollback (void)
- int CBL_OPEN_FILE (unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *)
- int CBL_CREATE_FILE (unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *)
- int CBL_READ_FILE (unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *)
- int CBL_WRITE_FILE (unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *)
- int CBL_CLOSE_FILE (unsigned char *)
- int CBL_FLUSH_FILE (unsigned char *)
- int CBL_DELETE_FILE (unsigned char *)
- int CBL_COPY_FILE (unsigned char *, unsigned char *)
- int CBL_CHECK_FILE_EXIST (unsigned char *, unsigned char *)
- int CBL_RENAME_FILE (unsigned char *, unsigned char *)
- int CBL_GET_CURRENT_DIR (const int, const int, unsigned char *)
- int CBL_CHANGE_DIR (unsigned char *)
- int CBL_CREATE_DIR (unsigned char *)
- int CBL_DELETE_DIR (unsigned char *)
- int cob_acuw_chdir (unsigned char *, unsigned char *)
- int cob_acuw_mkdir (unsigned char *)
- int cob_acuw_copyfile (unsigned char *, unsigned char *, unsigned char *)
- int cob_acuw_file_info (unsigned char *, unsigned char *)
- int cob_acuw_file_delete (unsigned char *, unsigned char *)
- void cob_file_sort_init (cob_file *, const int, const unsigned char *, void *, cob_field *)
- void cob_file_sort_init_key (cob_file *, const int, cob_field *, size_t)
- void cob_file_sort_close (cob_file *)

- void cob_file_sort_using (cob_file *, cob_file *)
- void cob_file_sort_giving (cob_file *, const size_t,...)
- void cob_file_release (cob_file *)
- void cob_file_return (cob_file *)

Variables

- DLL_EXPIMP cob_file * cob_error_file

7.35.1 Define Documentation

7.35.1.1 #define COB_ACCESS_DYNAMIC 2

Definition at line 53 of file fileio.h.

7.35.1.2 #define COB_ACCESS_RANDOM 3

Definition at line 54 of file fileio.h.

7.35.1.3 #define COB_ACCESS_SEQUENTIAL 1

Definition at line 52 of file fileio.h.

7.35.1.4 #define COB_ASCENDING 0

Definition at line 36 of file fileio.h.

7.35.1.5 #define COB_CLOSE_LOCK 1

Definition at line 83 of file fileio.h.

7.35.1.6 #define COB_CLOSE_NO_REWIND 2

Definition at line 84 of file fileio.h.

7.35.1.7 #define COB_CLOSE_NORMAL 0

Definition at line 82 of file fileio.h.

7.35.1.8 #define COB_CLOSE_UNIT 3

Definition at line 85 of file fileio.h.

7.35.1.9 #define COB_CLOSE_UNIT_REMOVAL 4

Definition at line 86 of file fileio.h.

7.35.1.10 #define COB_DESCENDING 1

Definition at line 37 of file fileio.h.

7.35.1.11 #define COB_EQ 1

Definition at line 29 of file fileio.h.

7.35.1.12 #define COB_FILE_MODE 0644

Definition at line 39 of file fileio.h.

7.35.1.13 #define COB_FILE_VERSION 0

Definition at line 27 of file fileio.h.

7.35.1.14 #define COB_GE 5

Definition at line 33 of file fileio.h.

7.35.1.15 #define COB_GT 4

Definition at line 32 of file fileio.h.

7.35.1.16 #define COB_LE 3

Definition at line 31 of file fileio.h.

7.35.1.17 #define COB_LINAGE_INVALID 16384

Definition at line 146 of file fileio.h.

7.35.1.18 #define COB_LOCK_AUTOMATIC 4

Definition at line 67 of file fileio.h.

7.35.1.19 #define COB_LOCK_EXCLUSIVE 1

Definition at line 65 of file fileio.h.

7.35.1.20 #define COB_LOCK_MANUAL 2

Definition at line 66 of file fileio.h.

7.35.1.21 #define COB_LOCK_MASK 0x7

Definition at line 69 of file fileio.h.

7.35.1.22 #define COB_LOCK_MULTIPLE 8

Definition at line 68 of file fileio.h.

7.35.1.23 #define COB_LT 2

Definition at line 30 of file fileio.h.

7.35.1.24 #define COB_NE 6

Definition at line 34 of file fileio.h.

7.35.1.25 #define COB_NOT_CONFIGURED 32768

Definition at line 148 of file fileio.h.

7.35.1.26 #define COB_OPEN_CLOSED 0

Definition at line 73 of file fileio.h.

7.35.1.27 #define COB_OPEN_EXTEND 4

Definition at line 77 of file fileio.h.

7.35.1.28 #define COB_OPEN_I_O 3

Definition at line 76 of file fileio.h.

7.35.1.29 #define COB_OPEN_INPUT 1

Definition at line 74 of file fileio.h.

7.35.1.30 #define COB_OPEN_LOCKED 5

Definition at line 78 of file fileio.h.

7.35.1.31 #define COB_OPEN_OUTPUT 2

Definition at line 75 of file fileio.h.

7.35.1.32 #define COB_ORG_INDEXED 3

Definition at line 46 of file fileio.h.

7.35.1.33 #define COB_ORG_LINE_SEQUENTIAL 1

Definition at line 44 of file fileio.h.

7.35.1.34 #define COB_ORG_MAX 5

Definition at line 48 of file fileio.h.

7.35.1.35 #define COB_ORG_RELATIVE 2

Definition at line 45 of file fileio.h.

7.35.1.36 #define COB_ORG_SEQUENTIAL 0

Definition at line 43 of file fileio.h.

7.35.1.37 #define COB_ORG_SORT 4

Definition at line 47 of file fileio.h.

7.35.1.38 #define COB_READ_FIRST 0x04

Definition at line 102 of file fileio.h.

7.35.1.39 #define COB_READ_IGNORE_LOCK 0x100

Definition at line 108 of file fileio.h.

7.35.1.40 #define COB_READ_KEPT_LOCK 0x40

Definition at line 106 of file fileio.h.

7.35.1.41 #define COB_READ_LAST 0x08

Definition at line 103 of file fileio.h.

7.35.1.42 #define COB_READ_LOCK 0x10

Definition at line 104 of file fileio.h.

7.35.1.43 #define COB_READ_NEXT 0x01

Definition at line 100 of file fileio.h.

7.35.1.44 #define COB_READ_NO_LOCK 0x20

Definition at line 105 of file fileio.h.

7.35.1.45 #define COB_READ_PREVIOUS 0x02

Definition at line 101 of file fileio.h.

7.35.1.46 #define COB_READ_WAIT_LOCK 0x80

Definition at line 107 of file fileio.h.

7.35.1.47 #define COB_SELECT_EXTERNAL 0x02

Definition at line 59 of file fileio.h.

7.35.1.48 #define COB_SELECT_FILE_STATUS 0x01

Definition at line 58 of file fileio.h.

7.35.1.49 #define COB_SELECT_LINAGE 0x04

Definition at line 60 of file fileio.h.

7.35.1.50 #define COB_SELECT_SPLITKEY 0x08

Definition at line 61 of file fileio.h.

7.35.1.51 #define COB_STATUS_00_SUCCESS 00

Definition at line 112 of file fileio.h.

7.35.1.52 #define COB_STATUS_02_SUCCESS_DUPLICATE 02

Definition at line 113 of file fileio.h.

7.35.1.53 #define COB_STATUS_04_SUCCESS_INCOMPLETE 04

Definition at line 114 of file fileio.h.

7.35.1.54 #define COB_STATUS_05_SUCCESS_OPTIONAL 05

Definition at line 115 of file fileio.h.

7.35.1.55 #define COB_STATUS_07_SUCCESS_NO_UNIT 07

Definition at line 116 of file fileio.h.

7.35.1.56 #define COB_STATUS_10_END_OF_FILE 10

Definition at line 117 of file fileio.h.

7.35.1.57 #define COB_STATUS_14_OUT_OF_KEY_RANGE 14

Definition at line 118 of file fileio.h.

7.35.1.58 #define COB_STATUS_21_KEY_INVALID 21

Definition at line 119 of file fileio.h.

7.35.1.59 #define COB_STATUS_22_KEY_EXISTS 22

Definition at line 120 of file fileio.h.

7.35.1.60 #define COB_STATUS_23_KEY_NOT_EXISTS 23

Definition at line 121 of file fileio.h.

7.35.1.61 #define COB_STATUS_30_PERMANENT_ERROR 30

Definition at line 122 of file fileio.h.

7.35.1.62 #define COB_STATUS_31_INCONSISTENT_FILENAME 31

Definition at line 123 of file fileio.h.

7.35.1.63 #define COB_STATUS_34_BOUNDARY_VIOLATION 34

Definition at line 124 of file fileio.h.

7.35.1.64 #define COB_STATUS_35_NOT_EXISTS 35

Definition at line 125 of file fileio.h.

7.35.1.65 #define COB_STATUS_37_PERMISSION_DENIED 37

Definition at line 126 of file fileio.h.

7.35.1.66 #define COB_STATUS_38_CLOSED_WITH_LOCK 38

Definition at line 127 of file fileio.h.

7.35.1.67 #define COB_STATUS_39_CONFLICT_ATTRIBUTE 39

Definition at line 128 of file fileio.h.

7.35.1.68 #define COB_STATUS_41_ALREADY_OPEN 41

Definition at line 129 of file fileio.h.

7.35.1.69 #define COB_STATUS_42_NOT_OPEN 42

Definition at line 130 of file fileio.h.

7.35.1.70 #define COB_STATUS_43_READ_NOT_DONE 43

Definition at line 131 of file fileio.h.

7.35.1.71 #define COB_STATUS_44_RECORD_OVERFLOW 44

Definition at line 132 of file fileio.h.

7.35.1.72 #define COB_STATUS_46_READ_ERROR 46

Definition at line 133 of file fileio.h.

7.35.1.73 #define COB_STATUS_47_INPUT_DENIED 47

Definition at line 134 of file fileio.h.

7.35.1.74 #define COB_STATUS_48_OUTPUT_DENIED 48

Definition at line 135 of file fileio.h.

7.35.1.75 #define COB_STATUS_49_I_O_DENIED 49

Definition at line 136 of file fileio.h.

7.35.1.76 #define COB_STATUS_51_RECORD_LOCKED 51

Definition at line 137 of file fileio.h.

7.35.1.77 #define COB_STATUS_52_EOP 52

Definition at line 138 of file fileio.h.

7.35.1.78 #define COB_STATUS_57_I_O_LINAGE 57

Definition at line 139 of file fileio.h.

7.35.1.79 #define COB_STATUS_61_FILE_SHARING 61

Definition at line 140 of file fileio.h.

7.35.1.80 #define COB_STATUS_91_NOT_AVAILABLE 91

Definition at line 141 of file fileio.h.

7.35.1.81 #define COB_WRITE_AFTER 0x00100000

Definition at line 94 of file fileio.h.

7.35.1.82 #define COB_WRITE_BEFORE 0x00200000

Definition at line 95 of file fileio.h.

7.35.1.83 #define COB_WRITE_CHANNEL 0x00040000

Definition at line 93 of file fileio.h.

7.35.1.84 #define COB_WRITE_EOP 0x00400000

Definition at line 96 of file fileio.h.

7.35.1.85 #define COB_WRITE_LINES 0x00010000

Definition at line 91 of file fileio.h.

7.35.1.86 #define COB_WRITE_LOCK 0x00800000

Definition at line 97 of file fileio.h.

7.35.1.87 #define COB_WRITE_MASK 0x0000ffff

Definition at line 90 of file fileio.h.

7.35.1.88 #define COB_WRITE_PAGE 0x00020000

Definition at line 92 of file fileio.h.

7.35.2 Function Documentation

7.35.2.1 int CBL_CHANGE_DIR (unsigned char *)

Definition at line 4857 of file fileio.c.

```
{
        char    *fn;
        int     ret;

        COB_CHK_PARMS (CBL_CHANGE_DIR, 1);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        ret = chdir (fn);
        free (fn);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.35.2.2 int CBL_CHECK_FILE_EXIST (unsigned char *, unsigned char *)

Definition at line 4722 of file fileio.c.

```
{
        char            *fn;
        struct tm       *tm;
        long long       sz;
        struct stat     st;
        short           y;
        char            d, m, hh, mm, ss;

        COB_CHK_PARMS (CBL_CHECK_FILE_EXIST, 2);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        if (stat (fn, &st) < 0) {
                free (fn);
                return 35;
        }
        free (fn);
        sz = st.st_size;
        tm = localtime (&st.st_mtime);
        d = (char) tm->tm_mday;
        m = (char) tm->tm_mon + 1;
        y = tm->tm_year + 1900;
        hh = (char) tm->tm_hour;
        mm = (char) tm->tm_min;
        ss = (char) tm->tm_sec;

#ifndef WORDS_BIGENDIAN
```

```
        sz = COB_BSWAP_64 (sz);
        y = COB_BSWAP_16 (y);
#endif
        memcpy (file_info, &sz, 8);
        file_info[8] = d;
        file_info[9] = m;
        memcpy (file_info+10, &y, 2);
        file_info[12] = hh;
        file_info[13] = mm;
        file_info[14] = ss;
        file_info[15] = 0;
        return 0;
}
```

7.35.2.3 int CBL_CLOSE_FILE (unsigned char *)

Definition at line 4631 of file fileio.c.

```
{
        int     fd;

        COB_CHK_PARMS (CBL_CLOSE_FILE, 1);

        memcpy (&fd, file_handle, 4);
        return close (fd);
}
```

7.35.2.4 int CBL_COPY_FILE (unsigned char *, unsigned char *)

Definition at line 4670 of file fileio.c.

```
{
        char    *fn1;
        char    *fn2;
#ifdef  O_BINARY
        int     flag = O_BINARY;
#else
        int     flag = 0;
#endif
        int     ret;
        int     i;
        int     fd1, fd2;

        COB_CHK_PARMS (CBL_COPY_FILE, 2);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        if (!cob_current_module->cob_procedure_parameters[1]) {
                return -1;
        }
        fn1 = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);

        flag |= O_RDONLY;
        fd1 = open (fn1, flag, 0);
```

```
        if (fd1 < 0) {
                free (fn1);
                return -1;
        }
        free (fn1);
        fn2 = cob_str_from_fld (cob_current_module->cob_procedure_parameters[1]);

        flag &= ~O_RDONLY;
        flag |= O_CREAT | O_TRUNC | O_WRONLY;
        fd2 = open (fn2, flag, 0660);
        if (fd2 < 0) {
                close (fd1);
                free (fn2);
                return -1;
        }
        free (fn2);
        ret = 0;
        while ((i = read (fd1, fn1, sizeof(fn1))) > 0) {
                if (write (fd2, fn1, (size_t)i) < 0) {
                        ret = -1;
                        break;
                }
        }
        close (fd1);
        close (fd2);
        return ret;
}
```

7.35.2.5 int CBL_CREATE_DIR (unsigned char *)

Definition at line 4833 of file fileio.c.

```
{
        char    *fn;
        int     ret;

        COB_CHK_PARMS (CBL_CREATE_DIR, 1);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
#ifdef  _WIN32
        ret = mkdir (fn);
#else
        ret = mkdir (fn, 0770);
#endif
        free (fn);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.35.2.6 int CBL_CREATE_FILE (unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *)

Definition at line 4548 of file fileio.c.

```
{
        COB_CHK_PARMS (CBL_CREATE_FILE, 5);

        return open_cbl_file (file_name, file_access, file_handle, O_CREAT | O_TR
      UNC);
}
```

7.35.2.7 int CBL_DELETE_DIR (unsigned char *)

Definition at line 4877 of file fileio.c.

```
{
        char    *fn;
        int     ret;

        COB_CHK_PARMS (CBL_DELETE_DIR, 1);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        ret = rmdir (fn);
        free (fn);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.35.2.8 int CBL_DELETE_FILE (unsigned char *)

Definition at line 4650 of file fileio.c.

```
{
        char    *fn;
        int     ret;

        COB_CHK_PARMS (CBL_DELETE_FILE, 1);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        ret = unlink (fn);
        free (fn);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.35.2.9 int CBL_FLUSH_FILE (unsigned char *)

Definition at line 4642 of file fileio.c.

```
{
        COB_CHK_PARMS (CBL_FLUSH_FILE, 1);

        return 0;
}
```

7.35.2.10 int CBL_GET_CURRENT_DIR (const int , const int , unsigned char *)

Definition at line 4793 of file fileio.c.

```
{
        char    *dirname;
        int     dir_size;
        int     has_space;

        COB_CHK_PARMS (CBL_GET_CURRENT_DIR, 3);

        if (dir_length < 1) {
                return 128;
        }
        if (flags) {
                return 129;
        }
        memset (dir, ' ', (size_t)dir_length);
        dirname = getcwd (NULL, 0);
        if (dirname == NULL) {
                return 128;
        }
        dir_size = (int) strlen (dirname);
        has_space = 0;
        if (strchr (dirname, ' ')) {
                has_space = 2;
        }
        if (dir_size + has_space > dir_length) {
                free (dirname);
                return 128;
        }
        if (has_space) {
                *dir = '"';
                memcpy (&dir[1], dirname, (size_t)dir_size);
                dir[dir_size + 1] = '"';
        } else {
                memcpy (dir, dirname, (size_t)dir_size);
        }
        free (dirname);
        return 0;
}
```

7.35.2.11 int CBL_OPEN_FILE (unsigned char * , unsigned char * , unsigned char * , unsigned char * , unsigned char *)

Definition at line 4538 of file fileio.c.

```
{
        COB_CHK_PARMS (CBL_OPEN_FILE, 5);

        return open_cbl_file (file_name, file_access, file_handle, 0);
}
```

7.35.2.12 int CBL_READ_FILE (unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *)

Definition at line 4558 of file fileio.c.

```
{
        long long       off;
        int             fd;
        int             len;
        int             rc = 0;
        struct stat     st;

        COB_CHK_PARMS (CBL_READ_FILE, 5);

        memcpy (&fd, file_handle, 4);
        memcpy (&off, file_offset, 8);
        memcpy (&len, file_len, 4);
#ifndef WORDS_BIGENDIAN
        off = COB_BSWAP_64 (off);
        len = COB_BSWAP_32 (len);
#endif
        if (lseek (fd, (off_t)off, SEEK_SET) < 0) {
                return -1;
        }
        if (len > 0) {
                rc = read (fd, buf, (size_t)len);
                if (rc < 0) {
                        rc = -1;
                } else if (rc == 0) {
                        rc = 10;
                } else {
                        rc = 0;
                }
        }
        if ((*flags & 0x80) != 0) {
                if (fstat (fd, &st) < 0) {
                        return -1;
                }
                off = st.st_size;
#ifndef WORDS_BIGENDIAN
                off = COB_BSWAP_64 (off);
#endif
                memcpy (file_offset, &off, 8);
        }
        return rc;
}
```

7.35.2.13 int CBL_RENAME_FILE (unsigned char *, unsigned char *)

Definition at line 4767 of file fileio.c.

```
{
        char    *fn1;
        char    *fn2;
        int     ret;

        COB_CHK_PARMS (CBL_RENAME_FILE, 2);

        if (!cob_current_module->cob_procedure_parameters[0]) {
                return -1;
        }
        if (!cob_current_module->cob_procedure_parameters[1]) {
                return -1;
        }
        fn1 = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);

        fn2 = cob_str_from_fld (cob_current_module->cob_procedure_parameters[1]);

        ret = rename (fn1, fn2);
        free (fn1);
        free (fn2);
        if (ret) {
                return 128;
        }
        return 0;
}
```

7.35.2.14 int CBL_WRITE_FILE (unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *)

Definition at line 4603 of file fileio.c.

```
{
        long long       off;
        int             fd;
        int             len;
        int             rc;

        COB_CHK_PARMS (CBL_WRITE_FILE, 5);

        memcpy (&fd, file_handle, 4);
        memcpy (&off, file_offset, 8);
        memcpy (&len, file_len, 4);
#ifndef WORDS_BIGENDIAN
        off = COB_BSWAP_64 (off);
        len = COB_BSWAP_32 (len);
#endif
        if (lseek (fd, (off_t)off, SEEK_SET) < 0) {
                return -1;
        }
        rc = write (fd, buf, (size_t)len);
        if (rc < 0) {
                return 30;
        }
        return 0;
}
```

7.35.2.15 int cob_acuw_chdir (unsigned char * , unsigned char *)

Definition at line 4911 of file fileio.c.

```
{
        int             ret;

        COB_CHK_PARMS (C$CHDIR, 2);

        ret = CBL_CHANGE_DIR (dir);
        if (ret < 0) {
                ret = 128;
        }
        cob_set_int (cob_current_module->cob_procedure_parameters[1], ret);
        return ret;
}
```

7.35.2.16 int cob_acuw_copyfile (unsigned char * , unsigned char * , unsigned char *)

Definition at line 4926 of file fileio.c.

```
{
        int             ret = 128;

        /* RXW - Type is not yet evaluated */

        COB_CHK_PARMS (C$COPY, 3);

        if (cob_call_params < 3) {
                return 128;
        }
        ret = CBL_COPY_FILE (fname1, fname2);
        if (ret < 0) {
                ret = 128;
        }
        return ret;
}
```

7.35.2.17 int cob_acuw_file_delete (unsigned char * , unsigned char *)

Definition at line 4993 of file fileio.c.

```
{
        int     ret;

        /* RXW - Type is not yet evaluated */
        COB_CHK_PARMS (C$DELETE, 2);

        if (cob_call_params < 2 || !cob_current_module->cob_procedure_parameters[
      0]) {
                return 128;
        }
        ret = CBL_DELETE_FILE (file_name);
        if (ret < 0) {
```

```
                ret = 128;
        }
        return ret;
}
```

7.35.2.18 int cob_acuw_file_info (unsigned char *, unsigned char *)

Definition at line 4945 of file fileio.c.

```
{
        char                    *fn;
        struct tm               *tm;
        unsigned long long      sz;
        unsigned int            dt;
        short                   y;
        short                   d, m, hh, mm, ss;
        struct stat             st;

        COB_CHK_PARMS (C$FILEINFO, 2);

        if (cob_call_params < 2 || !cob_current_module->cob_procedure_parameters[
      0]) {
                return 128;
        }
        fn = cob_str_from_fld (cob_current_module->cob_procedure_parameters[0]);
        if (stat (fn, &st) < 0) {
                free (fn);
                return 35;
        }
        free (fn);
        sz = st.st_size;
        tm = localtime (&st.st_mtime);
        d = tm->tm_mday;
        m = tm->tm_mon + 1;
        y = tm->tm_year + 1900;
        hh = tm->tm_hour;
        mm = tm->tm_min;
        ss = tm->tm_sec;

#ifndef WORDS_BIGENDIAN
        sz = COB_BSWAP_64 (sz);
#endif
        memcpy (file_info, &sz, 8);
        dt = (y * 10000) + (m * 100) + d;
#ifndef WORDS_BIGENDIAN
        dt = COB_BSWAP_32 (dt);
#endif
        memcpy (file_info + 8, &dt, 4);
        dt = (hh * 1000000) + (mm * 10000) + (ss * 100);
#ifndef WORDS_BIGENDIAN
        dt = COB_BSWAP_32 (dt);
#endif
        memcpy (file_info + 12, &dt, 4);
        return 0;
}
```

7.35.2.19 int cob_acuw_mkdir (unsigned char *)

Definition at line 4897 of file fileio.c.

```
{
        int             ret;

        COB_CHK_PARMS (C$MAKEDIR, 1);

        ret = CBL_CREATE_DIR (dir);
        if (ret < 0) {
                ret = 128;
        }
        return ret;
}
```

7.35.2.20 void cob_close (cob_file *, const int, cob_field *)

Definition at line 3987 of file fileio.c.

```
{
        int     ret;

        f->flag_read_done = 0;

        if (f->special) {
                f->open_mode = COB_OPEN_CLOSED;
                RETURN_STATUS (COB_STATUS_00_SUCCESS);
        }
        if (f->open_mode == COB_OPEN_CLOSED) {
                RETURN_STATUS (COB_STATUS_42_NOT_OPEN);
        }

        if (f->flag_nonexistent) {
                ret = COB_STATUS_00_SUCCESS;
        } else {
#ifdef  WITH_SEQRA_EXTFH
                if (f->organization != COB_ORG_INDEXED) {
                        ret = extfh_cob_file_close (f, opt);
                } else {
#endif
                        ret = fileio_funcs[(int)f->organization]->close (f, opt);

#ifdef  WITH_SEQRA_EXTFH
                }
#endif
        }

        if (ret == COB_STATUS_00_SUCCESS) {
                switch (opt) {
                case COB_CLOSE_LOCK:
                        f->open_mode = COB_OPEN_LOCKED;
                        break;
                default:
                        f->open_mode = COB_OPEN_CLOSED;
                        break;
                }
        }
```

```
        RETURN_STATUS (ret);
}
```

7.35.2.21 void cob_commit (void)

Definition at line 4273 of file fileio.c.

```
{
        struct file_list        *l;

        for (l = file_cache; l; l = l->next) {
                cob_file_unlock (l->file);
        }
}
```

7.35.2.22 void cob_default_error_handle (void)

Definition at line 4293 of file fileio.c.

```
{
        const char      *msg;
        unsigned char   *file_status;
        char            *filename;
        int             status;

        file_status = cob_error_file->file_status;
        status = cob_d2i(file_status[0]) * 10 + cob_d2i(file_status[1]);
        switch (status) {
        case COB_STATUS_10_END_OF_FILE:
                msg = "End of file";
                break;
        case COB_STATUS_14_OUT_OF_KEY_RANGE:
                msg = "Key out of range";
                break;
        case COB_STATUS_21_KEY_INVALID:
                msg = "Key order not ascending";
                break;
        case COB_STATUS_22_KEY_EXISTS:
                msg = "Record key already exists";
                break;
        case COB_STATUS_23_KEY_NOT_EXISTS:
                msg = "Record key does not exist";
                break;
        case COB_STATUS_30_PERMANENT_ERROR:
                msg = "Permanent file error";
                break;
        case COB_STATUS_35_NOT_EXISTS:
                msg = "File does not exist";
                break;
        case COB_STATUS_37_PERMISSION_DENIED:
                msg = "Permission denied";
                break;
        case COB_STATUS_41_ALREADY_OPEN:
                msg = "File already open";
```

```
            break;
        case COB_STATUS_42_NOT_OPEN:
            msg = "File not open";
            break;
        case COB_STATUS_43_READ_NOT_DONE:
            msg = "READ must be executed first";
            break;
        case COB_STATUS_44_RECORD_OVERFLOW:
            msg = "Record overflow";
            break;
        case COB_STATUS_46_READ_ERROR:
            msg = "Failed to read";
            break;
        case COB_STATUS_47_INPUT_DENIED:
            msg = "READ/START not allowed";
            break;
        case COB_STATUS_48_OUTPUT_DENIED:
            msg = "WRITE not allowed";
            break;
        case COB_STATUS_49_I_O_DENIED:
            msg = "DELETE/REWRITE not allowed";
            break;
        case COB_STATUS_51_RECORD_LOCKED:
            msg = "Record locked by another file connector";
            break;
        case COB_STATUS_52_EOP:
            msg = "A page overflow condition occurred";
            break;
        case COB_STATUS_57_I_O_LINAGE:
            msg = "LINAGE values invalid";
            break;
        case COB_STATUS_61_FILE_SHARING:
            msg = "File sharing conflict";
            break;
        case COB_STATUS_91_NOT_AVAILABLE:
            msg = "Runtime library is not configured for this operation";
            break;
        default:
            msg = "Unknown file error";
            break;
        }

        filename = cob_malloc (COB_MEDIUM_BUFF);
        cob_field_to_string (cob_error_file->assign, filename);
        cob_runtime_error ("%s (STATUS = %02d) File : '%s'", msg,
                           status, filename);
        free (filename);
}
```

7.35.2.23 void cob_delete (cob_file *, cob_field *)

Definition at line 4247 of file fileio.c.

```
{
        int     ret;
        int     read_done = f->flag_read_done;

        f->flag_read_done = 0;
```

```
        if (unlikely(f->open_mode == COB_OPEN_CLOSED ||
            f->open_mode != COB_OPEN_I_O)) {
                RETURN_STATUS (COB_STATUS_49_I_O_DENIED);
        }

        if (f->access_mode == COB_ACCESS_SEQUENTIAL && !read_done) {
                RETURN_STATUS (COB_STATUS_43_READ_NOT_DONE);
        }

        ret = fileio_funcs[(int)f->organization]->fdelete (f);

        if (unlikely(cob_do_sync && ret == 0)) {
                cob_sync (f, cob_do_sync);
        }

        RETURN_STATUS (ret);
}
```

7.35.2.24 void cob_file_release (cob_file *)

Definition at line 5654 of file fileio.c.

```
{
        struct cobsort  *hp;
        cob_field       *fnstatus = NULL;
        int             ret;

        hp = f->file;
        if (likely(hp)) {
                fnstatus = hp->fnstatus;
        }
        ret = cob_file_sort_submit (f, f->record->data);
        switch (ret) {
        case 0:
                RETURN_STATUS (COB_STATUS_00_SUCCESS);
                break;
        default:
                if (likely(hp)) {
                        *(int *)(hp->sort_return) = 16;
                }
                RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
                break;
        }
}
```

7.35.2.25 void cob_file_return (cob_file *)

Definition at line 5679 of file fileio.c.

```
{
        struct cobsort  *hp;
        cob_field       *fnstatus = NULL;
        int             ret;

        hp = f->file;
```

```
        if (likely(hp)) {
                fnstatus = hp->fnstatus;
        }
        ret = cob_file_sort_retrieve (f, f->record->data);
        switch (ret) {
        case 0:
                RETURN_STATUS (COB_STATUS_00_SUCCESS);
                break;
        case COBSORTEND:
                RETURN_STATUS (COB_STATUS_10_END_OF_FILE);
                break;
        default:
                if (likely(hp)) {
                        *(int *)(hp->sort_return) = 16;
                }
                RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
                break;
        }
}
```

7.35.2.26 void cob_file_sort_close (cob_file *)

Definition at line 5631 of file fileio.c.

```
{
        struct cobsort  *hp;
        cob_field       *fnstatus = NULL;
        size_t          i;

        hp = f->file;
        if (likely(hp)) {
                fnstatus = hp->fnstatus;
                cob_free_list (hp->empty);
                for (i = 0; i < 4; i++) {
                        cob_free_list (hp->queue[i].first);
                        if (hp->file[i].fp != NULL) {
                                fclose (hp->file[i].fp);
                        }
                }
                free (hp);
        }
        f->file = NULL;
        RETURN_STATUS (COB_STATUS_00_SUCCESS);
}
```

7.35.2.27 void cob_file_sort_giving (cob_file *, const size_t, ...)

Definition at line 5544 of file fileio.c.

```
{
        cob_file        **fbase;
        struct cobsort  *hp;
        size_t          i;
        int             ret;
        int             opt;
```

```
        va_list         args;

        fbase = cob_malloc (varcnt * sizeof(cob_file *));
        va_start (args, varcnt);
        for (i = 0; i < varcnt; i++) {
                fbase[i] = va_arg (args, cob_file *);
        }
        va_end (args);
        for (i = 0; i < varcnt; i++) {
                cob_open (fbase[i], COB_OPEN_OUTPUT, 0, NULL);
        }
        while (1) {
                ret = cob_file_sort_retrieve (sort_file, sort_file->record->data)
     ;
                if (ret) {
                        if (ret == COBSORTEND) {
                                sort_file->file_status[0] = '1';
                                sort_file->file_status[1] = '0';
                        } else {
                                hp = sort_file->file;
                                *(int *)(hp->sort_return) = 16;
                                sort_file->file_status[0] = '3';
                                sort_file->file_status[1] = '0';
                        }
                        break;
                }
                for (i = 0; i < varcnt; i++) {
                        if (fbase[i]->special ||
                            fbase[i]->organization == COB_ORG_LINE_SEQUENTIAL) {
                                opt = COB_WRITE_BEFORE | COB_WRITE_LINES | 1;
                        } else {
                                opt = 0;
                        }
                        cob_copy_check (fbase[i], sort_file);
                        cob_write (fbase[i], fbase[i]->record, opt, NULL);
                }
        }
        for (i = 0; i < varcnt; i++) {
                cob_close (fbase[i], COB_CLOSE_NORMAL, NULL);
        }
        free (fbase);
}
```

7.35.2.28 void cob_file_sort_init (cob_file * , const int , const unsigned char * , void * , cob_field *)

Definition at line 5594 of file fileio.c.

```
{
        struct cobsort  *p;

        p = cob_malloc (sizeof (struct cobsort));
        p->fnstatus = fnstatus;
        p->size = f->record_max;
        p->r_size = f->record_max + sizeof(size_t);
        p->w_size = f->record_max + sizeof(size_t) + 1;
        p->pointer = f;
        p->sort_return = sort_return;
        *(int *)sort_return = 0;
```

```
        p->memory = (size_t)cob_sort_memory / (p->size + sizeof(struct cobitem));

        f->file = p;
        f->keys = cob_malloc (sizeof (struct cob_file_key) * nkeys);
        f->nkeys = 0;
        if (collating_sequence) {
                f->sort_collating = collating_sequence;
        } else {
                f->sort_collating = cob_current_module->collating_sequence;
        }
        RETURN_STATUS (COB_STATUS_00_SUCCESS);
}
```

7.35.2.29 void cob_file_sort_init_key (cob_file *, const int, cob_field *, size_t)

Definition at line 5621 of file fileio.c.

```
{
        f->keys[f->nkeys].flag = flag;
        f->keys[f->nkeys].field = field;
        f->keys[f->nkeys].offset = offset;
        f->nkeys++;
}
```

7.35.2.30 void cob_file_sort_using (cob_file *, cob_file *)

Definition at line 5524 of file fileio.c.

```
{
        int             ret;

        cob_open (data_file, COB_OPEN_INPUT, 0, NULL);
        while (1) {
                cob_read (data_file, NULL, NULL, COB_READ_NEXT);
                if (data_file->file_status[0] != '0') {
                        break;
                }
                cob_copy_check (sort_file, data_file);
                ret = cob_file_sort_submit (sort_file, sort_file->record->data);
                if (ret) {
                        break;
                }
        }
        cob_close (data_file, COB_CLOSE_NORMAL, NULL);
}
```

7.35.2.31 void cob_open (cob_file *, const int, const int, cob_field *)

Definition at line 3734 of file fileio.c.

```
{
        char            *p;
```

```
	char		*src;
	char		*dst;
	size_t		i;
	size_t		simple;
	int		was_not_exist = 0;
	struct stat	st;

	f->flag_read_done = 0;

	/* file was previously closed with lock */
	if (f->open_mode == COB_OPEN_LOCKED) {
		RETURN_STATUS (COB_STATUS_38_CLOSED_WITH_LOCK);
	}

	/* file is already open */
	if (f->open_mode != COB_OPEN_CLOSED) {
		RETURN_STATUS (COB_STATUS_41_ALREADY_OPEN);
	}

	f->last_open_mode = mode;
	f->flag_nonexistent = 0;
	f->flag_end_of_file = 0;
	f->flag_begin_of_file = 0;
	f->flag_first_read = 2;

	if (f->special) {
		if (f->special == 1) {
			if (mode != COB_OPEN_INPUT) {
				RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
			}
			f->file = stdin;
			f->open_mode = mode;
			RETURN_STATUS (COB_STATUS_00_SUCCESS);
		} else {
			if (mode != COB_OPEN_OUTPUT) {
				RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
			}
			f->file = stdout;
			f->open_mode = mode;
			RETURN_STATUS (COB_STATUS_00_SUCCESS);
		}
	}

	/* obtain the file name */
	cob_field_to_string (f->assign, file_open_name);

#ifdef	WITH_INDEX_EXTFH
	if (f->organization == COB_ORG_INDEXED) {
		int	ret;

		ret = extfh_indexed_locate (f, file_open_name);
		switch (ret) {
		case COB_NOT_CONFIGURED:
			/* EXTFH requires OC to process the filename */
			break;
		case COB_STATUS_00_SUCCESS:
			/* EXTFH recognized the file */
			goto file_available;
		default:
			/* EXTFH detected an error */
			RETURN_STATUS (ret);
		}
```

```
        }
#endif  /* WITH_INDEX_EXTFH */

#ifdef  WITH_SEQRA_EXTFH
        if (f->organization != COB_ORG_INDEXED) {
                int     ret;

                ret = extfh_seqra_locate (f, file_open_name);
                switch (ret) {
                case COB_NOT_CONFIGURED:
                        /* EXTFH requires OC to process the filename */
                        break;
                case COB_STATUS_00_SUCCESS:
                        /* EXTFH recognized the file */
                        goto file_available;
                default:
                        /* EXTFH detected an error */
                        RETURN_STATUS (ret);
                }
        }
#endif  /* WITH_SEQRA_EXTFH */

        if (cob_current_module->flag_filename_mapping) {
                src = file_open_name;
                dst = file_open_buff;
                simple = 1;
                /* expand envoronment variables */
                /* ex. "$TMPDIR/foo" -> "/tmp/foo" */
                while (*src) {
                        if (!isalnum (*src) && *src != '_' && *src != '-') {
                                simple = 0;
                        }
                        if (*src == '$') {
                                for (i = 1; ; i++) {
                                        if (!isalnum (src[i]) && src[i] != '_' &&
       *src != '-') {
                                                break;
                                        }
                                }
                                memcpy (file_open_env, src + 1, i - 1);
                                file_open_env[i - 1] = 0;
                                if ((p = getenv (file_open_env)) != NULL) {
                                        strcpy (dst, p);
                                        dst += strlen (p);
                                }
                                src += i;
                        } else {
                                *dst++ = *src++;
                        }
                }
                *dst = 0;
                strncpy (file_open_name, file_open_buff, COB_SMALL_MAX);

                /* resolve by environment variables */
                /* ex. "TMPFILE" -> DD_TMPFILE, dd_TMPFILE, or TMPFILE */
                if (simple) {
                        for (i = 0; i < NUM_PREFIX; i++) {
                                snprintf (file_open_buff, COB_SMALL_MAX, "%s%s",
                                          prefix[i], file_open_name);
                                if ((p = getenv (file_open_buff)) != NULL) {
                                        strncpy (file_open_name, p,
      COB_SMALL_MAX);
```

```
                                        break;
                                }
                        }
                        if (i == NUM_PREFIX && cob_file_path) {
                                snprintf (file_open_buff, COB_SMALL_MAX, "%s/%s",

                                          cob_file_path, file_open_name);
                                strncpy (file_open_name, file_open_buff,
                                         COB_SMALL_MAX);
                        }
                }
        }
        /* check if the file exists */
#ifdef  USE_DB41
        if (f->organization == COB_ORG_INDEXED) {
                if ((bdb_env && bdb_nofile (file_open_name)) ||
                     (!bdb_env && stat (file_open_name, &st) == -1 && errno == EN
      OENT)) {
                        was_not_exist = 1;
                        if (mode != COB_OPEN_OUTPUT && f->flag_optional == 0) {
                                RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                        }
                }
        } else if (stat (file_open_name, &st) == -1 && errno == ENOENT) {
#else   /* USE_DB41 */

#if     defined(WITH_CISAM) || defined(WITH_DISAM) || defined(WITH_VBISAM)
        if (f->organization == COB_ORG_INDEXED) {
                strncpy (file_open_buff, file_open_name, COB_SMALL_MAX);
                strcat (file_open_buff, ".idx");
                if (stat (file_open_buff, &st) == -1 && errno == ENOENT) {
                        was_not_exist = 1;
                        if (mode != COB_OPEN_OUTPUT && f->flag_optional == 0) {
                                RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                        }
                }
                strncpy (file_open_buff, file_open_name, COB_SMALL_MAX);
                strcat (file_open_buff, ".dat");
                if (stat (file_open_buff, &st) == -1 && errno == ENOENT) {
                        was_not_exist = 1;
                        if (mode != COB_OPEN_OUTPUT && f->flag_optional == 0) {
                                RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                        }
                }
        } else if (stat (file_open_name, &st) == -1 && errno == ENOENT) {
#else   /* WITH_CISAM || WITH_DISAM || WITH_VBISAM */
        if (stat (file_open_name, &st) == -1 && errno == ENOENT) {
#endif  /* WITH_CISAM || WITH_DISAM || WITH_VBISAM */

#endif  /* USE_DB41 */

                was_not_exist = 1;
                if (mode != COB_OPEN_OUTPUT && f->flag_optional == 0) {
                        RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                }
        }

#if     defined(WITH_INDEX_EXTFH) || defined(WITH_SEQRA_EXTFH)
file_available:
#endif  /* WITH_INDEX_EXTFH || WITH_SEQRA_EXTFH */

        cob_cache_file (f);
```

```
        /* open the file */
#ifdef  WITH_SEQRA_EXTFH
        if (f->organization != COB_ORG_INDEXED) {
                int     ret;

                ret = extfh_cob_file_open (f, file_open_name, mode, sharing);
                switch (ret) {
                case COB_STATUS_00_SUCCESS:
                        f->open_mode = mode;
                        break;
                case COB_STATUS_35_NOT_EXISTS:
                        if (f->flag_optional) {
                                f->open_mode = mode;
                                f->flag_nonexistent = 1;
                                f->flag_end_of_file = 1;
                                f->flag_begin_of_file = 1;
                                RETURN_STATUS (COB_STATUS_05_SUCCESS_OPTIONAL);
                        }
                        break;
                }
                RETURN_STATUS (ret);
        }
#endif
#if     defined(WITH_CISAM) || defined(WITH_DISAM) || defined(WITH_VBISAM)
        if (f->organization == COB_ORG_INDEXED) {
                /* Do this here to avoid mangling of the status in the 'switch' b
      elow */
                RETURN_STATUS (fileio_funcs[(int)f->organization]->open (f, file_
      open_name, mode, sharing));
        }
#endif
        switch (fileio_funcs[(int)f->organization]->open (f, file_open_name, mode
      , sharing)) {
        case 0:
                f->open_mode = mode;
                if (f->flag_optional && was_not_exist) {
                        RETURN_STATUS (COB_STATUS_05_SUCCESS_OPTIONAL);
                } else {
                        RETURN_STATUS (COB_STATUS_00_SUCCESS);
                }
        case ENOENT:
                if (mode == COB_OPEN_EXTEND || mode == COB_OPEN_OUTPUT) {
                        RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
                }
                if (f->flag_optional) {
                        f->open_mode = mode;
                        f->flag_nonexistent = 1;
                        f->flag_end_of_file = 1;
                        f->flag_begin_of_file = 1;
                        RETURN_STATUS (COB_STATUS_05_SUCCESS_OPTIONAL);
                } else {
                        RETURN_STATUS (COB_STATUS_35_NOT_EXISTS);
                }
        case EACCES:
        case EISDIR:
        case EROFS:
                RETURN_STATUS (COB_STATUS_37_PERMISSION_DENIED);
        case EAGAIN:
        case COB_STATUS_61_FILE_SHARING:
                RETURN_STATUS (COB_STATUS_61_FILE_SHARING);
        case COB_STATUS_91_NOT_AVAILABLE:
```

```
                RETURN_STATUS (COB_STATUS_91_NOT_AVAILABLE);
        case COB_LINAGE_INVALID:
                RETURN_STATUS (COB_STATUS_57_I_O_LINAGE);
        default:
                RETURN_STATUS (COB_STATUS_30_PERMANENT_ERROR);
        }
}
```

7.35.2.32 void cob_read (cob_file *, cob_field *, cob_field *, int)

Definition at line 4081 of file fileio.c.

```
{
        int     ret;

        f->flag_read_done = 0;

        if (unlikely(f->flag_nonexistent)) {
                if (f->flag_first_read == 0) {
                        RETURN_STATUS (COB_STATUS_23_KEY_NOT_EXISTS);
                }
                f->flag_first_read = 0;
                RETURN_STATUS (COB_STATUS_10_END_OF_FILE);
        }

        /* sequential read at the end of file is an error */
        if (key == NULL) {
                if (f->flag_end_of_file && !(read_opts & COB_READ_PREVIOUS)) {
                        RETURN_STATUS (COB_STATUS_46_READ_ERROR);
                }
                if (f->flag_begin_of_file && (read_opts & COB_READ_PREVIOUS)) {
                        RETURN_STATUS (COB_STATUS_46_READ_ERROR);
                }
        }

        if (unlikely(f->open_mode == COB_OPEN_CLOSED
            || f->open_mode == COB_OPEN_OUTPUT
            || f->open_mode == COB_OPEN_EXTEND)) {
                RETURN_STATUS (COB_STATUS_47_INPUT_DENIED);
        }

#ifdef  USE_DB41
        if (f->organization == COB_ORG_INDEXED && bdb_env != NULL) {
                if (f->open_mode != COB_OPEN_I_O  ||
                    (f->lock_mode & COB_LOCK_EXCLUSIVE)) {
                        read_opts &= ~COB_READ_LOCK;
                } else if ((f->lock_mode & COB_LOCK_AUTOMATIC) &&
                   !(read_opts & COB_READ_NO_LOCK)) {
                        read_opts |= COB_READ_LOCK;
                }
        } else {
                read_opts &= ~COB_READ_LOCK;
        }
#endif
        if (key) {
                ret = fileio_funcs[(int)f->organization]->read (f, key, read_opts
      );
        } else {
                ret = fileio_funcs[(int)f->organization]->read_next (f, read_opts
```

```
      );
        }

        switch (ret) {
        case COB_STATUS_00_SUCCESS:
                f->flag_first_read = 0;
                f->flag_read_done = 1;
                f->flag_end_of_file = 0;
                f->flag_begin_of_file = 0;
                if (f->record_size && f->organization != COB_ORG_LINE_SEQUENTIAL)
       {
                        cob_set_int (f->record_size, (int) f->record->size);
                }
                break;
        case COB_STATUS_10_END_OF_FILE:
                if (read_opts & COB_READ_PREVIOUS) {
                        f->flag_begin_of_file = 1;
                } else {
                        f->flag_end_of_file = 1;
                }
                break;
        }

        RETURN_STATUS (ret);
}
```

7.35.2.33 void cob_rewrite (cob_file *, cob_field *, const int, cob_field *)

Definition at line 4201 of file fileio.c.

```
{
        int     ret;
        int     read_done = f->flag_read_done;

        f->flag_read_done = 0;

        if (unlikely(f->open_mode == COB_OPEN_CLOSED ||
            f->open_mode != COB_OPEN_I_O)) {
                RETURN_STATUS (COB_STATUS_49_I_O_DENIED);
        }

        if (f->access_mode == COB_ACCESS_SEQUENTIAL && !read_done) {
                RETURN_STATUS (COB_STATUS_43_READ_NOT_DONE);
        }

        if (f->organization == COB_ORG_SEQUENTIAL) {
                if (f->record->size != rec->size) {
                        RETURN_STATUS (COB_STATUS_44_RECORD_OVERFLOW);
                }

                if (f->record_size) {
                        if (f->record->size != (size_t)cob_get_int (f->record_siz
      e)) {
                                RETURN_STATUS (COB_STATUS_44_RECORD_OVERFLOW);
                        }
                }
        }

/* RXW
```

```
#ifdef  USE_DB41
        if (f->organization != COB_ORG_INDEXED || bdb_env == NULL) {
                opt &= ~COB_WRITE_LOCK;
        }
#endif
*/

        ret = fileio_funcs[(int)f->organization]->rewrite (f, opt);

        if (unlikely(cob_do_sync && ret == 0)) {
                cob_sync (f, cob_do_sync);
        }

        RETURN_STATUS (ret);
}
```

7.35.2.34 void cob_rollback (void)

Definition at line 4283 of file fileio.c.

```
{
        struct file_list        *l;

        for (l = file_cache; l; l = l->next) {
                cob_file_unlock (l->file);
        }
}
```

7.35.2.35 void cob_start (cob_file *, const int, cob_field *, cob_field *)

Definition at line 4052 of file fileio.c.

```
{
        int     ret;

        f->flag_read_done = 0;
        f->flag_first_read = 0;

        if (f->flag_nonexistent) {
                RETURN_STATUS (COB_STATUS_23_KEY_NOT_EXISTS);
        }

        if (f->open_mode == COB_OPEN_CLOSED
            || f->open_mode == COB_OPEN_OUTPUT
            || f->open_mode == COB_OPEN_EXTEND
            || f->access_mode == COB_ACCESS_RANDOM) {
                RETURN_STATUS (COB_STATUS_47_INPUT_DENIED);
        }

        ret = fileio_funcs[(int)f->organization]->start (f, cond, key);
        if (ret == COB_STATUS_00_SUCCESS) {
                f->flag_end_of_file = 0;
                f->flag_begin_of_file = 0;
                f->flag_first_read = 1;
        }
```

```
        RETURN_STATUS (ret);
}
```

7.35.2.36 void cob_unlock_file (cob_file *, cob_field *)

Definition at line 3727 of file fileio.c.

```
{
        cob_file_unlock (f);
        RETURN_STATUS (COB_STATUS_00_SUCCESS);
}
```

7.35.2.37 void cob_write (cob_file *, cob_field *, const int, cob_field *)

Definition at line 4153 of file fileio.c.

```
{
        int     ret;

        f->flag_read_done = 0;

        if (f->access_mode == COB_ACCESS_SEQUENTIAL) {
                if (f->open_mode == COB_OPEN_CLOSED
                    || f->open_mode == COB_OPEN_INPUT
                    || f->open_mode == COB_OPEN_I_O) {
                        RETURN_STATUS (COB_STATUS_48_OUTPUT_DENIED);
                }
        } else {
                if (f->open_mode == COB_OPEN_CLOSED
                    || f->open_mode == COB_OPEN_INPUT
                    || f->open_mode == COB_OPEN_EXTEND) {
                        RETURN_STATUS (COB_STATUS_48_OUTPUT_DENIED);
                }
        }

        if (f->record_size) {
                f->record->size = cob_get_int (f->record_size);
        } else {
                f->record->size = rec->size;
        }

        if (f->record->size < f->record_min || f->record_max < f->record->size) {

                RETURN_STATUS (COB_STATUS_44_RECORD_OVERFLOW);
        }

/* RXW
#ifdef  USE_DB41
        if (f->organization != COB_ORG_INDEXED || bdb_env == NULL) {
                opt &= ~COB_WRITE_LOCK;
        }
#endif
*/
```

```
        ret = fileio_funcs[(int)f->organization]->write (f, opt);

        if (unlikely(cob_do_sync && ret == 0)) {
                cob_sync (f, cob_do_sync);
        }

        RETURN_STATUS (ret);
}
```

7.35.3 Variable Documentation

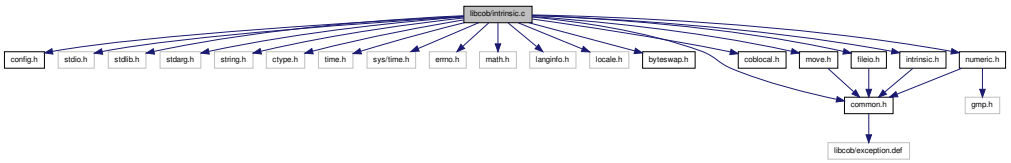
7.35.3.1 DLL_EXPIMP cob_file* cob_error_file

Definition at line 197 of file fileio.c.

7.36 libcob/intrinsic.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <sys/time.h>
#include <errno.h>
#include <math.h>
#include <langinfo.h>
#include <locale.h>
#include "byteswap.h"
#include "common.h"
#include "coblocal.h"
#include "move.h"
#include "numeric.h"
#include "fileio.h"
#include "intrinsic.h"
```

Include dependency graph for intrinsic.c:



Defines

- #define DEPTH_LEVEL 8
- #define COB_FIELD_INIT(x, y, z)

Functions

- cob_field * cob_intr_binop (cob_field *f1, int op, cob_field *f2)
- cob_field * cob_intr_length (cob_field *srcfield)
- cob_field * cob_intr_integer (cob_field *srcfield)
- cob_field * cob_intr_integer_part (cob_field *srcfield)
- cob_field * cob_intr_fraction_part (cob_field *srcfield)
- cob_field * cob_intr_sign (cob_field *srcfield)
- cob_field * cob_intr_upper_case (const int offset, const int length, cob_field *srcfield)
- cob_field * cob_intr_lower_case (const int offset, const int length, cob_field *srcfield)
- cob_field * cob_intr_reverse (const int offset, const int length, cob_field *srcfield)
- cob_field * cob_intr_concatenate (const int offset, const int length, const int params,...)
- cob_field * cob_intr_substitute (const int offset, const int length, const int params,...)
- cob_field * cob_intr_substitute_case (const int offset, const int length, const int params,...)
- cob_field * cob_intr_trim (const int offset, const int length, cob_field *srcfield, const int direction)
- cob_field * cob_intr_exception_file (void)
- cob_field * cob_intr_exception_location (void)
- cob_field * cob_intr_exception_status (void)
- cob_field * cob_intr_exception_statement (void)
- cob_field * cob_intr_when_compiled (const int offset, const int length, cob_field *f)
- cob_field * cob_intr_current_date (const int offset, const int length)
- cob_field * cob_intr_char (cob_field *srcfield)
- cob_field * cob_intr_ord (cob_field *srcfield)
- cob_field * cob_intr_stored_char_length (cob_field *srcfield)
- cob_field * cob_intr_combined_datetime (cob_field *srcdays, cob_field *srctime)
- cob_field * cob_intr_date_of_integer (cob_field *srcdays)
- cob_field * cob_intr_day_of_integer (cob_field *srcdays)

- cob_field * cob_intr_integer_of_date (cob_field *srcfield)
- cob_field * cob_intr_integer_of_day (cob_field *srcfield)
- cob_field * cob_intr_test_date_yyyymmdd (cob_field *srcfield)
- cob_field * cob_intr_test_day_yyyyddd (cob_field *srcfield)
- cob_field * cob_intr_factorial (cob_field *srcfield)
- cob_field * cob_intr_exp (cob_field *srcfield)
- cob_field * cob_intr_exp10 (cob_field *srcfield)
- cob_field * cob_intr_abs (cob_field *srcfield)
- cob_field * cob_intr_acos (cob_field *srcfield)
- cob_field * cob_intr_asin (cob_field *srcfield)
- cob_field * cob_intr_atan (cob_field *srcfield)
- cob_field * cob_intr_cos (cob_field *srcfield)
- cob_field * cob_intr_log (cob_field *srcfield)
- cob_field * cob_intr_log10 (cob_field *srcfield)
- cob_field * cob_intr_sin (cob_field *srcfield)
- cob_field * cob_intr_sqrt (cob_field *srcfield)
- cob_field * cob_intr_tan (cob_field *srcfield)
- cob_field * cob_intr_numval (cob_field *srcfield)
- cob_field * cob_intr_numval_c (cob_field *srcfield, cob_field *currency)
- cob_field * cob_intr_annuity (cob_field *srcfield1, cob_field *srcfield2)
- cob_field * cob_intr_sum (const int params,...)
- cob_field * cob_intr_ord_min (const int params,...)
- cob_field * cob_intr_ord_max (const int params,...)
- cob_field * cob_intr_min (const int params,...)
- cob_field * cob_intr_max (const int params,...)
- cob_field * cob_intr_midrange (const int params,...)
- cob_field * cob_intr_median (const int params,...)
- cob_field * cob_intr_mean (const int params,...)
- cob_field * cob_intr_mod (cob_field *srcfield1, cob_field *srcfield2)
- cob_field * cob_intr_range (const int params,...)
- cob_field * cob_intr_rem (cob_field *srcfield1, cob_field *srcfield2)
- cob_field * cob_intr_random (const int params,...)
- cob_field * cob_intr_variance (const int params,...)
- cob_field * cob_intr_standard_deviation (const int params,...)
- cob_field * cob_intr_present_value (const int params,...)
- cob_field * cob_intr_year_to_yyyy (const int params,...)
- cob_field * cob_intr_date_to_yyyymmdd (const int params,...)
- cob_field * cob_intr_day_to_yyyyddd (const int params,...)
- cob_field * cob_intr_seconds_past_midnight (void)
- cob_field * cob_intr_seconds_from_formatted_time (cob_field *format, cob_field *value)
- cob_field * cob_intr_locale_date (const int offset, const int length, cob_field *srcfield, cob_field *locale_field)
- cob_field * cob_intr_locale_time (const int offset, const int length, cob_field *srcfield, cob_field *locale_field)
- cob_field * cob_intr_lcl_time_from_secs (const int offset, const int length, cob_field *srcfield, cob_field *locale_field)
- void cob_init_intrinsic (void)

Variables

- static cob_decimal d2
- static cob_decimal d3
- static cob_decimal d4
- static cob_decimal d5

7.36.1 Define Documentation

7.36.1.1 #define COB_FIELD_INIT(x, y, z)

Value:

```
do { \
        field.size = x; \
        field.data = y; \
        field.attr = z; \
        } while (0)
```

Definition at line 59 of file intrinsic.c.

7.36.1.2 #define DEPTH_LEVEL 8

Definition at line 57 of file intrinsic.c.

7.36.2 Function Documentation

7.36.2.1 void cob_init_intrinsic (void)

Definition at line 3269 of file intrinsic.c.

```
{
        size_t          i;

        cob_decimal_init (&d1);
        cob_decimal_init (&d2);
        cob_decimal_init (&d3);
        cob_decimal_init (&d4);
        cob_decimal_init (&d5);
        /* mpz_init2 (mp, 256); */
        memset ((char *)&calc_field[0], 0, sizeof (calc_field));
        memset ((char *)&calc_attr[0], 0, sizeof (calc_attr));
        for (i = 0; i < DEPTH_LEVEL; ++i) {
                calc_field[i].data = cob_malloc (256);
                calc_field[i].size = 256;
                calc_size[i] = 256;
        }
        locale_buff = cob_malloc (COB_SMALL_BUFF);
}
```

7.36.2.2 cob_field* cob_intr_abs (cob_field * *srcfield*)

Definition at line 1545 of file intrinsic.c.

```
{

        make_field_entry (srcfield);

        cob_decimal_set_field (&d1, srcfield);
        mpz_abs (d1.value, d1.value);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.3 cob_field* cob_intr_acos (cob_field * *srcfield*)

Definition at line 1557 of file intrinsic.c.

```
{
        unsigned long long      result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, 0, NULL);
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = acos (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        result = (unsigned long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.36.2.4 cob_field* cob_intr_annuity (cob_field * *srcfield1*, cob_field * *srcfield2*)

Definition at line 1974 of file intrinsic.c.

```
{
        double          mathd1, mathd2;

        make_double_entry ();

        cob_decimal_set_field (&d1, srcfield1);
        cob_decimal_set_field (&d2, srcfield2);

        mathd1 = intr_get_double (&d1);
        mathd2 = intr_get_double (&d2);
        if (mathd1 == 0) {
                mathd1 = 1.0 / mathd2;
                memcpy (curr_field->data, (char *)&mathd1, sizeof (double));
                return curr_field;
        }
        mathd1 /= (1.0 - pow (mathd1 + 1.0, 0.0 - mathd2));
        memcpy (curr_field->data, (char *)&mathd1, sizeof (double));
        return curr_field;
}
```

7.36.2.5 cob_field* cob_intr_asin (cob_field * *srcfield*)

Definition at line 1591 of file intrinsic.c.

```
{
        long long               result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = asin (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        result = (long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.36.2.6 cob_field* cob_intr_atan (cob_field * *srcfield*)

Definition at line 1624 of file intrinsic.c.

```
{
        long long               result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = atan (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        result = (long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.36.2.7 cob_field* cob_intr_binop (cob_field * *f1*, int *op*, cob_field * *f2*)

Definition at line 432 of file intrinsic.c.

```
{
/* RXW
        size_t          bitnum;
        size_t          sign;
        size_t          attrsign;
        cob_field_attr  attr;
        cob_field       field;
*/

        cob_decimal_set_field (&d1, f1);
        cob_decimal_set_field (&d2, f2);
        switch (op) {
        case '+':
                cob_decimal_add (&d1, &d2);
                break;
        case '-':
                cob_decimal_sub (&d1, &d2);
                break;
```

```
        case '*':
                cob_decimal_mul (&d1, &d2);
                break;
        case '/':
                cob_decimal_div (&d1, &d2);
                break;
        case '^':
                cob_decimal_pow (&d1, &d2);
                break;
        default:
                break;
        }

/* RXW
        if (mpz_sgn (d1.value) < 0) {
                attrsign = COB_FLAG_HAVE_SIGN;
                sign = 1;
        } else {
                attrsign = 0;
                sign = 0;
        }
        bitnum = mpz_sizeinbase (d1.value, 2);
        if (bitnum < 33 - sign) {
                COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, attrsign, NULL);
                COB_FIELD_INIT (4, NULL, &attr);
                attr.scale = d1.scale;
                make_field_entry (&field);
        } else if (bitnum < 65 - sign) {
                COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, attrsign, NULL);
                COB_FIELD_INIT (8, NULL, &attr);
                attr.scale = d1.scale;
                make_field_entry (&field);
        } else {
*/
                make_double_entry ();
/* RXW
        }
*/
        cob_decimal_get_field (&d1, curr_field, 0);

        return curr_field;
}
```

7.36.2.8 cob_field* cob_intr_char (cob_field * *srcfield*)

Definition at line 1121 of file intrinsic.c.

```
{
        int             i;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (1, NULL, &attr);
        make_field_entry (&field);

        i = cob_get_int (srcfield);
        if (i < 1 || i > 256) {
                *curr_field->data = 0;
```

```
        } else {
                *curr_field->data = i - 1;
        }
        return curr_field;
}
```

7.36.2.9 cob_field* cob_intr_combined_datetime (cob_field * *srcdays,* cob_field * *srctime*)

Definition at line 1178 of file intrinsic.c.

```
{
        int             srdays;
        int             srtime;
        cob_field_attr  attr;
        cob_field       field;
        char            buff[16];

        COB_ATTR_INIT (COB_TYPE_NUMERIC_DISPLAY, 12, 5, 0, NULL);
        COB_FIELD_INIT (12, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        srdays = cob_get_int (srcdays);
        if (srdays < 1 || srdays > 3067671) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                memset (curr_field->data, '0', 12);
                return curr_field;
        }
        srtime = cob_get_int (srctime);
        if (srtime < 1 || srtime > 86400) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                memset (curr_field->data, '0', 12);
                return curr_field;
        }
        snprintf (buff, 15, "%7.7d%5.5d", srdays, srtime);
        memcpy (curr_field->data, buff, 12);
        return curr_field;
}
```

7.36.2.10 cob_field* cob_intr_concatenate (const int *offset,* const int *length,* const int *params,* ...)

Definition at line 639 of file intrinsic.c.

```
{
        cob_field       **f;
        unsigned char   *p;
        size_t          calcsize;
        int             i;
        cob_field_attr  attr;
        cob_field       field;
        va_list         args;

        f = cob_malloc (params * sizeof (cob_field *));
```

```
        va_start (args, params);

        /* Extract args / calculate size */
        calcsize = 0;
        for (i = 0; i < params; ++i) {
                f[i] = va_arg (args, cob_field *);
                calcsize += f[i]->size;
        }

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (calcsize, NULL, &attr);
        make_field_entry (&field);

        p = curr_field->data;
        for (i = 0; i < params; ++i) {
                memcpy (p, f[i]->data, f[i]->size);
                p += f[i]->size;
        }

        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        free (f);
        return curr_field;
}
```

7.36.2.11 cob_field* cob_intr_cos (cob_field * *srcfield*)

Definition at line 1657 of file intrinsic.c.

```
{
        long long               result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = cos (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        result = (long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
```

```
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.36.2.12 cob_field* cob_intr_current_date (const int *offset*, const int *length*)

Definition at line 1032 of file intrinsic.c.

```
{
#if     defined(_WIN32) && !defined(__CYGWIN__)
        long            contz;
        struct tm       *tmptr;
        struct _timeb   tmb;
        cob_field_attr  attr;
        cob_field       field;
#else
#if !defined(__linux__) && !defined(__CYGWIN__) && !defined(COB_STRFTIME) && defi
      ned(HAVE_TIMEZONE)
        struct tm       *tmptr;
        long            contz;
#endif
        time_t          curtime;
        cob_field_attr  attr;
        cob_field       field;
#if defined(HAVE_SYS_TIME_H) && defined(HAVE_GETTIMEOFDAY)
        struct timeval  tmv;
        char            buff2[8];
#endif
#endif  /* _WIN32 */
        char            buff[24];

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (21, NULL, &attr);
        make_field_entry (&field);
        memset (buff, 0, sizeof(buff));

#if     defined(_WIN32) && !defined(__CYGWIN__)
        _ftime (&tmb);
        tmptr = localtime (&(tmb.time));
        if (tmb.timezone <= 0) {
                contz = -tmb.timezone;
                snprintf (buff, 23,
                        "%4.4d%2.2d%2.2d%2.2d%2.2d%2.2d%2.2d%2.2d+%2.2ld%2.2ld",
                        tmptr->tm_year + 1900, tmptr->tm_mon + 1, tmptr->tm_mday,

                        tmptr->tm_hour, tmptr->tm_min, tmptr->tm_sec,
                        tmb.millitm / 100, contz / 60, contz % 60);
        } else {
                contz = tmb.timezone;
                snprintf (buff, 23,
                        "%4.4d%2.2d%2.2d%2.2d%2.2d%2.2d%2.2d-%2.2ld%2.2ld",
                        tmptr->tm_year + 1900, tmptr->tm_mon + 1, tmptr->tm_mday,

                        tmptr->tm_hour, tmptr->tm_min, tmptr->tm_sec,
                        tmb.millitm / 100, contz / 60, contz % 60);
        }
#else
#if defined(HAVE_SYS_TIME_H) && defined(HAVE_GETTIMEOFDAY)
        gettimeofday (&tmv, NULL);
```

```
        curtime = tmv.tv_sec;
#else
        curtime = time (NULL);
#endif

#if defined(__linux__) || defined(__CYGWIN__) || defined(COB_STRFTIME)
        strftime (buff, 22, "%Y%m%d%H%M%S00%z", localtime (&curtime));
#elif defined(HAVE_TIMEZONE)
        tmptr = localtime (&curtime);
        strftime (buff, 17, "%Y%m%d%H%M%S00", tmptr);
        /* RXW - Hack for DST - Need something better */
        if (tmptr->tm_isdst > 0) {
                timezone -= 3600;
        }
        if (timezone <= 0) {
                contz = -timezone;
                buff[16] = '+';
        } else {
                contz = timezone;
                buff[16] = '-';
        }
        sprintf(&buff[17], "%2.2ld%2.2ld", contz / 3600, (contz % 3600) / 60);
#else
        strftime (buff, 22, "%Y%m%d%H%M%S0000000", localtime (&curtime));
#endif

#if defined(HAVE_SYS_TIME_H) && defined(HAVE_GETTIMEOFDAY)
        snprintf(buff2, 7, "%2.2ld", tmv.tv_usec / 10000);
        memcpy (&buff[14], buff2, 2);
#endif
#endif  /* _WIN32 */

        memcpy (curr_field->data, buff, 21);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.36.2.13 cob_field* cob_intr_date_of_integer (cob_field * *srcdays*)

Definition at line 1209 of file intrinsic.c.

```
{
        int             i;
        int             days;
        int             baseyear = 1601;
        int             leapyear = 365;
        cob_field_attr  attr;
        cob_field       field;
        char            buff[16];

        COB_ATTR_INIT (COB_TYPE_NUMERIC_DISPLAY, 8, 0, 0, NULL);
        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        /* Base 1601-01-01 */
        days = cob_get_int (srcdays);
```

```
        if (days < 1 || days > 3067671) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                memset (curr_field->data, '0', 8);
                return curr_field;
        }
        while (days > leapyear) {
                days -= leapyear;
                ++baseyear;
                if (leap_year (baseyear)) {
                        leapyear = 366;
                } else {
                        leapyear = 365;
                }
        }
        for (i = 0; i < 13; ++i) {
                if (leap_year (baseyear)) {
                        if (days <= leap_days[i]) {
                                days -= leap_days[i-1];
                                break;
                        }
                } else {
                        if (days <= normal_days[i]) {
                                days -= normal_days[i-1];
                                break;
                        }
                }
        }
        snprintf (buff, 15, "%4.4d%2.2d%2.2d", baseyear, i, days);
        memcpy (curr_field->data, buff, 8);
        return curr_field;
}
```

7.36.2.14 cob_field* cob_intr_date_to_yyyymmdd (const int *params*, ...)

Definition at line 2646 of file intrinsic.c.

```
{
        cob_field       *f;
        struct tm       *timeptr;
        va_list         args;
        time_t          t;
        int             year;
        int             mmdd;
        int             interval;
        int             xqtyear;
        int             maxyear;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        va_start (args, params);
        f = va_arg (args, cob_field *);
        year = cob_get_int (f);
        mmdd = year % 10000;
        year /= 10000;
```

```
        if (params > 1) {
                f = va_arg (args, cob_field *);
                interval = cob_get_int (f);
        } else {
                interval = 50;
        }
        if (params > 2) {
                f = va_arg (args, cob_field *);
                xqtyear = cob_get_int (f);
        } else {
                t = time (NULL);
                timeptr = localtime (&t);
                xqtyear = 1900 + timeptr->tm_year;
        }
        va_end (args);

        if (year < 0 || year > 999999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (xqtyear < 1601 || xqtyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        maxyear = xqtyear + interval;
        if (maxyear < 1700 || maxyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (maxyear % 100 >= year) {
                year += 100 * (maxyear / 100);
        } else {
                year += 100 * ((maxyear / 100) - 1);
        }
        year *= 10000;
        year += mmdd;
        cob_set_int (curr_field, year);
        return curr_field;
}
```

7.36.2.15 cob_field* cob_intr_day_of_integer (cob_field * *srcdays*)

Definition at line 1259 of file intrinsic.c.

```
{
        int             days;
        int             baseyear = 1601;
        int             leapyear = 365;
        cob_field_attr  attr;
        cob_field       field;
        char            buff[16];

        COB_ATTR_INIT (COB_TYPE_NUMERIC_DISPLAY, 7, 0, 0, NULL);
        COB_FIELD_INIT (7, NULL, &attr);
        make_field_entry (&field);
```

```
        cob_exception_code = 0;
        /* Base 1601-01-01 */
        days = cob_get_int (srcdays);
        if (days < 1 || days > 3067671) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                memset (curr_field->data, '0', 7);
                return curr_field;
        }
        while (days > leapyear) {
                days -= leapyear;
                ++baseyear;
                if (leap_year (baseyear)) {
                        leapyear = 366;
                } else {
                        leapyear = 365;
                }
        }
        snprintf (buff, 15, "%4.4d%3.3d", baseyear, days);
        memcpy (curr_field->data, buff, 7);
        return curr_field;
}
```

7.36.2.16 cob_field* cob_intr_day_to_yyyyddd (const int *params*, ...)

Definition at line 2714 of file intrinsic.c.

```
{
        cob_field       *f;
        struct tm       *timeptr;
        va_list         args;
        time_t          t;
        int             year;
        int             days;
        int             interval;
        int             xqtyear;
        int             maxyear;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        va_start (args, params);
        f = va_arg (args, cob_field *);
        year = cob_get_int (f);
        days = year % 1000;
        year /= 1000;
        if (params > 1) {
                f = va_arg (args, cob_field *);
                interval = cob_get_int (f);
        } else {
                interval = 50;
        }
        if (params > 2) {
                f = va_arg (args, cob_field *);
                xqtyear = cob_get_int (f);
        } else {
```

```
                t = time (NULL);
                timeptr = localtime (&t);
                xqtyear = 1900 + timeptr->tm_year;
        }
        va_end (args);

        if (year < 0 || year > 999999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (xqtyear < 1601 || xqtyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        maxyear = xqtyear + interval;
        if (maxyear < 1700 || maxyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (maxyear % 100 >= year) {
                year += 100 * (maxyear / 100);
        } else {
                year += 100 * ((maxyear / 100) - 1);
        }
        year *= 1000;
        year += days;
        cob_set_int (curr_field, year);
        return curr_field;
}
```

7.36.2.17 cob_field* cob_intr_exception_file (void)

Definition at line 916 of file intrinsic.c.

```
{
        size_t          flen;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        if (cob_exception_code == 0 || !cob_error_file ||
            (cob_exception_code & 0x0500) != 0x0500) {
                field.size = 2;
                make_field_entry (&field);
                memcpy (curr_field->data, "00", 2);
        } else {
                flen = strlen (cob_error_file->select_name);
                field.size = flen + 2;
                make_field_entry (&field);
                memcpy (curr_field->data, cob_error_file->file_status, 2);
                memcpy (&(curr_field->data[2]), cob_error_file->select_name, flen
      );
        }
        return curr_field;
}
```

7.36.2.18 cob_field* cob_intr_exception_location (void)

Definition at line 940 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        if (!cob_got_exception || !cob_orig_program_id) {
                field.size = 1;
                make_field_entry (&field);
                *(curr_field->data) = ' ';
                return curr_field;
        }
        memset (locale_buff, 0, COB_SMALL_BUFF);
        if (cob_orig_section && cob_orig_paragraph) {
                snprintf (locale_buff, COB_SMALL_MAX, "%s; %s OF %s; %d",
                          cob_orig_program_id, cob_orig_paragraph,
                          cob_orig_section, cob_orig_line);
        } else if (cob_orig_section) {
                snprintf (locale_buff, COB_SMALL_MAX, "%s; %s; %d",
                          cob_orig_program_id, cob_orig_section, cob_orig_line);
        } else if (cob_orig_paragraph) {
                snprintf (locale_buff, COB_SMALL_MAX, "%s; %s; %d",
                          cob_orig_program_id, cob_orig_paragraph, cob_orig_line)
      ;
        } else {
                snprintf (locale_buff, COB_SMALL_MAX, "%s; ; %d",
                          cob_orig_program_id, cob_orig_line);
        }
        field.size = strlen (locale_buff);
        make_field_entry (&field);
        memcpy (curr_field->data, locale_buff, field.size);
        return curr_field;
}
```

7.36.2.19 cob_field* cob_intr_exception_statement (void)

Definition at line 997 of file intrinsic.c.

```
{
        size_t          flen;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (31, NULL, &attr);
        make_field_entry (&field);

        memset (curr_field->data, ' ', 31);
        if (cob_exception_code && cob_orig_statement) {
                flen = strlen (cob_orig_statement);
                if (flen > 31) {
                        memcpy (curr_field->data, cob_orig_statement, 31);
                } else {
                        memcpy (curr_field->data, cob_orig_statement, flen);
```

```
                }
        }
        return curr_field;
}
```

7.36.2.20 cob_field* cob_intr_exception_status (void)

Definition at line 975 of file intrinsic.c.

```
{
        const char      *except_name;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (31, NULL, &attr);
        make_field_entry (&field);

        memset (curr_field->data, ' ', 31);
        if (cob_exception_code) {
                except_name = cob_get_exception_name (cob_exception_code);
                if (except_name == NULL) {
                        except_name = "EXCEPTION-OBJECT";
                }
                memcpy (curr_field->data, except_name, strlen (except_name));
        }
        return curr_field;
}
```

7.36.2.21 cob_field* cob_intr_exp (cob_field * *srcfield*)

Definition at line 1509 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = pow (2.7182818284590452354, intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.36.2.22 cob_field* cob_intr_exp10 (cob_field * *srcfield*)

Definition at line 1527 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = pow (10.0, intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.36.2.23 cob_field* cob_intr_factorial (cob_field * *srcfield*)

Definition at line 1485 of file intrinsic.c.

```
{
        int             srcval;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, 0, NULL);
        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        srcval = cob_get_int (srcfield);
        if (srcval < 0) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        d1.scale = 0;
        mpz_fac_ui (d1.value, (unsigned int)srcval);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.24 cob_field* cob_intr_fraction_part (cob_field * *srcfield*)

Definition at line 552 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 18, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);
```

```
        cob_move (srcfield, curr_field);
        return curr_field;
}
```

7.36.2.25 cob_field* cob_intr_integer (cob_field * *srcfield*)

Definition at line 511 of file intrinsic.c.

```
{
        int             i, scale;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_decimal_set_field (&d1, srcfield);
        if (mpz_sgn (d1.value) >= 0) {
                cob_decimal_get_field (&d1, curr_field, 0);
                return curr_field;
        }
        scale = 1;
        for (i = 0; i < d1.scale; ++i) {
                scale *= 10;
        }
        if (mpz_fdiv_ui (d1.value, (unsigned int)scale)) {
                mpz_sub_ui (d1.value, d1.value, (unsigned int)scale);
        }
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.26 cob_field* cob_intr_integer_of_date (cob_field * *srcfield*)

Definition at line 1295 of file intrinsic.c.

```
{
        int             indate;
        int             days;
        int             totaldays;
        int             month;
        int             year;
        int             baseyear = 1601;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        /* Base 1601-01-01 */
        indate = cob_get_int (srcfield);
```

```
        year = indate / 10000;
        if (year < 1601 || year > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        indate %= 10000;
        month = indate / 100;
        if (month < 1 || month > 12) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        days = indate % 100;
        if (days < 1 || days > 31) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (leap_year (year)) {
                if (days > leap_month_days[month]) {
                        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                        cob_set_int (curr_field, 0);
                        return curr_field;
                }
        } else {
                if (days > normal_month_days[month]) {
                        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                        cob_set_int (curr_field, 0);
                        return curr_field;
                }
        }
        totaldays = 0;
        while (baseyear != year) {
                if (leap_year (baseyear)) {
                        totaldays += 366;
                } else {
                        totaldays += 365;
                }
                ++baseyear;
        }
        if (leap_year (baseyear)) {
                totaldays += leap_days[month - 1];
        } else {
                totaldays += normal_days[month - 1];
        }
        totaldays += days;
        cob_set_int (curr_field, totaldays);
        return curr_field;
}
```

7.36.2.27 cob_field* cob_intr_integer_of_day (cob_field * *srcfield*)

Definition at line 1365 of file intrinsic.c.

```
{
        int             indate;
        int             days;
        int             totaldays;
```

```
        int             year;
        int             baseyear = 1601;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        /* Base 1601-01-01 */
        indate = cob_get_int (srcfield);
        year = indate / 1000;
        if (year < 1601 || year > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        days = indate % 1000;
        if (days < 1 || days > 365 + leap_year (year)) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        totaldays = 0;
        while (baseyear != year) {
                if (leap_year (baseyear)) {
                        totaldays += 366;
                } else {
                        totaldays += 365;
                }
                ++baseyear;
        }
        totaldays += days;
        cob_set_int (curr_field, totaldays);
        return curr_field;
}
```

7.36.2.28 cob_field* cob_intr_integer_part (cob_field * *srcfield*)

Definition at line 538 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_move (srcfield, curr_field);
        return curr_field;
}
```

7.36.2.29 cob_field* cob_intr_lcl_time_from_secs (const int *offset*, const int *length*, cob_field * *srcfield*, cob_field * *locale_field*)

Definition at line 3149 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;
#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        size_t          len;
        int             indate;
        int             hours;
        int             minutes;
        int             seconds;
#ifdef  HAVE_LANGINFO_CODESET
        char            *deflocale = NULL;
        char            *localep = NULL;
        char            *localep2;
        struct tm       tstruct;
        char            buff2[128];
#else
        char            *p;
        LCID            localeid = LOCALE_USER_DEFAULT;
        SYSTEMTIME      syst;
#endif
        char            buff[128];
#endif

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        cob_exception_code = 0;

#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        if (COB_FIELD_IS_NUMERIC (srcfield)) {
                indate = cob_get_int (srcfield);
        } else {
                goto derror;
        }
        if (indate > 86400) {
                goto derror;
        }
        hours = indate / 3600;
        indate %= 3600;
        minutes = indate / 60;
        seconds = indate % 60;

#ifdef  HAVE_LANGINFO_CODESET
        memset ((void *)&tstruct, 0, sizeof(struct tm));
        tstruct.tm_hour = hours;
        tstruct.tm_min = minutes;
        tstruct.tm_sec = seconds;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                deflocale = locale_buff;
                localep2 = setlocale (LC_TIME, NULL);
                if (localep2) {
                        localep = strdup (localep2);
                }
```

```
                (void) setlocale (LC_TIME, deflocale);
        }
        memset (buff2, 0, sizeof(buff2));
        snprintf(buff2, sizeof(buff2) - 1, "%s", nl_langinfo(T_FMT));
        if (deflocale) {
                if (localep) {
                        (void) setlocale (LC_TIME, localep);
                }
        }
        strftime (buff, sizeof(buff), buff2, &tstruct);
#else
        memset ((void *)&syst, 0, sizeof(syst));
        syst.wHour = hours;
        syst.wMinute = minutes;
        syst.wSecond = seconds;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                for (p = locale_buff; *p; ++p) {
                        if (isalnum(*p) || *p == '_') {
                                continue;
                        }
                        break;
                }
                *p = 0;
                for (len = 0; len < WINLOCSIZE; ++len) {
                        if (!strcmp(locale_buff, wintable[len].winlocalename)) {
                                localeid = wintable[len].winlocaleid;
                                break;
                        }
                }
                if (len == WINLOCSIZE) {
                        goto derror;
                }
        }
        if (!GetTimeFormat (localeid, LOCALE_NOUSEROVERRIDE, &syst, NULL, buff, s
      izeof(buff))) {

                goto derror;
        }
#endif
        len = strlen (buff);
        field.size = len;
        make_field_entry (&field);
        memcpy (curr_field->data, buff, len);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
derror:
#endif
        field.size = 10;
        make_field_entry (&field);
        memset (curr_field->data, ' ', 10);
        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
        return curr_field;
}
```

7.36.2.30 cob_field* cob_intr_length (cob_field * *srcfield*)

Definition at line 497 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_set_int (curr_field, (int)srcfield->size);
        return curr_field;
}
```

7.36.2.31 cob_field* cob_intr_locale_date (const int *offset,* const int *length,* cob_field * *srcfield,* cob_field * *locale_field*)

Definition at line 2866 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;
#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        size_t          len;
        int             indate;
        int             days;
        int             month;
        int             year;
#ifdef  HAVE_LANGINFO_CODESET
        unsigned char   *p;
        char            *deflocale = NULL;
        char            *localep = NULL;
        char            *localep2;
        struct tm       tstruct;
        char            buff2[128];
#else
        char            *p;
        LCID            localeid = LOCALE_USER_DEFAULT;
        SYSTEMTIME      syst;
#endif
        char            buff[128];
#endif

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        cob_exception_code = 0;

#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        if (COB_FIELD_IS_NUMERIC (srcfield)) {
                indate = cob_get_int (srcfield);
        } else {
                if (srcfield->size < 8) {
                        goto derror;
                }
                p = srcfield->data;
                indate = 0;
```

```
                for (len = 0; len < 8; ++len, ++p) {
                        if (isdigit (*p)) {
                                indate *= 10;
                                indate += (*p - '0');
                        } else {
                                goto derror;
                        }
                }
        }
        year = indate / 10000;
        if (year < 1601 || year > 9999) {
                goto derror;
        }
        indate %= 10000;
        month = indate / 100;
        if (month < 1 || month > 12) {
                goto derror;
        }
        days = indate % 100;
        if (days < 1 || days > 31) {
                goto derror;
        }
        if (leap_year (year)) {
                if (days > leap_month_days[month]) {
                        goto derror;
                }
        } else {
                if (days > normal_month_days[month]) {
                        goto derror;
                }
        }
#ifdef  HAVE_LANGINFO_CODESET
        month--;

        memset ((void *)&tstruct, 0, sizeof(struct tm));
        tstruct.tm_year = year - 1900;
        tstruct.tm_mon = month;
        tstruct.tm_mday = days;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                deflocale = locale_buff;
                localep2 = setlocale (LC_TIME, NULL);
                if (localep2) {
                        localep = strdup (localep2);
                }
                (void) setlocale (LC_TIME, deflocale);
        }
        memset (buff2, 0, sizeof(buff2));
        snprintf(buff2, sizeof(buff2) - 1, "%s", nl_langinfo(D_FMT));
        if (deflocale) {
                if (localep) {
                        (void) setlocale (LC_TIME, localep);
                }
        }
        strftime (buff, sizeof(buff), buff2, &tstruct);
#else
        memset ((void *)&syst, 0, sizeof(syst));
        syst.wYear = year;
        syst.wMonth = month;
```

```
        syst.wDay = days;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                for (p = locale_buff; *p; ++p) {
                        if (isalnum(*p) || *p == '_') {
                                continue;
                        }
                        break;
                }
                *p = 0;
                for (len = 0; len < WINLOCSIZE; ++len) {
                        if (!strcmp(locale_buff, wintable[len].winlocalename)) {
                                localeid = wintable[len].winlocaleid;
                                break;
                        }
                }
                if (len == WINLOCSIZE) {
                        goto derror;
                }
        }
        if (!GetDateFormat (localeid, DATE_SHORTDATE, &syst, NULL, buff, sizeof(b
      uff))) {
                goto derror;
        }
#endif
        len = strlen (buff);
        field.size = len;
        make_field_entry (&field);
        memcpy (curr_field->data, buff, len);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
derror:
#endif
        field.size = 10;
        make_field_entry (&field);
        memset (curr_field->data, ' ', 10);
        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
        return curr_field;
}
```

7.36.2.32 cob_field* cob_intr_locale_time (const int *offset*, const int *length*, cob_field * *srcfield*, cob_field * *locale_field*)

Definition at line 3012 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;
#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        size_t          len;
        int             indate;
        int             hours;
        int             minutes;
        int             seconds;
```

```
#ifdef  HAVE_LANGINFO_CODESET
        unsigned char   *p;
        char            *deflocale = NULL;
        char            *localep = NULL;
        char            *localep2;
        struct tm       tstruct;
        char            buff2[128];
#else
        char            *p;
        LCID            localeid = LOCALE_USER_DEFAULT;
        SYSTEMTIME      syst;
#endif
        char            buff[128];
#endif

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        cob_exception_code = 0;

#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        if (COB_FIELD_IS_NUMERIC (srcfield)) {
                indate = cob_get_int (srcfield);
        } else {
                if (srcfield->size < 6) {
                        goto derror;
                }
                p = srcfield->data;
                indate = 0;
                for (len = 0; len < 6; ++len, ++p) {
                        if (isdigit (*p)) {
                                indate *= 10;
                                indate += (*p - '0');
                        } else {
                                goto derror;
                        }
                }
        }
        hours = indate / 10000;
        if (hours < 0 || hours > 24) {
                goto derror;
        }
        indate %= 10000;
        minutes = indate / 100;
        if (minutes < 0 || minutes > 59) {
                goto derror;
        }
        seconds = indate % 100;
        if (seconds < 0 || seconds > 59) {
                goto derror;
        }

#ifdef  HAVE_LANGINFO_CODESET
        memset ((void *)&tstruct, 0, sizeof(struct tm));
        tstruct.tm_hour = hours;
        tstruct.tm_min = minutes;
        tstruct.tm_sec = seconds;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                deflocale = locale_buff;
```

```
                localep2 = setlocale (LC_TIME, NULL);
                if (localep2) {
                        localep = strdup (localep2);
                }
                (void) setlocale (LC_TIME, deflocale);
        }
        memset (buff2, 0, sizeof(buff2));
        snprintf(buff2, sizeof(buff2) - 1, "%s", nl_langinfo(T_FMT));
        if (deflocale) {
                if (localep) {
                        (void) setlocale (LC_TIME, localep);
                }
        }
        strftime (buff, sizeof(buff), buff2, &tstruct);
#else
        memset ((void *)&syst, 0, sizeof(syst));
        syst.wHour = hours;
        syst.wMinute = minutes;
        syst.wSecond = seconds;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                for (p = locale_buff; *p; ++p) {
                        if (isalnum(*p) || *p == '_') {
                                continue;
                        }
                        break;
                }
                *p = 0;
                for (len = 0; len < WINLOCSIZE; ++len) {
                        if (!strcmp(locale_buff, wintable[len].winlocalename)) {
                                localeid = wintable[len].winlocaleid;
                                break;
                        }
                }
                if (len == WINLOCSIZE) {
                        goto derror;
                }
        }
        if (!GetTimeFormat (localeid, LOCALE_NOUSEROVERRIDE, &syst, NULL, buff, s
      izeof(buff))) {

                goto derror;
        }
#endif
        len = strlen (buff);
        field.size = len;
        make_field_entry (&field);
        memcpy (curr_field->data, buff, len);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
derror:
#endif
        field.size = 10;
        make_field_entry (&field);
        memset (curr_field->data, ' ', 10);
        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
        return curr_field;
```

```
}
```

7.36.2.33 cob_field* cob_intr_log (cob_field * *srcfield*)

Definition at line 1690 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = log (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.36.2.34 cob_field* cob_intr_log10 (cob_field * *srcfield*)

Definition at line 1708 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = log10 (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.36.2.35 cob_field* cob_intr_lower_case (const int *offset,* const int *length,* cob_field * *srcfield*)

Definition at line 605 of file intrinsic.c.

```
{
        size_t          i, size;

        make_field_entry (srcfield);
```

```
        size = srcfield->size;
        for (i = 0; i < size; ++i) {
                curr_field->data[i] = tolower (srcfield->data[i]);
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.36.2.36 cob_field* cob_intr_max (const int *params*, ...)

Definition at line 2125 of file intrinsic.c.

```
{
        cob_field       *f, *basef;
        va_list         args;
        int             i;

        va_start (args, params);

        basef = va_arg (args, cob_field *);
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basef) > 0) {
                        basef = f;
                }
        }
        va_end (args);

        return basef;
}
```

7.36.2.37 cob_field* cob_intr_mean (const int *params*, ...)

Definition at line 2224 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        long long       n;
        union {
                unsigned char   data[8];
                long long       datall;
        } datun;
        int             i;
        cob_field_attr  attr;
        cob_field       field;


        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        mpz_set_ui (d1.value, 0);
```

```
        d1.scale = 0;

        va_start (args, params);
        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                cob_decimal_add (&d1, &d2);
        }
        va_end (args);

        mpz_set_ui (d2.value, (unsigned int)params);
        d2.scale = 0;
        cob_decimal_div (&d1, &d2);
        field.data = datun.data;
        cob_decimal_get_field (&d1, &field, 0);
        n = datun.datall;
        for (i = 0; n; n /= 10, ++i) ;
        field.data = NULL;
        if (i <= 18) {
                attr.scale = 18 - i;
        }
        make_field_entry (&field);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.38 cob_field* cob_intr_median (const int *params*, ...)

Definition at line 2179 of file intrinsic.c.

```
{
        cob_field       *f;
        cob_field       **field_alloc;
        va_list         args;
        int             i;

        va_start (args, params);

        f = va_arg (args, cob_field *);
        if (params == 1) {
                va_end (args);
                return f;
        }

        field_alloc = cob_malloc (params * sizeof (cob_field *));
        field_alloc[0] = f;

        for (i = 1; i < params; ++i) {
                field_alloc[i] = va_arg (args, cob_field *);
        }
        va_end (args);

        qsort (field_alloc, (size_t)params, (size_t)sizeof (cob_field *), comp_fi
      eld);

        i = params / 2;
        if (params % 2) {
                f = field_alloc[i];
        } else {
```

```
                make_double_entry ();
                cob_decimal_set_field (&d1, field_alloc[i-1]);
                cob_decimal_set_field (&d2, field_alloc[i]);
                cob_decimal_add (&d1, &d2);
                mpz_set_ui (d2.value, 2);
                d2.scale = 0;
                cob_decimal_div (&d1, &d2);
                cob_decimal_get_field (&d1, curr_field, 0);
                f = curr_field;
        }

        free (field_alloc);
        return f;
}
```

7.36.2.39 cob_field* cob_intr_midrange (const int *params*, ...)

Definition at line 2146 of file intrinsic.c.

```
{
        cob_field       *f, *basemin, *basemax;
        va_list         args;
        int             i;

        make_double_entry ();
        va_start (args, params);

        basemin = va_arg (args, cob_field *);
        basemax = basemin;
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basemin) < 0) {
                        basemin = f;
                }
                if (cob_cmp (f, basemax) > 0) {
                        basemax = f;
                }
        }
        va_end (args);

        cob_decimal_set_field (&d1, basemin);
        cob_decimal_set_field (&d2, basemax);
        cob_decimal_add (&d1, &d2);
        mpz_set_ui (d2.value, 2);
        d2.scale = 0;
        cob_decimal_div (&d1, &d2);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.40 cob_field* cob_intr_min (const int *params*, ...)

Definition at line 2104 of file intrinsic.c.

```
{
```

```
        cob_field       *f, *basef;
        va_list         args;
        int             i;

        va_start (args, params);

        basef = va_arg (args, cob_field *);
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basef) < 0) {
                        basef = f;
                }
        }
        va_end (args);

        return basef;
}
```

7.36.2.41 cob_field* cob_intr_mod (cob_field * *srcfield1,* cob_field * *srcfield2*)

Definition at line 2268 of file intrinsic.c.

```
{
        cob_field       *f1;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        f1 = cob_intr_integer (cob_intr_binop (srcfield1, '/', srcfield2));
        cob_decimal_set_field (&d1, srcfield2);
        cob_decimal_set_field (&d2, f1);
        cob_decimal_mul (&d2, &d1);
        cob_decimal_set_field (&d1, srcfield1);
        cob_decimal_sub (&d1, &d2);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.42 cob_field* cob_intr_numval (cob_field * *srcfield*)

Definition at line 1795 of file intrinsic.c.

```
{
        long long       llval = 0;
        double          val;
        size_t          i;
        int             integer_digits = 0;
        int             decimal_digits = 0;
        int             sign = 0;
        int             decimal_seen = 0;
        cob_field_attr  attr;
```

```
  cob_field       field;
  unsigned char   integer_buff[64];
  unsigned char   decimal_buff[64];
  unsigned char   final_buff[64];

  COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

  COB_FIELD_INIT (8, NULL, &attr);
  memset (integer_buff, 0, sizeof (integer_buff));
  memset (decimal_buff, 0, sizeof (decimal_buff));
  memset (final_buff, 0, sizeof (final_buff));

  for (i = 0; i < srcfield->size; ++i) {
          if (i < (srcfield->size - 2)) {
                  if (strcasecmp ((char *)&srcfield->data[i], "CR") == 0
                       || strcasecmp ((char *)&srcfield->data[i], "DB") ==
0) {
                          sign = 1;
                          break;
                  }
          }
          if (srcfield->data[i] == ' ') {
                  continue;
          }
          if (srcfield->data[i] == '+') {
                  continue;
          }
          if (srcfield->data[i] == '-') {
                  sign = 1;
                  continue;
          }
          if (srcfield->data[i] == cob_current_module->decimal_point) {
                  decimal_seen = 1;
                  continue;
          }
          if (srcfield->data[i] >= '0' && srcfield->data[i] <= '9') {
                  llval *= 10;
                  llval += srcfield->data[i] - '0';
                  if (decimal_seen) {
                          decimal_buff[decimal_digits++] = srcfield->data[i
];
                  } else {
                          integer_buff[integer_digits++] = srcfield->data[i
];
                  }
          }
          if ((integer_digits + decimal_digits) > 30) {
                  break;
          }
  }
  if (!integer_digits) {
          integer_buff[0] = '0';
  }
  if (!decimal_digits) {
          decimal_buff[0] = '0';
  }
  if (sign) {
          llval = -llval;
  }
  if ((integer_digits + decimal_digits) <= 18) {
          attr.scale = decimal_digits;
          make_field_entry (&field);
```

```
                memcpy (curr_field->data, (char *)&llval, 8);
        } else {
                snprintf ((char *)final_buff, 63, "%s%s.%s", sign ? "-" : "",
                          integer_buff, decimal_buff);
                sscanf ((char *)final_buff, "%lf", &val);
                make_double_entry ();
                memcpy (curr_field->data, (char *)&val, sizeof (double));
        }
        return curr_field;
}
```

7.36.2.43 cob_field* cob_intr_numval_c (cob_field * *srcfield,* cob_field * *currency*)

Definition at line 1875 of file intrinsic.c.

```
{
        unsigned char   *currency_data;
        long long       llval = 0;
        double          val;
        size_t          i;
        int             integer_digits = 0;
        int             decimal_digits = 0;
        int             sign = 0;
        int             decimal_seen = 0;
        cob_field_attr  attr;
        cob_field       field;
        unsigned char   integer_buff[64];
        unsigned char   decimal_buff[64];
        unsigned char   final_buff[64];

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        memset (integer_buff, 0, sizeof (integer_buff));
        memset (decimal_buff, 0, sizeof (decimal_buff));
        memset (final_buff, 0, sizeof (final_buff));

        currency_data = NULL;
        if (currency) {
                if (currency->size < srcfield->size) {
                        currency_data = currency->data;
                }
        }
        for (i = 0; i < srcfield->size; ++i) {
                if (i < (srcfield->size - 2)) {
                        if (strcasecmp ((char *)&srcfield->data[i], "CR") == 0
                            || strcasecmp ((char *)&srcfield->data[i], "DB") ==
      0) {
                                sign = 1;
                                break;
                        }
                }
                if (currency_data) {
                        if (i < (srcfield->size - currency->size)) {
                                if (memcmp ((char *)&srcfield->data[i], currency_
      data,
                                            currency->size) == 0) {
                                        i += (currency->size - 1);
                                        continue;
```

```
                                }
                        }
                }
                if (srcfield->data[i] == ' ') {
                        continue;
                }
                if (srcfield->data[i] == '+') {
                        continue;
                }
                if (srcfield->data[i] == '-') {
                        sign = 1;
                        continue;
                }
                if (srcfield->data[i] == cob_current_module->decimal_point) {
                        decimal_seen = 1;
                        continue;
                }
                if (srcfield->data[i] == cob_current_module->currency_symbol) {
                        continue;
                }
                if (srcfield->data[i] >= '0' && srcfield->data[i] <= '9') {
                        llval *= 10;
                        llval += srcfield->data[i] - '0';
                        if (decimal_seen) {
                                decimal_buff[decimal_digits++] = srcfield->data[i
      ];
                        } else {
                                integer_buff[integer_digits++] = srcfield->data[i
      ];
                        }
                }
                if ((integer_digits + decimal_digits) > 30) {
                        break;
                }
        }
        if (!integer_digits) {
                integer_buff[0] = '0';
        }
        if (!decimal_digits) {
                decimal_buff[0] = '0';
        }
        if (sign) {
                llval = -llval;
        }
        if ((integer_digits + decimal_digits) <= 18) {
                attr.scale = decimal_digits;
                make_field_entry (&field);
                memcpy (curr_field->data, (char *)&llval, 8);
        } else {
                snprintf ((char *)final_buff, 63, "%s%s.%s", sign ? "-" : "",
                         integer_buff, decimal_buff);
                sscanf ((char *)final_buff, "%lf", &val);
                make_double_entry ();
                memcpy (curr_field->data, (char *)&val, sizeof (double));
        }
        return curr_field;
}
```

7.36.2.44 cob_field* cob_intr_ord (cob_field * *srcfield*)

Definition at line 1141 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_set_int (curr_field, (int)(*srcfield->data + 1));
        return curr_field;
}
```

7.36.2.45 cob_field* cob_intr_ord_max (const int *params*, ...)

Definition at line 2069 of file intrinsic.c.

```
{
        cob_field       *f, *basef;
        int             ordmin = 0;
        int             i;
        va_list         args;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        if (params <= 1) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        va_start (args, params);

        basef = va_arg (args, cob_field *);
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basef) > 0) {
                        basef = f;
                        ordmin = i;
                }
        }
        va_end (args);

        cob_set_int (curr_field, ordmin + 1);
        return curr_field;
}
```

7.36.2.46 cob_field* cob_intr_ord_min (const int *params*, ...)

Definition at line 2034 of file intrinsic.c.

```
{
        cob_field       *f, *basef;
        int             i;
        int             ordmin = 0;
        va_list         args;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        if (params <= 1) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        va_start (args, params);

        basef = va_arg (args, cob_field *);
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basef) < 0) {
                        basef = f;
                        ordmin = i;
                }
        }
        va_end (args);

        cob_set_int (curr_field, ordmin + 1);
        return curr_field;
}
```

7.36.2.47 cob_field* cob_intr_present_value (const int *params*, ...)

Definition at line 2538 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        int             i;

        va_start (args, params);
        make_double_entry ();

        if (params < 2) {
                va_end (args);
                fprintf (stderr, "Wrong number of parameters for FUNCTION PRESENT
      -VALUE\n");
                fflush (stderr);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        f = va_arg (args, cob_field *);
        cob_decimal_set_field (&d1, f);
        mpz_set_ui (d2.value, 1);
        d2.scale = 0;
        cob_decimal_add (&d1, &d2);
```

```
        mpz_set_ui (d4.value, 0);
        d4.scale = 0;

        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                mpz_set (d3.value, d1.value);
                d3.scale = d1.scale;
                if (i > 1) {
                        mpz_set_ui (d5.value, (unsigned int)i);
                        d5.scale = 0;
                        cob_decimal_pow (&d3, &d5);
                }
                cob_decimal_div (&d2, &d3);
                cob_decimal_add (&d4, &d2);
        }
        va_end (args);

        cob_decimal_get_field (&d4, curr_field, 0);
        return curr_field;
}
```

7.36.2.48 cob_field* cob_intr_random (const int *params*, ...)

Definition at line 2352 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        int             seed = 1;
        int             randnum;
        int             i;
        int             exp10;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 9, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        va_start (args, params);

        if (params) {
                f = va_arg (args, cob_field *);
                seed = cob_get_int (f);
                if (seed < 0) {
                        seed = 0;
                }
#ifdef  __CYGWIN__
                srandom ((unsigned int)seed);
#else
                srand ((unsigned int)seed);
#endif
        }
        va_end (args);

#ifdef  __CYGWIN__
        randnum = (int)random ();
#else
        randnum = rand ();
```

```
#endif
        exp10 = 1;
        for (i = 0; i < 10; ++i) {
                if ((randnum / exp10) == 0) {
                        break;
                }
                exp10 *= 10;
        }
        if (i == 0) {
                i = 1;
        }
        attr.scale = i;
        make_field_entry (&field);
        *(long long *)curr_field->data = (long long)randnum;
        return curr_field;
}
```

7.36.2.49 cob_field* cob_intr_range (const int *params*, ...)

Definition at line 2289 of file intrinsic.c.

```
{
        cob_field       *f, *basemin, *basemax;
        va_list         args;
        int             i;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        va_start (args, params);

        basemin = va_arg (args, cob_field *);
        basemax = basemin;
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basemin) < 0) {
                        basemin = f;
                }
                if (cob_cmp (f, basemax) > 0) {
                        basemax = f;
                }
        }
        va_end (args);

        attr.scale = COB_FIELD_SCALE(basemin);
        if (COB_FIELD_SCALE(basemax) > attr.scale) {
                attr.scale = COB_FIELD_SCALE(basemax);
        }
        make_field_entry (&field);
        cob_decimal_set_field (&d1, basemax);
        cob_decimal_set_field (&d2, basemin);
        cob_decimal_sub (&d1, &d2);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.50 cob_field* cob_intr_rem (cob_field * *srcfield1*, cob_field * *srcfield2*)

Definition at line 2327 of file intrinsic.c.

```
{
        cob_field       *f1;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        f1 = cob_intr_integer_part (cob_intr_binop (srcfield1, '/', srcfield2));
        cob_decimal_set_field (&d1, srcfield2);
        cob_decimal_set_field (&d2, f1);
        cob_decimal_mul (&d2, &d1);
        cob_decimal_set_field (&d1, srcfield1);
        cob_decimal_sub (&d1, &d2);

        attr.scale = COB_FIELD_SCALE(srcfield1);
        if (COB_FIELD_SCALE(srcfield2) > attr.scale) {
                attr.scale = COB_FIELD_SCALE(srcfield2);
        }
        make_field_entry (&field);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.51 cob_field* cob_intr_reverse (const int *offset*, const int *length*, cob_field * *srcfield*)

Definition at line 622 of file intrinsic.c.

```
{
        size_t          i, size;

        make_field_entry (srcfield);

        size = srcfield->size;
        for (i = 0; i < size; ++i) {
                curr_field->data[i] = srcfield->data[srcfield->size - i - 1];
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.36.2.52 cob_field* cob_intr_seconds_from_formatted_time (cob_field * *format*, cob_field * *value*)

Definition at line 2803 of file intrinsic.c.

```
{
```

```
        unsigned char   *p1;
        unsigned char   *p2;
        size_t          n;
        int             seconds = 0;
        int             minutes = 0;
        int             hours = 0;
        int             seconds_seen = 0;
        int             minutes_seen = 0;
        int             hours_seen = 0;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        if (value->size < format->size) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        p1 = format->data;
        p2 = value->data;
        for (n = 0; n < format->size - 1; ++n, ++p1, ++p2) {
                if (!memcmp (p1, "hh", 2) && !hours_seen) {
                        if (*p2 >= '0' && *p2 <= '9' &&
                            *(p2 + 1) >= '0' && *(p2 + 1) <= '9') {
                                hours = ((*p2 - '0') * 10) + (*(p2 + 1) - '0');
                                hours_seen = 1;
                                continue;
                        }
                }
                if (!memcmp (p1, "mm", 2) && !minutes_seen) {
                        if (*p2 >= '0' && *p2 <= '9' &&
                            *(p2 + 1) >= '0' && *(p2 + 1) <= '9') {
                                minutes = ((*p2 - '0') * 10) + (*(p2 + 1) - '0');

                                minutes_seen = 1;
                                continue;
                        }
                }
                if (!memcmp (p1, "ss", 2) && !seconds_seen) {
                        if (*p2 >= '0' && *p2 <= '9' &&
                            *(p2 + 1) >= '0' && *(p2 + 1) <= '9') {
                                seconds = ((*p2 - '0') * 10) + (*(p2 + 1) - '0');

                                seconds_seen = 1;
                                continue;
                        }
                }
        }
        if (hours_seen && minutes_seen && seconds_seen) {
                seconds += (hours * 3600) + (minutes * 60);
        } else {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                seconds = 0;
        }
        cob_set_int (curr_field, seconds);
        return curr_field;
}
```

7.36.2.53 cob_field* cob_intr_seconds_past_midnight (void)

Definition at line 2782 of file intrinsic.c.

```
{
        struct tm       *timeptr;
        time_t          t;
        int             seconds;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        t = time (NULL);
        timeptr = localtime (&t);
        seconds = (timeptr->tm_hour * 3600) + (timeptr->tm_min * 60) +
                        timeptr->tm_sec;
        cob_set_int (curr_field, seconds);
        return curr_field;
}
```

7.36.2.54 cob_field* cob_intr_sign (cob_field * *srcfield*)

Definition at line 566 of file intrinsic.c.

```
{
        int             n;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, COB_FLAG_HAVE_SIGN, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_set_int (curr_field, 0);
        n = cob_cmp (srcfield, curr_field);
        if (n < 0) {
                cob_set_int (curr_field, -1);
        } else if (n > 0) {
                cob_set_int (curr_field, 1);
        }

        return curr_field;
}
```

7.36.2.55 cob_field* cob_intr_sin (cob_field * *srcfield*)

Definition at line 1726 of file intrinsic.c.

```
{
        long long               result;
        double                  mathd2;
```

```
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = sin (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        result = (long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.36.2.56 cob_field* cob_intr_sqrt (cob_field * *srcfield*)

Definition at line 1759 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = sqrt (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.36.2.57 cob_field* cob_intr_standard_deviation (const int *params, ...*)

Definition at line 2472 of file intrinsic.c.

```
{
        cob_field       *f;
```

```
        va_list         args;
        int             i;

        va_start (args, params);
        make_double_entry ();

        if (params == 1) {
                va_end (args);
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        /* MEAN for all params */
        mpz_set_ui (d1.value, 0);
        d1.scale = 0;

        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                cob_decimal_add (&d1, &d2);
        }
        va_end (args);
        mpz_set_ui (d2.value, (unsigned int)params);
        d2.scale = 0;
        cob_decimal_div (&d1, &d2);

        /* Got the MEAN in d1, iterate again */

        mpz_set_ui (d4.value, 0);
        d4.scale = 0;

        va_start (args, params);

        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                cob_decimal_sub (&d2, &d1);
                cob_decimal_mul (&d2, &d2);
                cob_decimal_add (&d4, &d2);
        }
        va_end (args);

        mpz_set_ui (d3.value, (unsigned int)params);
        d3.scale = 0;
        cob_decimal_div (&d4, &d3);
        /* We have the VARIANCE in d4, sqrt = STANDARD-DEVIATION */

/* Do not know why this does not work
        d5.scale = d4.scale;
        mpz_mul_ui (d5.value, d4.value, 1000000000);
        mpz_mul_ui (d4.value, d5.value, 1000000000);
        mpz_sqrt (d5.value, d4.value);
        mpz_div_ui (d4.value, d5.value, 1000000000);
        cob_decimal_get_field (&d4, curr_field, 0);
        return curr_field;
*/

        cob_decimal_get_field (&d4, curr_field, 0);
        f = cob_intr_sqrt (curr_field);
        return f;
}
```

7.36.2.58 cob_field* cob_intr_stored_char_length (cob_field * *srcfield*)

Definition at line 1155 of file intrinsic.c.

```
{
        unsigned char   *p;
        int             count;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        count = srcfield->size;
        p = srcfield->data + srcfield->size - 1;
        for (; count > 0; count--, p--) {
                if (*p != ' ') {
                        break;
                }
        }
        cob_set_int (curr_field, count);
        return curr_field;
}
```

7.36.2.59 cob_field* cob_intr_substitute (const int *offset,* const int *length,* const int *params,* ...)

Definition at line 678 of file intrinsic.c.

```
{
        cob_field       *var;
        cob_field       **f1;
        cob_field       **f2;
        unsigned char   *p1;
        unsigned char   *p2;
        size_t          varsize;
        size_t          calcsize;
        size_t          n;
        size_t          found;
        int             numreps;
        int             i;
        cob_field_attr  attr;
        cob_field       field;
        va_list         args;

        numreps = params / 2;
        f1 = cob_malloc (numreps * sizeof (cob_field *));
        f2 = cob_malloc (numreps * sizeof (cob_field *));

        va_start (args, params);

        var = va_arg (args, cob_field *);
        varsize = var->size;

        /* Extract args */
        for (i = 0; i < params - 1; ++i) {
                if ((i % 2) == 0) {
```

```
                f1[i / 2] = va_arg (args, cob_field *);
        } else {
                f2[i / 2] = va_arg (args, cob_field *);
        }
}

/* Calculate required size */
calcsize = 0;
found = 0;
p1 = var->data;
for (n = 0; n < varsize; ) {
        for (i = 0; i < numreps; ++i) {
                if (n + f1[i]->size <= varsize) {
                        if (!memcmp (p1, f1[i]->data, f1[i]->size)) {
                                p1 += f1[i]->size;
                                n += f1[i]->size;
                                calcsize += f2[i]->size;
                                found = 1;
                                break;
                        }
                }
        }
        if (found) {
                found = 0;
                continue;
        }
        ++n;
        ++p1;
        ++calcsize;
}

COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
COB_FIELD_INIT (0, NULL, &attr);
field.size = calcsize;
make_field_entry (&field);

found = 0;
p1 = var->data;
p2 = curr_field->data;
for (n = 0; n < varsize; ) {
        for (i = 0; i < numreps; ++i) {
                if (n + f1[i]->size <= varsize) {
                        if (!memcmp (p1, f1[i]->data, f1[i]->size)) {
                                memcpy (p2, f2[i]->data, f2[i]->size);
                                p1 += f1[i]->size;
                                p2 += f2[i]->size;
                                n += f1[i]->size;
                                found = 1;
                                break;
                        }
                }
        }
        if (found) {
                found = 0;
                continue;
        }
        ++n;
        *p2++ = *p1++;
}
if (unlikely(offset > 0)) {
        calc_ref_mod (curr_field, offset, length);
}
```

```
        free (f1);
        free (f2);
        return curr_field;
}
```

7.36.2.60 cob_field* cob_intr_substitute_case (const int *offset*, const int *length*, const int *params*, ...)

Definition at line 775 of file intrinsic.c.

```
{
        cob_field       *var;
        cob_field       **f1;
        cob_field       **f2;
        unsigned char   *p1;
        unsigned char   *p2;
        size_t          varsize;
        size_t          calcsize;
        size_t          n;
        size_t          found;
        int             numreps;
        int             i;
        cob_field_attr  attr;
        cob_field       field;
        va_list         args;

        numreps = params / 2;
        f1 = cob_malloc (numreps * sizeof (cob_field *));
        f2 = cob_malloc (numreps * sizeof (cob_field *));

        va_start (args, params);

        var = va_arg (args, cob_field *);
        varsize = var->size;

        /* Extract args */
        for (i = 0; i < params - 1; ++i) {
                if ((i % 2) == 0) {
                        f1[i / 2] = va_arg (args, cob_field *);
                } else {
                        f2[i / 2] = va_arg (args, cob_field *);
                }
        }

        /* Calculate required size */
        calcsize = 0;
        found = 0;
        p1 = var->data;
        for (n = 0; n < varsize; ) {
                for (i = 0; i < numreps; ++i) {
                        if (n + f1[i]->size <= varsize) {
                                if (!strncasecmp ((const char *)p1,
                                                  (const char *)(f1[i]->data),
                                                  f1[i]->size)) {
                                        p1 += f1[i]->size;
                                        n += f1[i]->size;
                                        calcsize += f2[i]->size;
                                        found = 1;
                                        break;
```

```
                                }
                        }
                }
                if (found) {
                        found = 0;
                        continue;
                }
                ++n;
                ++p1;
                ++calcsize;
        }

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        field.size = calcsize;
        make_field_entry (&field);

        found = 0;
        p1 = var->data;
        p2 = curr_field->data;
        for (n = 0; n < varsize; ) {
                for (i = 0; i < numreps; ++i) {
                        if (n + f1[i]->size <= varsize) {
                                if (!strncasecmp ((const char *)p1,
                                                  (const char *)(f1[i]->data),
                                                  f1[i]->size)) {
                                        memcpy (p2, f2[i]->data, f2[i]->size);
                                        p1 += f1[i]->size;
                                        p2 += f2[i]->size;
                                        n += f1[i]->size;
                                        found = 1;
                                        break;
                                }
                        }
                }
                if (found) {
                        found = 0;
                        continue;
                }
                ++n;
                *p2++ = *p1++;
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        free (f1);
        free (f2);
        return curr_field;
}
```

7.36.2.61 cob_field* cob_intr_sum (const int *params*, ...)

Definition at line 1996 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        int             i;
        int             digits = 0;
```

```
        int             scale = 0;
        cob_field_attr  attr;
        cob_field       field;


        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        mpz_set_ui (d1.value, 0);
        d1.scale = 0;

        va_start (args, params);

        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if ((COB_FIELD_DIGITS(f) - COB_FIELD_SCALE(f)) > digits) {
                        digits = COB_FIELD_DIGITS(f) - COB_FIELD_SCALE(f);
                }
                if (COB_FIELD_SCALE(f) > scale) {
                        scale = COB_FIELD_SCALE(f);
                }
                cob_decimal_set_field (&d2, f);
                cob_decimal_add (&d1, &d2);
        }
        va_end (args);

        attr.scale = scale;
        make_field_entry (&field);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.36.2.62 cob_field* cob_intr_tan (cob_field * *srcfield*)

Definition at line 1777 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = tan (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.36.2.63 cob_field* cob_intr_test_date_yyyymmdd (cob_field * *srcfield*)

Definition at line 1409 of file intrinsic.c.

```
{
        int             indate;
        int             days;
        int             month;
        int             year;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        /* Base 1601-01-01 */
        indate = cob_get_int (srcfield);
        year = indate / 10000;
        if (year < 1601 || year > 9999) {
                cob_set_int (curr_field, 1);
                return curr_field;
        }
        indate %= 10000;
        month = indate / 100;
        if (month < 1 || month > 12) {
                cob_set_int (curr_field, 2);
                return curr_field;
        }
        days = indate % 100;
        if (days < 1 || days > 31) {
                cob_set_int (curr_field, 3);
                return curr_field;
        }
        if (leap_year (year)) {
                if (days > leap_month_days[month]) {
                        cob_set_int (curr_field, 3);
                        return curr_field;
                }
        } else {
                if (days > normal_month_days[month]) {
                        cob_set_int (curr_field, 3);
                        return curr_field;
                }
        }
        cob_set_int (curr_field, 0);
        return curr_field;
}
```

7.36.2.64 cob_field* cob_intr_test_day_yyyyddd (cob_field * *srcfield*)

Definition at line 1456 of file intrinsic.c.

```
{
        int             indate;
        int             days;
        int             year;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);
```

```
        /* Base 1601-01-01 */
        indate = cob_get_int (srcfield);
        year = indate / 1000;
        if (year < 1601 || year > 9999) {
                cob_set_int (curr_field, 1);
                return curr_field;
        }
        days = indate % 1000;
        if (days < 1 || days > 365 + leap_year (year)) {
                cob_set_int (curr_field, 2);
                return curr_field;
        }
        cob_set_int (curr_field, 0);
        return curr_field;
}
```

7.36.2.65 cob_field* cob_intr_trim (const int *offset*, const int *length*, cob_field * *srcfield*, const int *direction*)

Definition at line 876 of file intrinsic.c.

```
{
        unsigned char   *begin;
        unsigned char   *end;
        size_t          i;
        size_t          size = 0;

        make_field_entry (srcfield);

        for (i = 0; i < srcfield->size; ++i) {
                if (srcfield->data[i] != ' ') {
                        break;
                }
        }
        if (i == srcfield->size) {
                curr_field->size = 1;
                curr_field->data[0] = ' ';
                return curr_field;
        }
        begin = srcfield->data;
        if (direction != 2) {
                for (; *begin == ' '; ++begin) ;
        }
        end = srcfield->data + srcfield->size - 1;
        if (direction != 1) {
                for (; *end == ' '; end--) ;
        }
        for (i = 0; begin <= end; ++begin, ++i) {
                curr_field->data[i] = *begin;
                ++size;
        }
        curr_field->size = size;
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.36.2.66 cob_field* cob_intr_upper_case (const int *offset*, const int *length*, cob_field * *srcfield*)

Definition at line 588 of file intrinsic.c.

```
{
        size_t          i, size;

        make_field_entry (srcfield);

        size = srcfield->size;
        for (i = 0; i < size; ++i) {
                curr_field->data[i] = toupper (srcfield->data[i]);
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.36.2.67 cob_field* cob_intr_variance (const int *params*, ...)

Definition at line 2403 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        long long       n;
        union {
                unsigned char   data[8];
                long long       datall;
        } datun;
        int             i;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        if (params == 1) {
                make_field_entry (&field);
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        /* MEAN for all params */
        mpz_set_ui (d1.value, 0);
        d1.scale = 0;

        va_start (args, params);
        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                cob_decimal_add (&d1, &d2);
        }
        va_end (args);
        mpz_set_ui (d2.value, (unsigned int)params);
        d2.scale = 0;
```

```
        cob_decimal_div (&d1, &d2);

        /* Got the MEAN in d1, iterate again */

        mpz_set_ui (d4.value, 0);
        d4.scale = 0;

        va_start (args, params);

        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                cob_decimal_sub (&d2, &d1);
                cob_decimal_mul (&d2, &d2);
                cob_decimal_add (&d4, &d2);
        }
        va_end (args);

        mpz_set_ui (d3.value, (unsigned int)params);
        d3.scale = 0;
        cob_decimal_div (&d4, &d3);
        field.data = datun.data;
        cob_decimal_get_field (&d4, &field, 0);
        n = datun.datall;
        for (i = 0; n; n /= 10, ++i) ;
        field.data = NULL;
        if (i <= 18) {
                attr.scale = 18 - i;
        }
        make_field_entry (&field);
        cob_decimal_get_field (&d4, curr_field, 0);
        return curr_field;
}
```

7.36.2.68 cob_field* cob_intr_when_compiled (const int *offset*, const int *length*, cob_field * *f*)

Definition at line 1020 of file intrinsic.c.

```
{
        make_field_entry (f);

        memcpy (curr_field->data, f->data, f->size);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.36.2.69 cob_field* cob_intr_year_to_yyyy (const int *params*, ...)

Definition at line 2583 of file intrinsic.c.

```
{
        cob_field       *f;
```

```
        struct tm       *timeptr;
        va_list         args;
        time_t          t;
        int             year;
        int             interval;
        int             xqtyear;
        int             maxyear;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        va_start (args, params);
        f = va_arg (args, cob_field *);
        year = cob_get_int (f);
        if (params > 1) {
                f = va_arg (args, cob_field *);
                interval = cob_get_int (f);
        } else {
                interval = 50;
        }
        if (params > 2) {
                f = va_arg (args, cob_field *);
                xqtyear = cob_get_int (f);
        } else {
                t = time (NULL);
                timeptr = localtime (&t);
                xqtyear = 1900 + timeptr->tm_year;
        }
        va_end (args);

        if (year < 0 || year > 99) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (xqtyear < 1601 || xqtyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        maxyear = xqtyear + interval;
        if (maxyear < 1700 || maxyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (maxyear % 100 >= year) {
                year += 100 * (maxyear / 100);
        } else {
                year += 100 * ((maxyear / 100) - 1);
        }
        cob_set_int (curr_field, year);
        return curr_field;
}
```

7.36.3 Variable Documentation

7.36.3.1 cob_decimal d2

Definition at line 68 of file intrinsic.c.

7.36.3.2 cob_decimal d3

Definition at line 68 of file intrinsic.c.

7.36.3.3 cob_decimal d4

Definition at line 68 of file intrinsic.c.

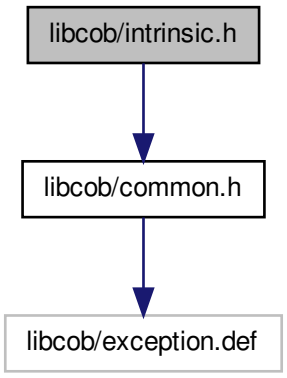
7.36.3.4 cob_decimal d5

Definition at line 68 of file intrinsic.c.

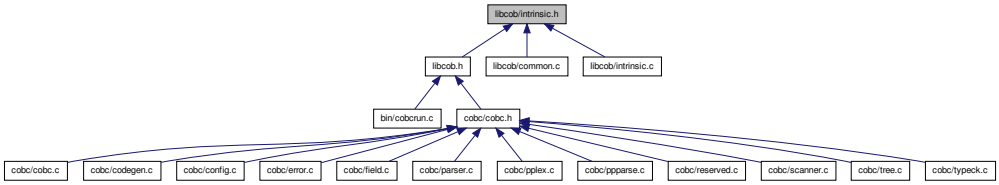
7.37 libcob/intrinsic.h File Reference

```
#include <libcob/common.h>
```

Include dependency graph for intrinsic.h:



This graph shows which files directly or indirectly include this file:



Functions

- cob_field * cob_intr_binop (cob_field *, int, cob_field *)
- cob_field * cob_intr_current_date (const int, const int)
- cob_field * cob_intr_when_compiled (const int, const int, cob_field *)
- cob_field * cob_intr_exception_file (void)
- cob_field * cob_intr_exception_location (void)
- cob_field * cob_intr_exception_status (void)
- cob_field * cob_intr_exception_statement (void)
- cob_field * cob_intr_char (cob_field *)
- cob_field * cob_intr_ord (cob_field *)
- cob_field * cob_intr_stored_char_length (cob_field *)
- cob_field * cob_intr_combined_datetime (cob_field *, cob_field *)
- cob_field * cob_intr_date_of_integer (cob_field *)
- cob_field * cob_intr_day_of_integer (cob_field *)
- cob_field * cob_intr_integer_of_date (cob_field *)
- cob_field * cob_intr_integer_of_day (cob_field *)
- cob_field * cob_intr_test_date_yyyymmdd (cob_field *)
- cob_field * cob_intr_test_day_yyyyddd (cob_field *)
- cob_field * cob_intr_factorial (cob_field *)
- cob_field * cob_intr_exp (cob_field *)
- cob_field * cob_intr_exp10 (cob_field *)
- cob_field * cob_intr_abs (cob_field *)
- cob_field * cob_intr_acos (cob_field *)
- cob_field * cob_intr_asin (cob_field *)
- cob_field * cob_intr_atan (cob_field *)
- cob_field * cob_intr_cos (cob_field *)
- cob_field * cob_intr_log (cob_field *)
- cob_field * cob_intr_log10 (cob_field *)
- cob_field * cob_intr_sin (cob_field *)
- cob_field * cob_intr_sqrt (cob_field *)
- cob_field * cob_intr_tan (cob_field *)
- cob_field * cob_intr_upper_case (const int, const int, cob_field *)
- cob_field * cob_intr_lower_case (const int, const int, cob_field *)
- cob_field * cob_intr_reverse (const int, const int, cob_field *)

- cob_field * cob_intr_concatenate (const int, const int, const int,...)
- cob_field * cob_intr_substitute (const int, const int, const int,...)
- cob_field * cob_intr_substitute_case (const int, const int, const int,...)
- cob_field * cob_intr_trim (const int, const int, cob_field *, const int)
- cob_field * cob_intr_length (cob_field *)
- cob_field * cob_intr_integer (cob_field *)
- cob_field * cob_intr_integer_part (cob_field *)
- cob_field * cob_intr_fraction_part (cob_field *)
- cob_field * cob_intr_sign (cob_field *)
- cob_field * cob_intr_numval (cob_field *)
- cob_field * cob_intr_numval_c (cob_field *, cob_field *)
- cob_field * cob_intr_annuity (cob_field *, cob_field *)
- cob_field * cob_intr_mod (cob_field *, cob_field *)
- cob_field * cob_intr_rem (cob_field *, cob_field *)
- cob_field * cob_intr_sum (const int,...)
- cob_field * cob_intr_ord_min (const int,...)
- cob_field * cob_intr_ord_max (const int,...)
- cob_field * cob_intr_min (const int,...)
- cob_field * cob_intr_max (const int,...)
- cob_field * cob_intr_midrange (const int,...)
- cob_field * cob_intr_median (const int,...)
- cob_field * cob_intr_mean (const int,...)
- cob_field * cob_intr_range (const int,...)
- cob_field * cob_intr_random (const int,...)
- cob_field * cob_intr_variance (const int,...)
- cob_field * cob_intr_standard_deviation (const int,...)
- cob_field * cob_intr_present_value (const int,...)
- cob_field * cob_intr_year_to_yyyy (const int,...)
- cob_field * cob_intr_date_to_yyyymmdd (const int,...)
- cob_field * cob_intr_day_to_yyyyddd (const int,...)
- cob_field * cob_intr_locale_date (const int, const int, cob_field *, cob_field *)
- cob_field * cob_intr_locale_time (const int, const int, cob_field *, cob_field *)
- cob_field * cob_intr_seconds_past_midnight (void)
- cob_field * cob_intr_lcl_time_from_secs (const int, const int, cob_field *, cob_field *)
- cob_field * cob_intr_seconds_from_formatted_time (cob_field *, cob_field *)

7.37.1 Function Documentation

7.37.1.1 cob_field* cob_intr_abs (cob_field *)

Definition at line 1545 of file intrinsic.c.

```
{

        make_field_entry (srcfield);

        cob_decimal_set_field (&d1, srcfield);
        mpz_abs (d1.value, d1.value);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.2 cob_field* cob_intr_acos (cob_field *)

Definition at line 1557 of file intrinsic.c.

```
{
        unsigned long long      result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, 0, NULL);
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = acos (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        result = (unsigned long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.37.1.3 cob_field* cob_intr_annuity (cob_field *, cob_field *)

Definition at line 1974 of file intrinsic.c.

```
{
        double          mathd1, mathd2;

        make_double_entry ();
```

```
        cob_decimal_set_field (&d1, srcfield1);
        cob_decimal_set_field (&d2, srcfield2);

        mathd1 = intr_get_double (&d1);
        mathd2 = intr_get_double (&d2);
        if (mathd1 == 0) {
                mathd1 = 1.0 / mathd2;
                memcpy (curr_field->data, (char *)&mathd1, sizeof (double));
                return curr_field;
        }
        mathd1 /= (1.0 - pow (mathd1 + 1.0, 0.0 - mathd2));
        memcpy (curr_field->data, (char *)&mathd1, sizeof (double));
        return curr_field;
}
```

7.37.1.4 cob_field* cob_intr_asin (cob_field *)

Definition at line 1591 of file intrinsic.c.

```
{
        long long               result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = asin (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        result = (long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.37.1.5 cob_field* cob_intr_atan (cob_field *)

Definition at line 1624 of file intrinsic.c.

```
{
```

```
        long long               result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = atan (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        result = (long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.37.1.6 cob_field* cob_intr_binop (cob_field *, int, cob_field *)

Definition at line 432 of file intrinsic.c.

```
{
/* RXW
        size_t          bitnum;
        size_t          sign;
        size_t          attrsign;
        cob_field_attr  attr;
        cob_field       field;
*/

        cob_decimal_set_field (&d1, f1);
        cob_decimal_set_field (&d2, f2);
        switch (op) {
        case '+':
                cob_decimal_add (&d1, &d2);
                break;
        case '-':
                cob_decimal_sub (&d1, &d2);
                break;
        case '*':
                cob_decimal_mul (&d1, &d2);
                break;
        case '/':
                cob_decimal_div (&d1, &d2);
                break;
```

```
        case '^':
                cob_decimal_pow (&d1, &d2);
                break;
        default:
                break;
        }

/* RXW
        if (mpz_sgn (d1.value) < 0) {
                attrsign = COB_FLAG_HAVE_SIGN;
                sign = 1;
        } else {
                attrsign = 0;
                sign = 0;
        }
        bitnum = mpz_sizeinbase (d1.value, 2);
        if (bitnum < 33 - sign) {
                COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, attrsign, NULL);
                COB_FIELD_INIT (4, NULL, &attr);
                attr.scale = d1.scale;
                make_field_entry (&field);
        } else if (bitnum < 65 - sign) {
                COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, attrsign, NULL);
                COB_FIELD_INIT (8, NULL, &attr);
                attr.scale = d1.scale;
                make_field_entry (&field);
        } else {
*/
                make_double_entry ();
/* RXW
        }
*/
        cob_decimal_get_field (&d1, curr_field, 0);

        return curr_field;
}
```

7.37.1.7 cob_field* cob_intr_char (cob_field *)

Definition at line 1121 of file intrinsic.c.

```
{
        int             i;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (1, NULL, &attr);
        make_field_entry (&field);

        i = cob_get_int (srcfield);
        if (i < 1 || i > 256) {
                *curr_field->data = 0;
        } else {
                *curr_field->data = i - 1;
        }
        return curr_field;
}
```

7.37.1.8 cob_field* cob_intr_combined_datetime (cob_field *, cob_field *)

Definition at line 1178 of file intrinsic.c.

```
{
        int             srdays;
        int             srtime;
        cob_field_attr  attr;
        cob_field       field;
        char            buff[16];

        COB_ATTR_INIT (COB_TYPE_NUMERIC_DISPLAY, 12, 5, 0, NULL);
        COB_FIELD_INIT (12, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        srdays = cob_get_int (srcdays);
        if (srdays < 1 || srdays > 3067671) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                memset (curr_field->data, '0', 12);
                return curr_field;
        }
        srtime = cob_get_int (srctime);
        if (srtime < 1 || srtime > 86400) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                memset (curr_field->data, '0', 12);
                return curr_field;
        }
        snprintf (buff, 15, "%7.7d%5.5d", srdays, srtime);
        memcpy (curr_field->data, buff, 12);
        return curr_field;
}
```

7.37.1.9 cob_field* cob_intr_concatenate (const int , const int , const int , ...)

Definition at line 639 of file intrinsic.c.

```
{
        cob_field       **f;
        unsigned char   *p;
        size_t          calcsize;
        int             i;
        cob_field_attr  attr;
        cob_field       field;
        va_list         args;

        f = cob_malloc (params * sizeof (cob_field *));

        va_start (args, params);

        /* Extract args / calculate size */
        calcsize = 0;
        for (i = 0; i < params; ++i) {
                f[i] = va_arg (args, cob_field *);
                calcsize += f[i]->size;
        }

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
```

```
        COB_FIELD_INIT (calcsize, NULL, &attr);
        make_field_entry (&field);

        p = curr_field->data;
        for (i = 0; i < params; ++i) {
                memcpy (p, f[i]->data, f[i]->size);
                p += f[i]->size;
        }

        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        free (f);
        return curr_field;
}
```

7.37.1.10 cob_field* cob_intr_cos (cob_field *)

Definition at line 1657 of file intrinsic.c.

```
{
        long long               result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = cos (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        result = (long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.37.1.11 cob_field* cob_intr_current_date (const int , const int)

Definition at line 1032 of file intrinsic.c.

```
{
#if     defined(_WIN32) && !defined(__CYGWIN__)
        long            contz;
        struct tm       *tmptr;
        struct _timeb   tmb;
        cob_field_attr  attr;
        cob_field       field;
#else
#if !defined(__linux__) && !defined(__CYGWIN__) && !defined(COB_STRFTIME) && defi
      ned(HAVE_TIMEZONE)
        struct tm       *tmptr;
        long            contz;
#endif
        time_t          curtime;
        cob_field_attr  attr;
        cob_field       field;
#if defined(HAVE_SYS_TIME_H) && defined(HAVE_GETTIMEOFDAY)
        struct timeval  tmv;
        char            buff2[8];
#endif
#endif  /* _WIN32 */
        char            buff[24];

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (21, NULL, &attr);
        make_field_entry (&field);
        memset (buff, 0, sizeof(buff));

#if     defined(_WIN32) && !defined(__CYGWIN__)
        _ftime (&tmb);
        tmptr = localtime (&(tmb.time));
        if (tmb.timezone <= 0) {
                contz = -tmb.timezone;
                snprintf (buff, 23,
                        "%4.4d%2.2d%2.2d%2.2d%2.2d%2.2d%2.2d+%2.2ld%2.2ld",
                        tmptr->tm_year + 1900, tmptr->tm_mon + 1, tmptr->tm_mday,

                        tmptr->tm_hour, tmptr->tm_min, tmptr->tm_sec,
                        tmb.millitm / 100, contz / 60, contz % 60);
        } else {
                contz = tmb.timezone;
                snprintf (buff, 23,
                        "%4.4d%2.2d%2.2d%2.2d%2.2d%2.2d%2.2d-%2.2ld%2.2ld",
                        tmptr->tm_year + 1900, tmptr->tm_mon + 1, tmptr->tm_mday,

                        tmptr->tm_hour, tmptr->tm_min, tmptr->tm_sec,
                        tmb.millitm / 100, contz / 60, contz % 60);
        }
#else
#if defined(HAVE_SYS_TIME_H) && defined(HAVE_GETTIMEOFDAY)
        gettimeofday (&tmv, NULL);
        curtime = tmv.tv_sec;
#else
        curtime = time (NULL);
#endif

#if defined(__linux__) || defined(__CYGWIN__) || defined(COB_STRFTIME)
        strftime (buff, 22, "%Y%m%d%H%M%S00%z", localtime (&curtime));
#elif defined(HAVE_TIMEZONE)
        tmptr = localtime (&curtime);
        strftime (buff, 17, "%Y%m%d%H%M%S00", tmptr);
        /* RXW - Hack for DST - Need something better */
```

```
        if (tmptr->tm_isdst > 0) {
                timezone -= 3600;
        }
        if (timezone <= 0) {
                contz = -timezone;
                buff[16] = '+';
        } else {
                contz = timezone;
                buff[16] = '-';
        }
        sprintf(&buff[17], "%2.2ld%2.2ld", contz / 3600, (contz % 3600) / 60);
#else
        strftime (buff, 22, "%Y%m%d%H%M%S0000000", localtime (&curtime));
#endif

#if defined(HAVE_SYS_TIME_H) && defined(HAVE_GETTIMEOFDAY)
        snprintf(buff2, 7, "%2.2ld", tmv.tv_usec / 10000);
        memcpy (&buff[14], buff2, 2);
#endif
#endif  /* _WIN32 */

        memcpy (curr_field->data, buff, 21);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.37.1.12 cob_field* cob_intr_date_of_integer (cob_field *)

Definition at line 1209 of file intrinsic.c.

```
{
        int             i;
        int             days;
        int             baseyear = 1601;
        int             leapyear = 365;
        cob_field_attr  attr;
        cob_field       field;
        char            buff[16];

        COB_ATTR_INIT (COB_TYPE_NUMERIC_DISPLAY, 8, 0, 0, NULL);
        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        /* Base 1601-01-01 */
        days = cob_get_int (srcdays);
        if (days < 1 || days > 3067671) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                memset (curr_field->data, '0', 8);
                return curr_field;
        }
        while (days > leapyear) {
                days -= leapyear;
                ++baseyear;
                if (leap_year (baseyear)) {
                        leapyear = 366;
                } else {
```

```
                        leapyear = 365;
                }
        }
        for (i = 0; i < 13; ++i) {
                if (leap_year (baseyear)) {
                        if (days <= leap_days[i]) {
                                days -= leap_days[i-1];
                                break;
                        }
                } else {
                        if (days <= normal_days[i]) {
                                days -= normal_days[i-1];
                                break;
                        }
                }
        }
        snprintf (buff, 15, "%4.4d%2.2d%2.2d", baseyear, i, days);
        memcpy (curr_field->data, buff, 8);
        return curr_field;
}
```

7.37.1.13 cob_field* cob_intr_date_to_yyyymmdd (const int , ...)

Definition at line 2646 of file intrinsic.c.

```
{
        cob_field       *f;
        struct tm       *timeptr;
        va_list         args;
        time_t          t;
        int             year;
        int             mmdd;
        int             interval;
        int             xqtyear;
        int             maxyear;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        va_start (args, params);
        f = va_arg (args, cob_field *);
        year = cob_get_int (f);
        mmdd = year % 10000;
        year /= 10000;
        if (params > 1) {
                f = va_arg (args, cob_field *);
                interval = cob_get_int (f);
        } else {
                interval = 50;
        }
        if (params > 2) {
                f = va_arg (args, cob_field *);
                xqtyear = cob_get_int (f);
        } else {
                t = time (NULL);
```

```
                timeptr = localtime (&t);
                xqtyear = 1900 + timeptr->tm_year;
        }
        va_end (args);

        if (year < 0 || year > 999999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (xqtyear < 1601 || xqtyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        maxyear = xqtyear + interval;
        if (maxyear < 1700 || maxyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (maxyear % 100 >= year) {
                year += 100 * (maxyear / 100);
        } else {
                year += 100 * ((maxyear / 100) - 1);
        }
        year *= 10000;
        year += mmdd;
        cob_set_int (curr_field, year);
        return curr_field;
}
```

7.37.1.14 cob_field* cob_intr_day_of_integer (cob_field *)

Definition at line 1259 of file intrinsic.c.

```
{
        int             days;
        int             baseyear = 1601;
        int             leapyear = 365;
        cob_field_attr  attr;
        cob_field       field;
        char            buff[16];

        COB_ATTR_INIT (COB_TYPE_NUMERIC_DISPLAY, 7, 0, 0, NULL);
        COB_FIELD_INIT (7, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        /* Base 1601-01-01 */
        days = cob_get_int (srcdays);
        if (days < 1 || days > 3067671) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                memset (curr_field->data, '0', 7);
                return curr_field;
        }
        while (days > leapyear) {
                days -= leapyear;
                ++baseyear;
```

```
                if (leap_year (baseyear)) {
                        leapyear = 366;
                } else {
                        leapyear = 365;
                }
        }
        snprintf (buff, 15, "%4.4d%3.3d", baseyear, days);
        memcpy (curr_field->data, buff, 7);
        return curr_field;
}
```

7.37.1.15 cob_field* cob_intr_day_to_yyyyddd (const int , ...)

Definition at line 2714 of file intrinsic.c.

```
{
        cob_field       *f;
        struct tm       *timeptr;
        va_list         args;
        time_t          t;
        int             year;
        int             days;
        int             interval;
        int             xqtyear;
        int             maxyear;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        va_start (args, params);
        f = va_arg (args, cob_field *);
        year = cob_get_int (f);
        days = year % 1000;
        year /= 1000;
        if (params > 1) {
                f = va_arg (args, cob_field *);
                interval = cob_get_int (f);
        } else {
                interval = 50;
        }
        if (params > 2) {
                f = va_arg (args, cob_field *);
                xqtyear = cob_get_int (f);
        } else {
                t = time (NULL);
                timeptr = localtime (&t);
                xqtyear = 1900 + timeptr->tm_year;
        }
        va_end (args);

        if (year < 0 || year > 999999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
```

```
        if (xqtyear < 1601 || xqtyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        maxyear = xqtyear + interval;
        if (maxyear < 1700 || maxyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (maxyear % 100 >= year) {
                year += 100 * (maxyear / 100);
        } else {
                year += 100 * ((maxyear / 100) - 1);
        }
        year *= 1000;
        year += days;
        cob_set_int (curr_field, year);
        return curr_field;
}
```

7.37.1.16 cob_field* cob_intr_exception_file (void)

Definition at line 916 of file intrinsic.c.

```
{
        size_t          flen;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        if (cob_exception_code == 0 || !cob_error_file ||
            (cob_exception_code & 0x0500) != 0x0500) {
                field.size = 2;
                make_field_entry (&field);
                memcpy (curr_field->data, "00", 2);
        } else {
                flen = strlen (cob_error_file->select_name);
                field.size = flen + 2;
                make_field_entry (&field);
                memcpy (curr_field->data, cob_error_file->file_status, 2);
                memcpy (&(curr_field->data[2]), cob_error_file->select_name, flen
      );
        }
        return curr_field;
}
```

7.37.1.17 cob_field* cob_intr_exception_location (void)

Definition at line 940 of file intrinsic.c.

```
{
        cob_field_attr  attr;
```

```
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        if (!cob_got_exception || !cob_orig_program_id) {
                field.size = 1;
                make_field_entry (&field);
                *(curr_field->data) = ' ';
                return curr_field;
        }
        memset (locale_buff, 0, COB_SMALL_BUFF);
        if (cob_orig_section && cob_orig_paragraph) {
                snprintf (locale_buff, COB_SMALL_MAX, "%s; %s OF %s; %d",
                          cob_orig_program_id, cob_orig_paragraph,
                          cob_orig_section, cob_orig_line);
        } else if (cob_orig_section) {
                snprintf (locale_buff, COB_SMALL_MAX, "%s; %s; %d",
                          cob_orig_program_id, cob_orig_section, cob_orig_line);
        } else if (cob_orig_paragraph) {
                snprintf (locale_buff, COB_SMALL_MAX, "%s; %s; %d",
                          cob_orig_program_id, cob_orig_paragraph, cob_orig_line)
      ;
        } else {
                snprintf (locale_buff, COB_SMALL_MAX, "%s; ; %d",
                          cob_orig_program_id, cob_orig_line);
        }
        field.size = strlen (locale_buff);
        make_field_entry (&field);
        memcpy (curr_field->data, locale_buff, field.size);
        return curr_field;
}
```

7.37.1.18 cob_field* cob_intr_exception_statement (void)

Definition at line 997 of file intrinsic.c.

```
{
        size_t          flen;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (31, NULL, &attr);
        make_field_entry (&field);

        memset (curr_field->data, ' ', 31);
        if (cob_exception_code && cob_orig_statement) {
                flen = strlen (cob_orig_statement);
                if (flen > 31) {
                        memcpy (curr_field->data, cob_orig_statement, 31);
                } else {
                        memcpy (curr_field->data, cob_orig_statement, flen);
                }
        }
        return curr_field;
}
```

7.37.1.19 cob_field* cob_intr_exception_status (void)

Definition at line 975 of file intrinsic.c.

```
{
        const char      *except_name;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (31, NULL, &attr);
        make_field_entry (&field);

        memset (curr_field->data, ' ', 31);
        if (cob_exception_code) {
                except_name = cob_get_exception_name (cob_exception_code);
                if (except_name == NULL) {
                        except_name = "EXCEPTION-OBJECT";
                }
                memcpy (curr_field->data, except_name, strlen (except_name));
        }
        return curr_field;
}
```

7.37.1.20 cob_field* cob_intr_exp (cob_field *)

Definition at line 1509 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = pow (2.7182818284590452354, intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.37.1.21 cob_field* cob_intr_exp10 (cob_field *)

Definition at line 1527 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();
```

```
        errno = 0;
        mathd2 = pow (10.0, intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.37.1.22 cob_field* cob_intr_factorial (cob_field *)

Definition at line 1485 of file intrinsic.c.

```
{
        int             srcval;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, 0, NULL);
        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        srcval = cob_get_int (srcfield);
        if (srcval < 0) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        d1.scale = 0;
        mpz_fac_ui (d1.value, (unsigned int)srcval);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.23 cob_field* cob_intr_fraction_part (cob_field *)

Definition at line 552 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 18, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_move (srcfield, curr_field);
        return curr_field;
}
```

7.37.1.24 cob_field* cob_intr_integer (cob_field *)

Definition at line 511 of file intrinsic.c.

```
{
        int             i, scale;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_decimal_set_field (&d1, srcfield);
        if (mpz_sgn (d1.value) >= 0) {
                cob_decimal_get_field (&d1, curr_field, 0);
                return curr_field;
        }
        scale = 1;
        for (i = 0; i < d1.scale; ++i) {
                scale *= 10;
        }
        if (mpz_fdiv_ui (d1.value, (unsigned int)scale)) {
                mpz_sub_ui (d1.value, d1.value, (unsigned int)scale);
        }
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.25 cob_field* cob_intr_integer_of_date (cob_field *)

Definition at line 1295 of file intrinsic.c.

```
{
        int             indate;
        int             days;
        int             totaldays;
        int             month;
        int             year;
        int             baseyear = 1601;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        /* Base 1601-01-01 */
        indate = cob_get_int (srcfield);
        year = indate / 10000;
        if (year < 1601 || year > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        indate %= 10000;
```

```
        month = indate / 100;
        if (month < 1 || month > 12) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        days = indate % 100;
        if (days < 1 || days > 31) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (leap_year (year)) {
                if (days > leap_month_days[month]) {
                        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                        cob_set_int (curr_field, 0);
                        return curr_field;
                }
        } else {
                if (days > normal_month_days[month]) {
                        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                        cob_set_int (curr_field, 0);
                        return curr_field;
                }
        }
        totaldays = 0;
        while (baseyear != year) {
                if (leap_year (baseyear)) {
                        totaldays += 366;
                } else {
                        totaldays += 365;
                }
                ++baseyear;
        }
        if (leap_year (baseyear)) {
                totaldays += leap_days[month - 1];
        } else {
                totaldays += normal_days[month - 1];
        }
        totaldays += days;
        cob_set_int (curr_field, totaldays);
        return curr_field;
}
```

7.37.1.26 cob_field* cob_intr_integer_of_day (cob_field *)

Definition at line 1365 of file intrinsic.c.

```
{
        int             indate;
        int             days;
        int             totaldays;
        int             year;
        int             baseyear = 1601;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
```

```
        make_field_entry (&field);

        cob_exception_code = 0;
        /* Base 1601-01-01 */
        indate = cob_get_int (srcfield);
        year = indate / 1000;
        if (year < 1601 || year > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        days = indate % 1000;
        if (days < 1 || days > 365 + leap_year (year)) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        totaldays = 0;
        while (baseyear != year) {
                if (leap_year (baseyear)) {
                        totaldays += 366;
                } else {
                        totaldays += 365;
                }
                ++baseyear;
        }
        totaldays += days;
        cob_set_int (curr_field, totaldays);
        return curr_field;
}
```

7.37.1.27 cob_field* cob_intr_integer_part (cob_field *)

Definition at line 538 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        cob_move (srcfield, curr_field);
        return curr_field;
}
```

7.37.1.28 cob_field* cob_intr_lcl_time_from_secs (const int , const int , cob_field * , cob_field *)

Definition at line 3149 of file intrinsic.c.

```
{
        cob_field_attr  attr;
```

```
        cob_field       field;
#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        size_t          len;
        int             indate;
        int             hours;
        int             minutes;
        int             seconds;
#ifdef  HAVE_LANGINFO_CODESET
        char            *deflocale = NULL;
        char            *localep = NULL;
        char            *localep2;
        struct tm       tstruct;
        char            buff2[128];
#else
        char            *p;
        LCID            localeid = LOCALE_USER_DEFAULT;
        SYSTEMTIME      syst;
#endif
        char            buff[128];
#endif

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        cob_exception_code = 0;

#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        if (COB_FIELD_IS_NUMERIC (srcfield)) {
                indate = cob_get_int (srcfield);
        } else {
                goto derror;
        }
        if (indate > 86400) {
                goto derror;
        }
        hours = indate / 3600;
        indate %= 3600;
        minutes = indate / 60;
        seconds = indate % 60;

#ifdef  HAVE_LANGINFO_CODESET
        memset ((void *)&tstruct, 0, sizeof(struct tm));
        tstruct.tm_hour = hours;
        tstruct.tm_min = minutes;
        tstruct.tm_sec = seconds;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                deflocale = locale_buff;
                localep2 = setlocale (LC_TIME, NULL);
                if (localep2) {
                        localep = strdup (localep2);
                }
                (void) setlocale (LC_TIME, deflocale);
        }
        memset (buff2, 0, sizeof(buff2));
        snprintf(buff2, sizeof(buff2) - 1, "%s", nl_langinfo(T_FMT));
        if (deflocale) {
                if (localep) {
                        (void) setlocale (LC_TIME, localep);
                }
```

```
        }
        strftime (buff, sizeof(buff), buff2, &tstruct);
#else
        memset ((void *)&syst, 0, sizeof(syst));
        syst.wHour = hours;
        syst.wMinute = minutes;
        syst.wSecond = seconds;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                for (p = locale_buff; *p; ++p) {
                        if (isalnum(*p) || *p == '_') {
                                continue;
                        }
                        break;
                }
                *p = 0;
                for (len = 0; len < WINLOCSIZE; ++len) {
                        if (!strcmp(locale_buff, wintable[len].winlocalename)) {
                                localeid = wintable[len].winlocaleid;
                                break;
                        }
                }
                if (len == WINLOCSIZE) {
                        goto derror;
                }
        }
        if (!GetTimeFormat (localeid, LOCALE_NOUSEROVERRIDE, &syst, NULL, buff, s
      izeof(buff))) {

                goto derror;
        }
#endif
        len = strlen (buff);
        field.size = len;
        make_field_entry (&field);
        memcpy (curr_field->data, buff, len);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
derror:
#endif
        field.size = 10;
        make_field_entry (&field);
        memset (curr_field->data, ' ', 10);
        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
        return curr_field;
}
```

7.37.1.29 cob_field* cob_intr_length (cob_field *)

Definition at line 497 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;
```

```
        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_set_int (curr_field, (int)srcfield->size);
        return curr_field;
}
```

7.37.1.30 cob_field* cob_intr_locale_date (const int , const int , cob_field * , cob_field *)

Definition at line 2866 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;
#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        size_t          len;
        int             indate;
        int             days;
        int             month;
        int             year;
#ifdef  HAVE_LANGINFO_CODESET
        unsigned char   *p;
        char            *deflocale = NULL;
        char            *localep = NULL;
        char            *localep2;
        struct tm       tstruct;
        char            buff2[128];
#else
        char            *p;
        LCID            localeid = LOCALE_USER_DEFAULT;
        SYSTEMTIME      syst;
#endif
        char            buff[128];
#endif

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        cob_exception_code = 0;

#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        if (COB_FIELD_IS_NUMERIC (srcfield)) {
                indate = cob_get_int (srcfield);
        } else {
                if (srcfield->size < 8) {
                        goto derror;
                }
                p = srcfield->data;
                indate = 0;
                for (len = 0; len < 8; ++len, ++p) {
                        if (isdigit (*p)) {
                                indate *= 10;
                                indate += (*p - '0');
                        } else {
                                goto derror;
                        }
                }
```

```
        }
        year = indate / 10000;
        if (year < 1601 || year > 9999) {
                goto derror;
        }
        indate %= 10000;
        month = indate / 100;
        if (month < 1 || month > 12) {
                goto derror;
        }
        days = indate % 100;
        if (days < 1 || days > 31) {
                goto derror;
        }
        if (leap_year (year)) {
                if (days > leap_month_days[month]) {
                        goto derror;
                }
        } else {
                if (days > normal_month_days[month]) {
                        goto derror;
                }
        }
#ifdef  HAVE_LANGINFO_CODESET
        month--;

        memset ((void *)&tstruct, 0, sizeof(struct tm));
        tstruct.tm_year = year - 1900;
        tstruct.tm_mon = month;
        tstruct.tm_mday = days;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                deflocale = locale_buff;
                localep2 = setlocale (LC_TIME, NULL);
                if (localep2) {
                        localep = strdup (localep2);
                }
                (void) setlocale (LC_TIME, deflocale);
        }
        memset (buff2, 0, sizeof(buff2));
        snprintf(buff2, sizeof(buff2) - 1, "%s", nl_langinfo(D_FMT));
        if (deflocale) {
                if (localep) {
                        (void) setlocale (LC_TIME, localep);
                }
        }
        strftime (buff, sizeof(buff), buff2, &tstruct);
#else
        memset ((void *)&syst, 0, sizeof(syst));
        syst.wYear = year;
        syst.wMonth = month;
        syst.wDay = days;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                for (p = locale_buff; *p; ++p) {
                        if (isalnum(*p) || *p == '_') {
```

```
                                continue;
                        }
                        break;
                }
                *p = 0;
                for (len = 0; len < WINLOCSIZE; ++len) {
                        if (!strcmp(locale_buff, wintable[len].winlocalename)) {
                                localeid = wintable[len].winlocaleid;
                                break;
                        }
                }
                if (len == WINLOCSIZE) {
                        goto derror;
                }
        }
        if (!GetDateFormat (localeid, DATE_SHORTDATE, &syst, NULL, buff, sizeof(b
      uff))) {
                goto derror;
        }
#endif
        len = strlen (buff);
        field.size = len;
        make_field_entry (&field);
        memcpy (curr_field->data, buff, len);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
derror:
#endif
        field.size = 10;
        make_field_entry (&field);
        memset (curr_field->data, ' ', 10);
        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
        return curr_field;
}
```

7.37.1.31 cob_field* cob_intr_locale_time (const int , const int , cob_field * , cob_field *)

Definition at line 3012 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;
#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        size_t          len;
        int             indate;
        int             hours;
        int             minutes;
        int             seconds;
#ifdef  HAVE_LANGINFO_CODESET
        unsigned char   *p;
        char            *deflocale = NULL;
        char            *localep = NULL;
        char            *localep2;
        struct tm       tstruct;
        char            buff2[128];
#else
```

```
        char            *p;
        LCID            localeid = LOCALE_USER_DEFAULT;
        SYSTEMTIME      syst;
#endif
        char            buff[128];
#endif

        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        cob_exception_code = 0;

#if defined(_WIN32) || defined(__CYGWIN__) || defined(HAVE_LANGINFO_CODESET)
        if (COB_FIELD_IS_NUMERIC (srcfield)) {
                indate = cob_get_int (srcfield);
        } else {
                if (srcfield->size < 6) {
                        goto derror;
                }
                p = srcfield->data;
                indate = 0;
                for (len = 0; len < 6; ++len, ++p) {
                        if (isdigit (*p)) {
                                indate *= 10;
                                indate += (*p - '0');
                        } else {
                                goto derror;
                        }
                }
        }
        hours = indate / 10000;
        if (hours < 0 || hours > 24) {
                goto derror;
        }
        indate %= 10000;
        minutes = indate / 100;
        if (minutes < 0 || minutes > 59) {
                goto derror;
        }
        seconds = indate % 100;
        if (seconds < 0 || seconds > 59) {
                goto derror;
        }

#ifdef  HAVE_LANGINFO_CODESET
        memset ((void *)&tstruct, 0, sizeof(struct tm));
        tstruct.tm_hour = hours;
        tstruct.tm_min = minutes;
        tstruct.tm_sec = seconds;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                deflocale = locale_buff;
                localep2 = setlocale (LC_TIME, NULL);
                if (localep2) {
                        localep = strdup (localep2);
                }
                (void) setlocale (LC_TIME, deflocale);
        }
        memset (buff2, 0, sizeof(buff2));
        snprintf(buff2, sizeof(buff2) - 1, "%s", nl_langinfo(T_FMT));
```

```
        if (deflocale) {
                if (localep) {
                        (void) setlocale (LC_TIME, localep);
                }
        }
        strftime (buff, sizeof(buff), buff2, &tstruct);
#else
        memset ((void *)&syst, 0, sizeof(syst));
        syst.wHour = hours;
        syst.wMinute = minutes;
        syst.wSecond = seconds;
        if (locale_field) {
                if (locale_field->size >= COB_SMALL_BUFF) {
                        goto derror;
                }
                cob_field_to_string (locale_field, locale_buff);
                for (p = locale_buff; *p; ++p) {
                        if (isalnum(*p) || *p == '_') {
                                continue;
                        }
                        break;
                }
                *p = 0;
                for (len = 0; len < WINLOCSIZE; ++len) {
                        if (!strcmp(locale_buff, wintable[len].winlocalename)) {
                                localeid = wintable[len].winlocaleid;
                                break;
                        }
                }
                if (len == WINLOCSIZE) {
                        goto derror;
                }
        }
        if (!GetTimeFormat (localeid, LOCALE_NOUSEROVERRIDE, &syst, NULL, buff, s
      izeof(buff))) {

                goto derror;
        }
#endif
        len = strlen (buff);
        field.size = len;
        make_field_entry (&field);
        memcpy (curr_field->data, buff, len);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
derror:
#endif
        field.size = 10;
        make_field_entry (&field);
        memset (curr_field->data, ' ', 10);
        cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
        return curr_field;
}
```

7.37.1.32 cob_field* cob_intr_log (cob_field *)

Definition at line 1690 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = log (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.37.1.33 cob_field* cob_intr_log10 (cob_field *)

Definition at line 1708 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = log10 (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.37.1.34 cob_field* cob_intr_lower_case (const int , const int , cob_field *)

Definition at line 605 of file intrinsic.c.

```
{
        size_t          i, size;

        make_field_entry (srcfield);

        size = srcfield->size;
        for (i = 0; i < size; ++i) {
                curr_field->data[i] = tolower (srcfield->data[i]);
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.37.1.35 cob_field* cob_intr_max (const int , ...)

Definition at line 2125 of file intrinsic.c.

```
{
        cob_field       *f, *basef;
        va_list         args;
        int             i;

        va_start (args, params);

        basef = va_arg (args, cob_field *);
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basef) > 0) {
                        basef = f;
                }
        }
        va_end (args);

        return basef;
}
```

7.37.1.36 cob_field* cob_intr_mean (const int , ...)

Definition at line 2224 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        long long       n;
        union {
                unsigned char   data[8];
                long long       datall;
        } datun;
        int             i;
        cob_field_attr  attr;
        cob_field       field;


        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        mpz_set_ui (d1.value, 0);
        d1.scale = 0;

        va_start (args, params);
        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                cob_decimal_add (&d1, &d2);
        }
        va_end (args);

        mpz_set_ui (d2.value, (unsigned int)params);
        d2.scale = 0;
        cob_decimal_div (&d1, &d2);
        field.data = datun.data;
```

```
        cob_decimal_get_field (&d1, &field, 0);
        n = datun.datall;
        for (i = 0; n; n /= 10, ++i) ;
        field.data = NULL;
        if (i <= 18) {
                attr.scale = 18 - i;
        }
        make_field_entry (&field);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.37 cob_field* cob_intr_median (const int , ...)

Definition at line 2179 of file intrinsic.c.

```
{
        cob_field       *f;
        cob_field       **field_alloc;
        va_list         args;
        int             i;

        va_start (args, params);

        f = va_arg (args, cob_field *);
        if (params == 1) {
                va_end (args);
                return f;
        }

        field_alloc = cob_malloc (params * sizeof (cob_field *));
        field_alloc[0] = f;

        for (i = 1; i < params; ++i) {
                field_alloc[i] = va_arg (args, cob_field *);
        }
        va_end (args);

        qsort (field_alloc, (size_t)params, (size_t)sizeof (cob_field *), comp_fi
      eld);

        i = params / 2;
        if (params % 2) {
                f = field_alloc[i];
        } else {
                make_double_entry ();
                cob_decimal_set_field (&d1, field_alloc[i-1]);
                cob_decimal_set_field (&d2, field_alloc[i]);
                cob_decimal_add (&d1, &d2);
                mpz_set_ui (d2.value, 2);
                d2.scale = 0;
                cob_decimal_div (&d1, &d2);
                cob_decimal_get_field (&d1, curr_field, 0);
                f = curr_field;
        }

        free (field_alloc);
        return f;
}
```

7.37.1.38 cob_field* cob_intr_midrange (const int , ...)

Definition at line 2146 of file intrinsic.c.

```
{
        cob_field       *f, *basemin, *basemax;
        va_list         args;
        int             i;

        make_double_entry ();
        va_start (args, params);

        basemin = va_arg (args, cob_field *);
        basemax = basemin;
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basemin) < 0) {
                        basemin = f;
                }
                if (cob_cmp (f, basemax) > 0) {
                        basemax = f;
                }
        }
        va_end (args);

        cob_decimal_set_field (&d1, basemin);
        cob_decimal_set_field (&d2, basemax);
        cob_decimal_add (&d1, &d2);
        mpz_set_ui (d2.value, 2);
        d2.scale = 0;
        cob_decimal_div (&d1, &d2);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.39 cob_field* cob_intr_min (const int , ...)

Definition at line 2104 of file intrinsic.c.

```
{
        cob_field       *f, *basef;
        va_list         args;
        int             i;

        va_start (args, params);

        basef = va_arg (args, cob_field *);
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basef) < 0) {
                        basef = f;
                }
        }
        va_end (args);

        return basef;
}
```

7.37.1.40 cob_field* cob_intr_mod (cob_field *, cob_field *)

Definition at line 2268 of file intrinsic.c.

```
{
        cob_field       *f1;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        make_field_entry (&field);

        f1 = cob_intr_integer (cob_intr_binop (srcfield1, '/', srcfield2));
        cob_decimal_set_field (&d1, srcfield2);
        cob_decimal_set_field (&d2, f1);
        cob_decimal_mul (&d2, &d1);
        cob_decimal_set_field (&d1, srcfield1);
        cob_decimal_sub (&d1, &d2);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.41 cob_field* cob_intr_numval (cob_field *)

Definition at line 1795 of file intrinsic.c.

```
{
        long long       llval = 0;
        double          val;
        size_t          i;
        int             integer_digits = 0;
        int             decimal_digits = 0;
        int             sign = 0;
        int             decimal_seen = 0;
        cob_field_attr  attr;
        cob_field       field;
        unsigned char   integer_buff[64];
        unsigned char   decimal_buff[64];
        unsigned char   final_buff[64];

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        memset (integer_buff, 0, sizeof (integer_buff));
        memset (decimal_buff, 0, sizeof (decimal_buff));
        memset (final_buff, 0, sizeof (final_buff));

        for (i = 0; i < srcfield->size; ++i) {
                if (i < (srcfield->size - 2)) {
                        if (strcasecmp ((char *)&srcfield->data[i], "CR") == 0
                             || strcasecmp ((char *)&srcfield->data[i], "DB") ==
      0) {
                                sign = 1;
                                break;
                        }
                }
```

```
                if (srcfield->data[i] == ' ') {
                        continue;
                }
                if (srcfield->data[i] == '+') {
                        continue;
                }
                if (srcfield->data[i] == '-') {
                        sign = 1;
                        continue;
                }
                if (srcfield->data[i] == cob_current_module->decimal_point) {
                        decimal_seen = 1;
                        continue;
                }
                if (srcfield->data[i] >= '0' && srcfield->data[i] <= '9') {
                        llval *= 10;
                        llval += srcfield->data[i] - '0';
                        if (decimal_seen) {
                                decimal_buff[decimal_digits++] = srcfield->data[i
      ];
                        } else {
                                integer_buff[integer_digits++] = srcfield->data[i
      ];
                        }
                }
                if ((integer_digits + decimal_digits) > 30) {
                        break;
                }
        }
        if (!integer_digits) {
                integer_buff[0] = '0';
        }
        if (!decimal_digits) {
                decimal_buff[0] = '0';
        }
        if (sign) {
                llval = -llval;
        }
        if ((integer_digits + decimal_digits) <= 18) {
                attr.scale = decimal_digits;
                make_field_entry (&field);
                memcpy (curr_field->data, (char *)&llval, 8);
        } else {
                snprintf ((char *)final_buff, 63, "%s%s.%s", sign ? "-" : "",
                         integer_buff, decimal_buff);
                sscanf ((char *)final_buff, "%lf", &val);
                make_double_entry ();
                memcpy (curr_field->data, (char *)&val, sizeof (double));
        }
        return curr_field;
}
```

7.37.1.42 cob_field* cob_intr_numval_c (cob_field *, cob_field *)

Definition at line 1875 of file intrinsic.c.

```
{
        unsigned char   *currency_data;
        long long       llval = 0;
```

```
  double          val;
  size_t          i;
  int             integer_digits = 0;
  int             decimal_digits = 0;
  int             sign = 0;
  int             decimal_seen = 0;
  cob_field_attr  attr;
  cob_field       field;
  unsigned char   integer_buff[64];
  unsigned char   decimal_buff[64];
  unsigned char   final_buff[64];

  COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

  COB_FIELD_INIT (8, NULL, &attr);
  memset (integer_buff, 0, sizeof (integer_buff));
  memset (decimal_buff, 0, sizeof (decimal_buff));
  memset (final_buff, 0, sizeof (final_buff));

  currency_data = NULL;
  if (currency) {
          if (currency->size < srcfield->size) {
                  currency_data = currency->data;
          }
  }
  for (i = 0; i < srcfield->size; ++i) {
          if (i < (srcfield->size - 2)) {
                  if (strcasecmp ((char *)&srcfield->data[i], "CR") == 0
                       || strcasecmp ((char *)&srcfield->data[i], "DB") ==
0) {
                          sign = 1;
                          break;
                  }
          }
          if (currency_data) {
                  if (i < (srcfield->size - currency->size)) {
                          if (memcmp ((char *)&srcfield->data[i], currency_
data,
                                      currency->size) == 0) {
                                  i += (currency->size - 1);
                                  continue;
                          }
                  }
          }
          if (srcfield->data[i] == ' ') {
                  continue;
          }
          if (srcfield->data[i] == '+') {
                  continue;
          }
          if (srcfield->data[i] == '-') {
                  sign = 1;
                  continue;
          }
          if (srcfield->data[i] == cob_current_module->decimal_point) {
                  decimal_seen = 1;
                  continue;
          }
          if (srcfield->data[i] == cob_current_module->currency_symbol) {
                  continue;
          }
          if (srcfield->data[i] >= '0' && srcfield->data[i] <= '9') {
```

```
                        llval *= 10;
                        llval += srcfield->data[i] - '0';
                        if (decimal_seen) {
                                decimal_buff[decimal_digits++] = srcfield->data[i
      ];
                        } else {
                                integer_buff[integer_digits++] = srcfield->data[i
      ];
                        }
                }
                if ((integer_digits + decimal_digits) > 30) {
                        break;
                }
        }
        if (!integer_digits) {
                integer_buff[0] = '0';
        }
        if (!decimal_digits) {
                decimal_buff[0] = '0';
        }
        if (sign) {
                llval = -llval;
        }
        if ((integer_digits + decimal_digits) <= 18) {
                attr.scale = decimal_digits;
                make_field_entry (&field);
                memcpy (curr_field->data, (char *)&llval, 8);
        } else {
                snprintf ((char *)final_buff, 63, "%s%s.%s", sign ? "-" : "",
                          integer_buff, decimal_buff);
                sscanf ((char *)final_buff, "%lf", &val);
                make_double_entry ();
                memcpy (curr_field->data, (char *)&val, sizeof (double));
        }
        return curr_field;
}
```

7.37.1.43 cob_field* cob_intr_ord (cob_field *)

Definition at line 1141 of file intrinsic.c.

```
{
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_set_int (curr_field, (int)(*srcfield->data + 1));
        return curr_field;
}
```

7.37.1.44 cob_field* cob_intr_ord_max (const int , ...)

Definition at line 2069 of file intrinsic.c.

```
{
        cob_field       *f, *basef;
        int             ordmin = 0;
        int             i;
        va_list         args;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        if (params <= 1) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        va_start (args, params);

        basef = va_arg (args, cob_field *);
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basef) > 0) {
                        basef = f;
                        ordmin = i;
                }
        }
        va_end (args);

        cob_set_int (curr_field, ordmin + 1);
        return curr_field;
}
```

7.37.1.45 cob_field* cob_intr_ord_min (const int , ...)

Definition at line 2034 of file intrinsic.c.

```
{
        cob_field       *f, *basef;
        int             i;
        int             ordmin = 0;
        va_list         args;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        if (params <= 1) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        va_start (args, params);

        basef = va_arg (args, cob_field *);
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
```

```
                if (cob_cmp (f, basef) < 0) {
                        basef = f;
                        ordmin = i;
                }
        }
        va_end (args);

        cob_set_int (curr_field, ordmin + 1);
        return curr_field;
}
```

7.37.1.46 cob_field* cob_intr_present_value (const int , ...)

Definition at line 2538 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        int             i;

        va_start (args, params);
        make_double_entry ();

        if (params < 2) {
                va_end (args);
                fprintf (stderr, "Wrong number of parameters for FUNCTION PRESENT
      -VALUE\n");
                fflush (stderr);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        f = va_arg (args, cob_field *);
        cob_decimal_set_field (&d1, f);
        mpz_set_ui (d2.value, 1);
        d2.scale = 0;
        cob_decimal_add (&d1, &d2);

        mpz_set_ui (d4.value, 0);
        d4.scale = 0;

        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                mpz_set (d3.value, d1.value);
                d3.scale = d1.scale;
                if (i > 1) {
                        mpz_set_ui (d5.value, (unsigned int)i);
                        d5.scale = 0;
                        cob_decimal_pow (&d3, &d5);
                }
                cob_decimal_div (&d2, &d3);
                cob_decimal_add (&d4, &d2);
        }
        va_end (args);

        cob_decimal_get_field (&d4, curr_field, 0);
        return curr_field;
}
```

7.37.1.47 cob_field* cob_intr_random (const int , ...)

Definition at line 2352 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        int             seed = 1;
        int             randnum;
        int             i;
        int             exp10;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 9, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        va_start (args, params);

        if (params) {
                f = va_arg (args, cob_field *);
                seed = cob_get_int (f);
                if (seed < 0) {
                        seed = 0;
                }
#ifdef  __CYGWIN__
                srandom ((unsigned int)seed);
#else
                srand ((unsigned int)seed);
#endif
        }
        va_end (args);

#ifdef  __CYGWIN__
        randnum = (int)random ();
#else
        randnum = rand ();
#endif
        exp10 = 1;
        for (i = 0; i < 10; ++i) {
                if ((randnum / exp10) == 0) {
                        break;
                }
                exp10 *= 10;
        }
        if (i == 0) {
                i = 1;
        }
        attr.scale = i;
        make_field_entry (&field);
        *(long long *)curr_field->data = (long long)randnum;
        return curr_field;
}
```

7.37.1.48 cob_field* cob_intr_range (const int , ...)

Definition at line 2289 of file intrinsic.c.

```
{
```

```
        cob_field       *f, *basemin, *basemax;
        va_list         args;
        int             i;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        va_start (args, params);

        basemin = va_arg (args, cob_field *);
        basemax = basemin;
        for (i = 1; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if (cob_cmp (f, basemin) < 0) {
                        basemin = f;
                }
                if (cob_cmp (f, basemax) > 0) {
                        basemax = f;
                }
        }
        va_end (args);

        attr.scale = COB_FIELD_SCALE(basemin);
        if (COB_FIELD_SCALE(basemax) > attr.scale) {
                attr.scale = COB_FIELD_SCALE(basemax);
        }
        make_field_entry (&field);
        cob_decimal_set_field (&d1, basemax);
        cob_decimal_set_field (&d2, basemin);
        cob_decimal_sub (&d1, &d2);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.49 cob_field* cob_intr_rem (cob_field *, cob_field *)

Definition at line 2327 of file intrinsic.c.

```
{
        cob_field       *f1;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        f1 = cob_intr_integer_part (cob_intr_binop (srcfield1, '/', srcfield2));
        cob_decimal_set_field (&d1, srcfield2);
        cob_decimal_set_field (&d2, f1);
        cob_decimal_mul (&d2, &d1);
        cob_decimal_set_field (&d1, srcfield1);
        cob_decimal_sub (&d1, &d2);

        attr.scale = COB_FIELD_SCALE(srcfield1);
        if (COB_FIELD_SCALE(srcfield2) > attr.scale) {
                attr.scale = COB_FIELD_SCALE(srcfield2);
        }
```

```
        make_field_entry (&field);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.50 cob_field* cob_intr_reverse (const int , const int , cob_field *)

Definition at line 622 of file intrinsic.c.

```
{
        size_t          i, size;

        make_field_entry (srcfield);

        size = srcfield->size;
        for (i = 0; i < size; ++i) {
                curr_field->data[i] = srcfield->data[srcfield->size - i - 1];
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.37.1.51 cob_field* cob_intr_seconds_from_formatted_time (cob_field * , cob_field *)

Definition at line 2803 of file intrinsic.c.

```
{
        unsigned char   *p1;
        unsigned char   *p2;
        size_t          n;
        int             seconds = 0;
        int             minutes = 0;
        int             hours = 0;
        int             seconds_seen = 0;
        int             minutes_seen = 0;
        int             hours_seen = 0;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        if (value->size < format->size) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        p1 = format->data;
        p2 = value->data;
        for (n = 0; n < format->size - 1; ++n, ++p1, ++p2) {
                if (!memcmp (p1, "hh", 2) && !hours_seen) {
```

```
                        if (*p2 >= '0' && *p2 <= '9' &&
                            *(p2 + 1) >= '0' && *(p2 + 1) <= '9') {
                                hours = ((*p2 - '0') * 10) + (*(p2 + 1) - '0');
                                hours_seen = 1;
                                continue;
                        }
                }
                if (!memcmp (p1, "mm", 2) && !minutes_seen) {
                        if (*p2 >= '0' && *p2 <= '9' &&
                            *(p2 + 1) >= '0' && *(p2 + 1) <= '9') {
                                minutes = ((*p2 - '0') * 10) + (*(p2 + 1) - '0');

                                minutes_seen = 1;
                                continue;
                        }
                }
                if (!memcmp (p1, "ss", 2) && !seconds_seen) {
                        if (*p2 >= '0' && *p2 <= '9' &&
                            *(p2 + 1) >= '0' && *(p2 + 1) <= '9') {
                                seconds = ((*p2 - '0') * 10) + (*(p2 + 1) - '0');

                                seconds_seen = 1;
                                continue;
                        }
                }
        }
        if (hours_seen && minutes_seen && seconds_seen) {
                seconds += (hours * 3600) + (minutes * 60);
        } else {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                seconds = 0;
        }
        cob_set_int (curr_field, seconds);
        return curr_field;
}
```

7.37.1.52 cob_field* cob_intr_seconds_past_midnight (void)

Definition at line 2782 of file intrinsic.c.

```
{
        struct tm       *timeptr;
        time_t          t;
        int             seconds;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        t = time (NULL);
        timeptr = localtime (&t);
        seconds = (timeptr->tm_hour * 3600) + (timeptr->tm_min * 60) +
                        timeptr->tm_sec;
        cob_set_int (curr_field, seconds);
        return curr_field;
}
```

7.37.1.53 cob_field* cob_intr_sign (cob_field *)

Definition at line 566 of file intrinsic.c.

```
{
        int             n;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, COB_FLAG_HAVE_SIGN, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_set_int (curr_field, 0);
        n = cob_cmp (srcfield, curr_field);
        if (n < 0) {
                cob_set_int (curr_field, -1);
        } else if (n > 0) {
                cob_set_int (curr_field, 1);
        }

        return curr_field;
}
```

7.37.1.54 cob_field* cob_intr_sin (cob_field *)

Definition at line 1726 of file intrinsic.c.

```
{
        long long               result;
        double                  mathd2;
        int                     i, tempres;
        cob_field_attr          attr;
        cob_field               field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 17, COB_FLAG_HAVE_SIGN, NULL)
      ;
        COB_FIELD_INIT (8, NULL, &attr);
        cob_decimal_set_field (&d1, srcfield);
        make_field_entry (&field);

        errno = 0;
        mathd2 = sin (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        result = (long long) mathd2;
        mathd2 -= result;
        for (i = 0; i < 17; ++i) {
                mathd2 *= 10;
                tempres = (int) mathd2;
                result *= 10;
                result += tempres;
                mathd2 -= tempres;
        }
        memcpy (curr_field->data, (char *)&result, 8);
        return curr_field;
}
```

7.37.1.55 cob_field* cob_intr_sqrt (cob_field *)

Definition at line 1759 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = sqrt (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.37.1.56 cob_field* cob_intr_standard_deviation (const int , ...)

Definition at line 2472 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        int             i;

        va_start (args, params);
        make_double_entry ();

        if (params == 1) {
                va_end (args);
                cob_set_int (curr_field, 0);
                return curr_field;
        }

        /* MEAN for all params */
        mpz_set_ui (d1.value, 0);
        d1.scale = 0;

        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                cob_decimal_add (&d1, &d2);
        }
        va_end (args);
        mpz_set_ui (d2.value, (unsigned int)params);
        d2.scale = 0;
        cob_decimal_div (&d1, &d2);

        /* Got the MEAN in d1, iterate again */

        mpz_set_ui (d4.value, 0);
        d4.scale = 0;

        va_start (args, params);
```

```
        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                cob_decimal_set_field (&d2, f);
                cob_decimal_sub (&d2, &d1);
                cob_decimal_mul (&d2, &d2);
                cob_decimal_add (&d4, &d2);
        }
        va_end (args);

        mpz_set_ui (d3.value, (unsigned int)params);
        d3.scale = 0;
        cob_decimal_div (&d4, &d3);
        /* We have the VARIANCE in d4, sqrt = STANDARD-DEVIATION */

/* Do not know why this does not work
        d5.scale = d4.scale;
        mpz_mul_ui (d5.value, d4.value, 1000000000);
        mpz_mul_ui (d4.value, d5.value, 1000000000);
        mpz_sqrt (d5.value, d4.value);
        mpz_div_ui (d4.value, d5.value, 1000000000);
        cob_decimal_get_field (&d4, curr_field, 0);
        return curr_field;
*/

        cob_decimal_get_field (&d4, curr_field, 0);
        f = cob_intr_sqrt (curr_field);
        return f;
}
```

7.37.1.57 cob_field* cob_intr_stored_char_length (cob_field *)

Definition at line 1155 of file intrinsic.c.

```
{
        unsigned char   *p;
        int             count;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        count = srcfield->size;
        p = srcfield->data + srcfield->size - 1;
        for (; count > 0; count--, p--) {
                if (*p != ' ') {
                        break;
                }
        }
        cob_set_int (curr_field, count);
        return curr_field;
}
```

7.37.1.58 cob_field* cob_intr_substitute (const int , const int , const int , ...)

Definition at line 678 of file intrinsic.c.

```
{
        cob_field       *var;
        cob_field       **f1;
        cob_field       **f2;
        unsigned char   *p1;
        unsigned char   *p2;
        size_t          varsize;
        size_t          calcsize;
        size_t          n;
        size_t          found;
        int             numreps;
        int             i;
        cob_field_attr  attr;
        cob_field       field;
        va_list         args;

        numreps = params / 2;
        f1 = cob_malloc (numreps * sizeof (cob_field *));
        f2 = cob_malloc (numreps * sizeof (cob_field *));

        va_start (args, params);

        var = va_arg (args, cob_field *);
        varsize = var->size;

        /* Extract args */
        for (i = 0; i < params - 1; ++i) {
                if ((i % 2) == 0) {
                        f1[i / 2] = va_arg (args, cob_field *);
                } else {
                        f2[i / 2] = va_arg (args, cob_field *);
                }
        }

        /* Calculate required size */
        calcsize = 0;
        found = 0;
        p1 = var->data;
        for (n = 0; n < varsize; ) {
                for (i = 0; i < numreps; ++i) {
                        if (n + f1[i]->size <= varsize) {
                                if (!memcmp (p1, f1[i]->data, f1[i]->size)) {
                                        p1 += f1[i]->size;
                                        n += f1[i]->size;
                                        calcsize += f2[i]->size;
                                        found = 1;
                                        break;
                                }
                        }
                }
                if (found) {
                        found = 0;
                        continue;
                }
                ++n;
                ++p1;
                ++calcsize;
        }
```

```
        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        COB_FIELD_INIT (0, NULL, &attr);
        field.size = calcsize;
        make_field_entry (&field);

        found = 0;
        p1 = var->data;
        p2 = curr_field->data;
        for (n = 0; n < varsize; ) {
                for (i = 0; i < numreps; ++i) {
                        if (n + f1[i]->size <= varsize) {
                                if (!memcmp (p1, f1[i]->data, f1[i]->size)) {
                                        memcpy (p2, f2[i]->data, f2[i]->size);
                                        p1 += f1[i]->size;
                                        p2 += f2[i]->size;
                                        n += f1[i]->size;
                                        found = 1;
                                        break;
                                }
                        }
                }
                if (found) {
                        found = 0;
                        continue;
                }
                ++n;
                *p2++ = *p1++;
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        free (f1);
        free (f2);
        return curr_field;
}
```

7.37.1.59 cob_field* cob_intr_substitute_case (const int , const int , const int , ...)

Definition at line 775 of file intrinsic.c.

```
{
        cob_field       *var;
        cob_field       **f1;
        cob_field       **f2;
        unsigned char   *p1;
        unsigned char   *p2;
        size_t          varsize;
        size_t          calcsize;
        size_t          n;
        size_t          found;
        int             numreps;
        int             i;
        cob_field_attr  attr;
        cob_field       field;
        va_list         args;

        numreps = params / 2;
        f1 = cob_malloc (numreps * sizeof (cob_field *));
```

```
f2 = cob_malloc (numreps * sizeof (cob_field *));

va_start (args, params);

var = va_arg (args, cob_field *);
varsize = var->size;

/* Extract args */
for (i = 0; i < params - 1; ++i) {
        if ((i % 2) == 0) {
                f1[i / 2] = va_arg (args, cob_field *);
        } else {
                f2[i / 2] = va_arg (args, cob_field *);
        }
}

/* Calculate required size */
calcsize = 0;
found = 0;
p1 = var->data;
for (n = 0; n < varsize; ) {
        for (i = 0; i < numreps; ++i) {
                if (n + f1[i]->size <= varsize) {
                        if (!strncasecmp ((const char *)p1,
                                          (const char *)(f1[i]->data),
                                          f1[i]->size)) {
                                p1 += f1[i]->size;
                                n += f1[i]->size;
                                calcsize += f2[i]->size;
                                found = 1;
                                break;
                        }
                }
        }
        if (found) {
                found = 0;
                continue;
        }
        ++n;
        ++p1;
        ++calcsize;
}

COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
COB_FIELD_INIT (0, NULL, &attr);
field.size = calcsize;
make_field_entry (&field);

found = 0;
p1 = var->data;
p2 = curr_field->data;
for (n = 0; n < varsize; ) {
        for (i = 0; i < numreps; ++i) {
                if (n + f1[i]->size <= varsize) {
                        if (!strncasecmp ((const char *)p1,
                                          (const char *)(f1[i]->data),
                                          f1[i]->size)) {
                                memcpy (p2, f2[i]->data, f2[i]->size);
                                p1 += f1[i]->size;
                                p2 += f2[i]->size;
                                n += f1[i]->size;
                                found = 1;
```

```
                                        break;
                                }
                        }
                }
                if (found) {
                        found = 0;
                        continue;
                }
                ++n;
                *p2++ = *p1++;
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        free (f1);
        free (f2);
        return curr_field;
}
```

7.37.1.60 cob_field* cob_intr_sum (const int , ...)

Definition at line 1996 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        int             i;
        int             digits = 0;
        int             scale = 0;
        cob_field_attr  attr;
        cob_field       field;


        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

        COB_FIELD_INIT (8, NULL, &attr);
        mpz_set_ui (d1.value, 0);
        d1.scale = 0;

        va_start (args, params);

        for (i = 0; i < params; ++i) {
                f = va_arg (args, cob_field *);
                if ((COB_FIELD_DIGITS(f) - COB_FIELD_SCALE(f)) > digits) {
                        digits = COB_FIELD_DIGITS(f) - COB_FIELD_SCALE(f);
                }
                if (COB_FIELD_SCALE(f) > scale) {
                        scale = COB_FIELD_SCALE(f);
                }
                cob_decimal_set_field (&d2, f);
                cob_decimal_add (&d1, &d2);
        }
        va_end (args);

        attr.scale = scale;
        make_field_entry (&field);
        cob_decimal_get_field (&d1, curr_field, 0);
        return curr_field;
}
```

7.37.1.61 cob_field* cob_intr_tan (cob_field *)

Definition at line 1777 of file intrinsic.c.

```
{
        double          mathd2;

        cob_decimal_set_field (&d1, srcfield);
        make_double_entry ();

        errno = 0;
        mathd2 = tan (intr_get_double (&d1));
        if (errno) {
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        memcpy (curr_field->data, (char *)&mathd2, 8);
        return curr_field;
}
```

7.37.1.62 cob_field* cob_intr_test_date_yyyymmdd (cob_field *)

Definition at line 1409 of file intrinsic.c.

```
{
        int             indate;
        int             days;
        int             month;
        int             year;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        /* Base 1601-01-01 */
        indate = cob_get_int (srcfield);
        year = indate / 10000;
        if (year < 1601 || year > 9999) {
                cob_set_int (curr_field, 1);
                return curr_field;
        }
        indate %= 10000;
        month = indate / 100;
        if (month < 1 || month > 12) {
                cob_set_int (curr_field, 2);
                return curr_field;
        }
        days = indate % 100;
        if (days < 1 || days > 31) {
                cob_set_int (curr_field, 3);
                return curr_field;
        }
        if (leap_year (year)) {
                if (days > leap_month_days[month]) {
                        cob_set_int (curr_field, 3);
                        return curr_field;
```

```
                }
        } else {
                if (days > normal_month_days[month]) {
                        cob_set_int (curr_field, 3);
                        return curr_field;
                }
        }
        cob_set_int (curr_field, 0);
        return curr_field;
}
```

7.37.1.63 cob_field* cob_intr_test_day_yyyyddd (cob_field *)

Definition at line 1456 of file intrinsic.c.

```
{
        int             indate;
        int             days;
        int             year;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        /* Base 1601-01-01 */
        indate = cob_get_int (srcfield);
        year = indate / 1000;
        if (year < 1601 || year > 9999) {
                cob_set_int (curr_field, 1);
                return curr_field;
        }
        days = indate % 1000;
        if (days < 1 || days > 365 + leap_year (year)) {
                cob_set_int (curr_field, 2);
                return curr_field;
        }
        cob_set_int (curr_field, 0);
        return curr_field;
}
```

7.37.1.64 cob_field* cob_intr_trim (const int , const int , cob_field * , const int)

Definition at line 876 of file intrinsic.c.

```
{
        unsigned char   *begin;
        unsigned char   *end;
        size_t          i;
        size_t          size = 0;

        make_field_entry (srcfield);

        for (i = 0; i < srcfield->size; ++i) {
```

```
                if (srcfield->data[i] != ' ') {
                        break;
                }
        }
        if (i == srcfield->size) {
                curr_field->size = 1;
                curr_field->data[0] = ' ';
                return curr_field;
        }
        begin = srcfield->data;
        if (direction != 2) {
                for (; *begin == ' '; ++begin) ;
        }
        end = srcfield->data + srcfield->size - 1;
        if (direction != 1) {
                for (; *end == ' '; end--) ;
        }
        for (i = 0; begin <= end; ++begin, ++i) {
                curr_field->data[i] = *begin;
                ++size;
        }
        curr_field->size = size;
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.37.1.65 cob_field* cob_intr_upper_case (const int , const int , cob_field *)

Definition at line 588 of file intrinsic.c.

```
{
        size_t          i, size;

        make_field_entry (srcfield);

        size = srcfield->size;
        for (i = 0; i < size; ++i) {
                curr_field->data[i] = toupper (srcfield->data[i]);
        }
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.37.1.66 cob_field* cob_intr_variance (const int , ...)

Definition at line 2403 of file intrinsic.c.

```
{
        cob_field       *f;
        va_list         args;
        long long       n;
```

```
union {
        unsigned char   data[8];
        long long       datall;
} datun;
int             i;
cob_field_attr  attr;
cob_field       field;

COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0, COB_FLAG_HAVE_SIGN, NULL);

COB_FIELD_INIT (8, NULL, &attr);
if (params == 1) {
        make_field_entry (&field);
        cob_set_int (curr_field, 0);
        return curr_field;
}

/* MEAN for all params */
mpz_set_ui (d1.value, 0);
d1.scale = 0;

va_start (args, params);
for (i = 0; i < params; ++i) {
        f = va_arg (args, cob_field *);
        cob_decimal_set_field (&d2, f);
        cob_decimal_add (&d1, &d2);
}
va_end (args);
mpz_set_ui (d2.value, (unsigned int)params);
d2.scale = 0;
cob_decimal_div (&d1, &d2);

/* Got the MEAN in d1, iterate again */

mpz_set_ui (d4.value, 0);
d4.scale = 0;

va_start (args, params);

for (i = 0; i < params; ++i) {
        f = va_arg (args, cob_field *);
        cob_decimal_set_field (&d2, f);
        cob_decimal_sub (&d2, &d1);
        cob_decimal_mul (&d2, &d2);
        cob_decimal_add (&d4, &d2);
}
va_end (args);

mpz_set_ui (d3.value, (unsigned int)params);
d3.scale = 0;
cob_decimal_div (&d4, &d3);
field.data = datun.data;
cob_decimal_get_field (&d4, &field, 0);
n = datun.datall;
for (i = 0; n; n /= 10, ++i) ;
field.data = NULL;
if (i <= 18) {
        attr.scale = 18 - i;
}
make_field_entry (&field);
cob_decimal_get_field (&d4, curr_field, 0);
return curr_field;
```

```
}
```

7.37.1.67 cob_field* cob_intr_when_compiled (const int , const int , cob_field *)

Definition at line 1020 of file intrinsic.c.

```
{
        make_field_entry (f);

        memcpy (curr_field->data, f->data, f->size);
        if (unlikely(offset > 0)) {
                calc_ref_mod (curr_field, offset, length);
        }
        return curr_field;
}
```

7.37.1.68 cob_field* cob_intr_year_to_yyyy (const int , ...)

Definition at line 2583 of file intrinsic.c.

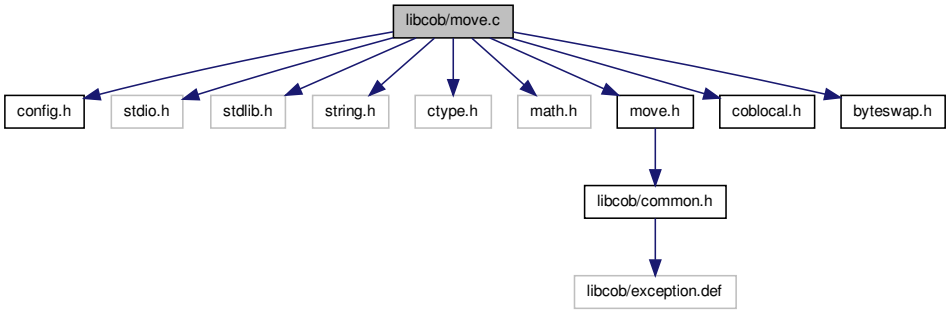
```
{
        cob_field       *f;
        struct tm       *timeptr;
        va_list         args;
        time_t          t;
        int             year;
        int             interval;
        int             xqtyear;
        int             maxyear;
        cob_field_attr  attr;
        cob_field       field;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 8, 0, 0, NULL);
        COB_FIELD_INIT (4, NULL, &attr);
        make_field_entry (&field);

        cob_exception_code = 0;
        va_start (args, params);
        f = va_arg (args, cob_field *);
        year = cob_get_int (f);
        if (params > 1) {
                f = va_arg (args, cob_field *);
                interval = cob_get_int (f);
        } else {
                interval = 50;
        }
        if (params > 2) {
                f = va_arg (args, cob_field *);
                xqtyear = cob_get_int (f);
        } else {
                t = time (NULL);
                timeptr = localtime (&t);
                xqtyear = 1900 + timeptr->tm_year;
        }
        va_end (args);
```

```
        if (year < 0 || year > 99) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (xqtyear < 1601 || xqtyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        maxyear = xqtyear + interval;
        if (maxyear < 1700 || maxyear > 9999) {
                cob_set_exception (COB_EC_ARGUMENT_FUNCTION);
                cob_set_int (curr_field, 0);
                return curr_field;
        }
        if (maxyear % 100 >= year) {
                year += 100 * (maxyear / 100);
        } else {
                year += 100 * ((maxyear / 100) - 1);
        }
        cob_set_int (curr_field, year);
        return curr_field;
}
```

7.38 libcob/move.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "move.h"
#include "coblocal.h"
#include "byteswap.h"
```

Include dependency graph for move.c:



Functions

- void cob_move (cob_field *src, cob_field *dst)
- void cob_set_int (cob_field *f, int n)
- int cob_get_int (cob_field *f)
- long long cob_get_long_long (cob_field *f)
- void cob_init_move (void)

7.38.1 Function Documentation

7.38.1.1 int cob_get_int (cob_field * f)

Definition at line 1304 of file move.c.

```
{
        int             n;
        cob_field       temp;
        cob_field_attr  attr;

        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                return cob_display_get_int (f);
        case COB_TYPE_NUMERIC_BINARY:
                return (int)cob_binary_mget_int64 (f);
        case COB_TYPE_NUMERIC_PACKED:
                return cob_packed_get_int (f);
        default:
                COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 9, 0,
                               COB_FLAG_HAVE_SIGN, NULL);
                temp.size = 4;
                temp.data = (unsigned char *)&n;
                temp.attr = &attr;
                cob_move (f, &temp);
                return n;
```

```
        }
}
```

7.38.1.2 long long cob_get_long_long (cob_field * f)

Definition at line 1329 of file move.c.

```
{
        long long       n;
        cob_field       temp;
        cob_field_attr  attr;

        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                return cob_display_get_long_long (f);
        case COB_TYPE_NUMERIC_BINARY:
                return cob_binary_mget_int64 (f);
        case COB_TYPE_NUMERIC_PACKED:
                return cob_packed_get_long_long (f);
        default:
                COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 18, 0,
                               COB_FLAG_HAVE_SIGN, NULL);
                temp.size = 8;
                temp.data = (unsigned char *)&n;
                temp.attr = &attr;
                cob_move (f, &temp);
                return n;
        }
}
```

7.38.1.3 void cob_init_move (void)

Definition at line 1354 of file move.c.

```
{
        lastdata = cob_malloc (COB_SMALL_BUFF);
        lastsize = COB_SMALL_BUFF;
}
```

7.38.1.4 void cob_move (cob_field * src, cob_field * dst)

Definition at line 1023 of file move.c.

```
{
        if (COB_FIELD_TYPE (src) == COB_TYPE_ALPHANUMERIC_ALL) {
                cob_move_all (src, dst);
                return;
        }
        if (dst->size == 0) {
                return;
        }
        if (src->size == 0) {
```

```
        src = &cob_space;
  }

  /* non-elementary move */
  if (COB_FIELD_TYPE (src) == COB_TYPE_GROUP || COB_FIELD_TYPE (dst) ==
COB_TYPE_GROUP) {
        cob_move_alphanum_to_alphanum (src, dst);
        return;
  }

  /* elementary move */
  switch (COB_FIELD_TYPE (src)) {
  case COB_TYPE_NUMERIC_DISPLAY:
        switch (COB_FIELD_TYPE (dst)) {
        case COB_TYPE_NUMERIC_FLOAT:
        case COB_TYPE_NUMERIC_DOUBLE:
                cob_move_display_to_fp (src, dst);
                return;
        case COB_TYPE_NUMERIC_DISPLAY:
                cob_move_display_to_display (src, dst);
                return;
        case COB_TYPE_NUMERIC_PACKED:
                cob_move_display_to_packed (src, dst);
                return;
        case COB_TYPE_NUMERIC_BINARY:
                cob_move_display_to_binary (src, dst);
                return;
        case COB_TYPE_NUMERIC_EDITED:
                cob_move_display_to_edited (src, dst);
                return;
        case COB_TYPE_ALPHANUMERIC_EDITED:
                if (COB_FIELD_SCALE(src) < 0 ||
                    COB_FIELD_SCALE(src) > COB_FIELD_DIGITS(src)) {
                        /* expand P's */
                        indirect_move (cob_move_display_to_display, src,
dst,
                                       (size_t)cob_max_int ((int)
COB_FIELD_DIGITS(src), (int)COB_FIELD_SCALE(src)),
                                       cob_max_int (0, (int)
COB_FIELD_SCALE(src)));
                        return;
                } else {
                        cob_move_alphanum_to_edited (src, dst);
                        return;
                }
        default:
                cob_move_display_to_alphanum (src, dst);
                return;
        }

  case COB_TYPE_NUMERIC_PACKED:
        switch (COB_FIELD_TYPE (dst)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                cob_move_packed_to_display (src, dst);
                return;
        default:
                indirect_move (cob_move_packed_to_display, src, dst,
                               COB_FIELD_DIGITS(src), COB_FIELD_SCALE(src)
);
                return;
        }
```

```
  case COB_TYPE_NUMERIC_BINARY:
          switch (COB_FIELD_TYPE (dst)) {
          case COB_TYPE_NUMERIC_DISPLAY:
                  cob_move_binary_to_display (src, dst);
                  return;
          case COB_TYPE_NUMERIC_BINARY:
          case COB_TYPE_NUMERIC_PACKED:
          case COB_TYPE_NUMERIC_EDITED:
          case COB_TYPE_NUMERIC_FLOAT:
          case COB_TYPE_NUMERIC_DOUBLE:
                  indirect_move (cob_move_binary_to_display, src, dst,
                                 20, COB_FIELD_SCALE(src));
                  return;
          default:
                  indirect_move (cob_move_binary_to_display, src, dst,
                                 COB_FIELD_DIGITS(src), COB_FIELD_SCALE(src)
);
                  return;
          }

  case COB_TYPE_NUMERIC_EDITED:
          switch (COB_FIELD_TYPE (dst)) {
          case COB_TYPE_NUMERIC_DISPLAY:
                  cob_move_edited_to_display (src, dst);
                  return;
          case COB_TYPE_NUMERIC_PACKED:
          case COB_TYPE_NUMERIC_BINARY:
          case COB_TYPE_NUMERIC_EDITED:
          case COB_TYPE_NUMERIC_FLOAT:
          case COB_TYPE_NUMERIC_DOUBLE:
                  indirect_move (cob_move_edited_to_display, src, dst, 36,
18);
                  return;
          case COB_TYPE_ALPHANUMERIC_EDITED:
                  cob_move_alphanum_to_edited (src, dst);
                  return;
          default:
                  cob_move_alphanum_to_alphanum (src, dst);
                  return;
          }

  case COB_TYPE_NUMERIC_FLOAT:
  case COB_TYPE_NUMERIC_DOUBLE:
          indirect_move (cob_move_fp_to_display, src, dst, 40, 20);
          return;

  default:
          switch (COB_FIELD_TYPE (dst)) {
          case COB_TYPE_NUMERIC_DISPLAY:
                  cob_move_alphanum_to_display (src, dst);
                  return;
          case COB_TYPE_NUMERIC_PACKED:
          case COB_TYPE_NUMERIC_BINARY:
          case COB_TYPE_NUMERIC_EDITED:
          case COB_TYPE_NUMERIC_FLOAT:
          case COB_TYPE_NUMERIC_DOUBLE:
                  indirect_move (cob_move_alphanum_to_display, src, dst, 36
, 18);
                  return;
          case COB_TYPE_ALPHANUMERIC_EDITED:
                  cob_move_alphanum_to_edited (src, dst);
                  return;
```

```
                default:
                        cob_move_alphanum_to_alphanum (src, dst);
                        return;
                }
        }
}
```

7.38.1.5 void cob_set_int (cob_field * f, int n)

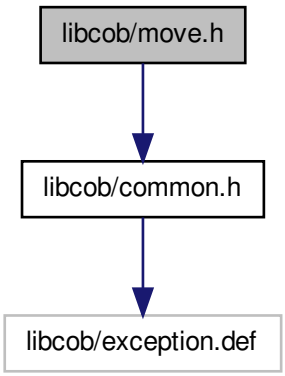
Definition at line 1291 of file move.c.

```
{
        cob_field       temp;
        cob_field_attr  attr;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 9, 0, COB_FLAG_HAVE_SIGN, NULL);
        temp.size = 4;
        temp.data = (unsigned char *)&n;
        temp.attr = &attr;
        cob_move (&temp, f);
}
```

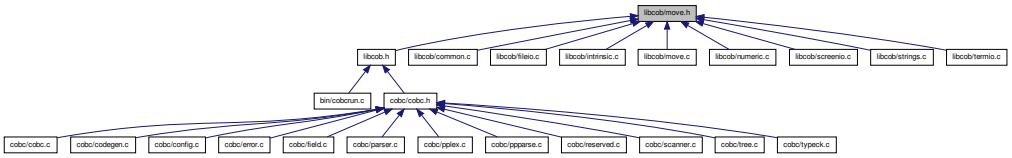
7.39 libcob/move.h File Reference

```
#include <libcob/common.h>
```

Include dependency graph for move.h:



This graph shows which files directly or indirectly include this file:



Functions

- void cob_move (cob_field *, cob_field *)
- void cob_set_int (cob_field *, int)
- int cob_get_int (cob_field *)

7.39.1 Function Documentation

7.39.1.1 int cob_get_int (cob_field *)

Definition at line 1304 of file move.c.

```
{
        int             n;
        cob_field       temp;
        cob_field_attr  attr;

        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                return cob_display_get_int (f);
        case COB_TYPE_NUMERIC_BINARY:
                return (int)cob_binary_mget_int64 (f);
        case COB_TYPE_NUMERIC_PACKED:
                return cob_packed_get_int (f);
        default:
                COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 9, 0,
                               COB_FLAG_HAVE_SIGN, NULL);
                temp.size = 4;
                temp.data = (unsigned char *)&n;
                temp.attr = &attr;
                cob_move (f, &temp);
                return n;
        }
}
```

7.39.1.2 void cob_move (cob_field * , cob_field *)

Definition at line 1023 of file move.c.

```
{
        if (COB_FIELD_TYPE (src) == COB_TYPE_ALPHANUMERIC_ALL) {
                cob_move_all (src, dst);
                return;
        }
        if (dst->size == 0) {
                return;
        }
        if (src->size == 0) {
                src = &cob_space;
        }

        /* non-elementary move */
        if (COB_FIELD_TYPE (src) == COB_TYPE_GROUP || COB_FIELD_TYPE (dst) ==
      COB_TYPE_GROUP) {
                cob_move_alphanum_to_alphanum (src, dst);
                return;
        }

        /* elementary move */
        switch (COB_FIELD_TYPE (src)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                switch (COB_FIELD_TYPE (dst)) {
                case COB_TYPE_NUMERIC_FLOAT:
                case COB_TYPE_NUMERIC_DOUBLE:
                        cob_move_display_to_fp (src, dst);
                        return;
                case COB_TYPE_NUMERIC_DISPLAY:
                        cob_move_display_to_display (src, dst);
                        return;
                case COB_TYPE_NUMERIC_PACKED:
                        cob_move_display_to_packed (src, dst);
                        return;
                case COB_TYPE_NUMERIC_BINARY:
                        cob_move_display_to_binary (src, dst);
                        return;
                case COB_TYPE_NUMERIC_EDITED:
                        cob_move_display_to_edited (src, dst);
                        return;
                case COB_TYPE_ALPHANUMERIC_EDITED:
                        if (COB_FIELD_SCALE(src) < 0 ||
                            COB_FIELD_SCALE(src) > COB_FIELD_DIGITS(src)) {
                                /* expand P's */
                                indirect_move (cob_move_display_to_display, src,
      dst,
                                               (size_t)cob_max_int ((int)
      COB_FIELD_DIGITS(src), (int)COB_FIELD_SCALE(src)),
                                               cob_max_int (0, (int)
      COB_FIELD_SCALE(src)));
                                return;
                        } else {
                                cob_move_alphanum_to_edited (src, dst);
                                return;
                        }
                default:
                        cob_move_display_to_alphanum (src, dst);
                        return;
                }

        case COB_TYPE_NUMERIC_PACKED:
                switch (COB_FIELD_TYPE (dst)) {
                case COB_TYPE_NUMERIC_DISPLAY:
```

```
                cob_move_packed_to_display (src, dst);
                return;
        default:
                indirect_move (cob_move_packed_to_display, src, dst,
                               COB_FIELD_DIGITS(src), COB_FIELD_SCALE(src)
);
                return;
        }

  case COB_TYPE_NUMERIC_BINARY:
        switch (COB_FIELD_TYPE (dst)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                cob_move_binary_to_display (src, dst);
                return;
        case COB_TYPE_NUMERIC_BINARY:
        case COB_TYPE_NUMERIC_PACKED:
        case COB_TYPE_NUMERIC_EDITED:
        case COB_TYPE_NUMERIC_FLOAT:
        case COB_TYPE_NUMERIC_DOUBLE:
                indirect_move (cob_move_binary_to_display, src, dst,
                               20, COB_FIELD_SCALE(src));
                return;
        default:
                indirect_move (cob_move_binary_to_display, src, dst,
                               COB_FIELD_DIGITS(src), COB_FIELD_SCALE(src)
);
                return;
        }

  case COB_TYPE_NUMERIC_EDITED:
        switch (COB_FIELD_TYPE (dst)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                cob_move_edited_to_display (src, dst);
                return;
        case COB_TYPE_NUMERIC_PACKED:
        case COB_TYPE_NUMERIC_BINARY:
        case COB_TYPE_NUMERIC_EDITED:
        case COB_TYPE_NUMERIC_FLOAT:
        case COB_TYPE_NUMERIC_DOUBLE:
                indirect_move (cob_move_edited_to_display, src, dst, 36,
18);
                return;
        case COB_TYPE_ALPHANUMERIC_EDITED:
                cob_move_alphanum_to_edited (src, dst);
                return;
        default:
                cob_move_alphanum_to_alphanum (src, dst);
                return;
        }

  case COB_TYPE_NUMERIC_FLOAT:
  case COB_TYPE_NUMERIC_DOUBLE:
        indirect_move (cob_move_fp_to_display, src, dst, 40, 20);
        return;

  default:
        switch (COB_FIELD_TYPE (dst)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                cob_move_alphanum_to_display (src, dst);
                return;
        case COB_TYPE_NUMERIC_PACKED:
        case COB_TYPE_NUMERIC_BINARY:
```

```
                case COB_TYPE_NUMERIC_EDITED:
                case COB_TYPE_NUMERIC_FLOAT:
                case COB_TYPE_NUMERIC_DOUBLE:
                        indirect_move (cob_move_alphanum_to_display, src, dst, 36
      , 18);
                        return;
                case COB_TYPE_ALPHANUMERIC_EDITED:
                        cob_move_alphanum_to_edited (src, dst);
                        return;
                default:
                        cob_move_alphanum_to_alphanum (src, dst);
                        return;
                }
        }
}
```

7.39.1.3 void cob_set_int (cob_field *, int)

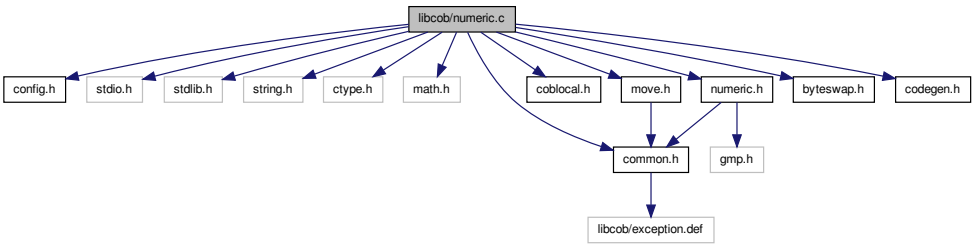
Definition at line 1291 of file move.c.

```
{
        cob_field       temp;
        cob_field_attr  attr;

        COB_ATTR_INIT (COB_TYPE_NUMERIC_BINARY, 9, 0, COB_FLAG_HAVE_SIGN, NULL);
        temp.size = 4;
        temp.data = (unsigned char *)&n;
        temp.attr = &attr;
        cob_move (&temp, f);
}
```

7.40 libcob/numeric.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "common.h"
#include "coblocal.h"
#include "move.h"
#include "numeric.h"
#include "byteswap.h"
#include "codegen.h"
```

Include dependency graph for numeric.c:



Defines

- #define COB_LIB_INCLUDE
- #define DECIMAL_NAN -128
- #define DECIMAL_CHECK(d1, d2)
- #define COB_MAX_BINARY 36

Functions

- void cob_decimal_init (cob_decimal *d)
- void cob_set_packed_zero (cob_field *f)
- void cob_set_packed_int (cob_field *f, const int val)
- void cob_decimal_set_field (cob_decimal *d, cob_field *f)
- int cob_decimal_get_field (cob_decimal *d, cob_field *f, const int opt)
- void cob_decimal_add (cob_decimal *d1, cob_decimal *d2)
- void cob_decimal_sub (cob_decimal *d1, cob_decimal *d2)
- void cob_decimal_mul (cob_decimal *d1, cob_decimal *d2)
- void cob_decimal_div (cob_decimal *d1, cob_decimal *d2)
- void cob_decimal_pow (cob_decimal *d1, cob_decimal *d2)
- int cob_decimal_cmp (cob_decimal *d1, cob_decimal *d2)
- int cob_add (cob_field *f1, cob_field *f2, const int opt)
- int cob_sub (cob_field *f1, cob_field *f2, const int opt)
- int cob_add_int (cob_field *f, const int n)
- int cob_sub_int (cob_field *f, const int n)
- int cob_div_quotient (cob_field *dividend, cob_field *divisor, cob_field *quotient, const int opt)
- int cob_div_remainder (cob_field *fld_remainder, const int opt)
- int cob_cmp_int (cob_field *f1, const int n)
- int cob_cmp_uint (cob_field *f1, const unsigned int n)
- int cob_numeric_cmp (cob_field *f1, cob_field *f2)
- int cob_cmp_packed (cob_field *f, int n)
- void cob_init_numeric (void)

- int cob_cmp_numdisp (const unsigned char *data, const size_t size, const int n)
- int cob_cmp_long_numdisp (const unsigned char *data, const size_t size, const int n)
- int cob_cmp_sign_numdisp (const unsigned char *data, const size_t size, const int n)
- int cob_cmp_long_sign_numdisp (const unsigned char *data, const size_t size, const int n)

7.40.1 Define Documentation

7.40.1.1 #define COB_LIB_INCLUDE

Definition at line 35 of file numeric.c.

7.40.1.2 #define COB_MAX_BINARY 36

Definition at line 45 of file numeric.c.

7.40.1.3 #define DECIMAL_CHECK(*d1*, d2)

Value:

```
if (unlikely(d1->scale == DECIMAL_NAN || d2->scale == DECIMAL_NAN)) { \
      d1->scale = DECIMAL_NAN; \
      return; \
    }
```

Definition at line 39 of file numeric.c.

7.40.1.4 #define DECIMAL_NAN -128

Definition at line 38 of file numeric.c.

7.40.2 Function Documentation

7.40.2.1 int cob_add (cob_field * *f1*, cob_field * *f2*, const int *opt*)

Definition at line 1215 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        cob_decimal_set_field (&cob_d2, f2);
        cob_decimal_add (&cob_d1, &cob_d2);
        return cob_decimal_get_field (&cob_d1, f1, opt);
}
```

7.40.2.2 int cob_add_int (cob_field * f, const int n)

Definition at line 1233 of file numeric.c.

```
{
        if (unlikely(n == 0)) {
                return 0;
        }
        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                return cob_display_add_int (f, n);
        case COB_TYPE_NUMERIC_PACKED:
                cob_add_packed (f, n);
                return 0;
        default:
                /* not optimized */
                cob_decimal_set_field (&cob_d1, f);
                mpz_set_si (cob_d2.value, n);
                cob_d2.scale = 0;
                if (cob_d1.scale) {
                        mpz_ui_pow_ui (cob_mexp, 10, (unsigned int)cob_d1.scale);

                        mpz_mul (cob_d2.value, cob_d2.value, cob_mexp);
                        cob_d2.scale = cob_d1.scale;
                }
                mpz_add (cob_d1.value, cob_d1.value, cob_d2.value);
                return cob_decimal_get_field (&cob_d1, f, 0);
        }
}
```

7.40.2.3 int cob_cmp_int (cob_field * f1, const int n)

Definition at line 1306 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        mpz_set_si (cob_d2.value, n);
        cob_d2.scale = 0;
        return cob_decimal_cmp (&cob_d1, &cob_d2);
}
```

7.40.2.4 int cob_cmp_long_numdisp (const unsigned char * data, const size_t size, const int n)

Definition at line 1431 of file numeric.c.

```
{
        const unsigned char     *p;
        long long               val = 0;
        size_t                  inc;

        p = data;
        for (inc = 0; inc < size; inc++, p++) {
                val = (val * 10) + (*p - (unsigned char)'0');
```

```
        }
        return (val < n) ? -1 : (val > n);
}
```

7.40.2.5 int cob_cmp_long_sign_numdisp (const unsigned char * *data,* const size_t *size,* const int *n*)

Definition at line 1682 of file numeric.c.

```
{
        const unsigned char     *p;
        long long               val = 0;
        size_t                  inc;

        p = data;
        for (inc = 0; inc < size - 1; inc++, p++) {
                val = (val * 10) + (*p - (unsigned char)'0');
        }
        val *= 10;
        if (*p >= '0' && *p <= '9') {
                val += (*p - (unsigned char)'0');
        } else {
                if (unlikely(cob_current_module->display_sign)) {
                        if (cob_get_long_ebcdic_sign (p, &val)) {
                                val = -val;
                        }
                } else {
#ifdef  COB_EBCDIC_MACHINE
                        cob_get_long_ascii_sign (p, &val);
#else
                        val += (*p - (unsigned char)'p');
#endif
                        val = -val;
                }
        }
        return (val < n) ? -1 : (val > n);
}
```

7.40.2.6 int cob_cmp_numdisp (const unsigned char * *data,* const size_t *size,* const int *n*)

Definition at line 1417 of file numeric.c.

```
{
        const unsigned char     *p;
        size_t                  inc;
        int                     val = 0;

        p = data;
        for (inc = 0; inc < size; inc++, p++) {
                val = (val * 10) + (*p - (unsigned char)'0');
        }
        return (val < n) ? -1 : (val > n);
}
```

7.40.2.7 int cob_cmp_packed (cob_field * f, int n)

Definition at line 1332 of file numeric.c.

```
{
        static int              lastval = 0;

        unsigned char           *p;
        size_t                  size;
        size_t                  inc = 0;
        int                     sign;
        unsigned char           val1[20];

        sign = cob_packed_get_sign (f);
        /* Field positive, value negative */
        if (sign >= 0 && n < 0) {
                return 1;
        }
        /* Field negative, value positive */
        if (sign < 0 && n >= 0) {
                return -1;
        }
        /* Both positive or both negative */
        p = f->data;
        for (size = 0; size < 20; size++) {
                if (size < 20 - f->size) {
                        val1[size] = 0;
                } else {
                        val1[size] = p[inc++];
                }
        }
        val1[19] &= 0xf0;
        if ((COB_FIELD_DIGITS(f) % 2) == 0) {
                val1[20 - f->size] &= 0x0f;
        }
        if (n != lastval) {
                lastval = n;
                if (n < 0) {
                        n = -n;
                }
                memset (&packed_value[14], 0, 6);
                if (n) {
                        p = &packed_value[19];
                        *p =  (n % 10) << 4;
                        p--;
                        n /= 10;
                        for (; n;) {
                                size = n % 100;
                                *p = (unsigned char)((size % 10) | ((size / 10) <
      < 4));
                                n /= 100;
                                p--;
                        }
                }
        }
        for (size = 0; size < 20; size++) {
                if (val1[size] != packed_value[size]) {
                        if (sign < 0) {
                                return packed_value[size] - val1[size];
                        } else {
                                return val1[size] - packed_value[size];
                        }
```

```
                }
        }
        return 0;
}
```

7.40.2.8 int cob_cmp_sign_numdisp (const unsigned char * *data*, const size_t *size*, const int *n*)

Definition at line 1651 of file numeric.c.

```
{
        const unsigned char     *p;
        int                     val = 0;
        size_t                  inc;

        p = data;
        for (inc = 0; inc < size - 1; inc++, p++) {
                val = (val * 10) + (*p - (unsigned char)'0');
        }
        val *= 10;
        if (*p >= '0' && *p <= '9') {
                val += (*p - (unsigned char)'0');
        } else {
                if (unlikely(cob_current_module->display_sign)) {
                        if (cob_get_ebcdic_sign (p, &val)) {
                                val = -val;
                        }
                } else {
#ifdef  COB_EBCDIC_MACHINE
                        cob_get_ascii_sign (p, &val);
#else
                        val += (*p - (unsigned char)'p');
#endif
                        val = -val;
                }
        }
        return (val < n) ? -1 : (val > n);
}
```

7.40.2.9 int cob_cmp_uint (cob_field * *f1*, const unsigned int *n*)

Definition at line 1315 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        mpz_set_ui (cob_d2.value, n);
        cob_d2.scale = 0;
        return cob_decimal_cmp (&cob_d1, &cob_d2);
}
```

7.40.2.10 void cob_decimal_add (cob_decimal * *d1*, cob_decimal * *d2*)

Definition at line 982 of file numeric.c.

```
{
        DECIMAL_CHECK (d1, d2);
        align_decimal (d1, d2);
        mpz_add (d1->value, d1->value, d2->value);
}
```

7.40.2.11 int cob_decimal_cmp (cob_decimal * *d1*, cob_decimal * *d2*)

Definition at line 1043 of file numeric.c.

```
{
        align_decimal (d1, d2);
        return mpz_cmp (d1->value, d2->value);
}
```

7.40.2.12 void cob_decimal_div (cob_decimal * *d1*, cob_decimal * *d2*)

Definition at line 1006 of file numeric.c.

```
{
        DECIMAL_CHECK (d1, d2);

        /* check for division by zero */
        if (unlikely(mpz_sgn (d2->value) == 0)) {
                d1->scale = DECIMAL_NAN;
                cob_set_exception (COB_EC_SIZE_ZERO_DIVIDE);
                return;
        }
        if (unlikely(mpz_sgn (d1->value) == 0)) {
                d1->scale = 0;
                return;
        }
        d1->scale -= d2->scale;
        shift_decimal (d1, 37 + ((d1->scale < 0) ? -d1->scale : 0));
        mpz_tdiv_q (d1->value, d1->value, d2->value);
}
```

7.40.2.13 int cob_decimal_get_field (cob_decimal * *d*, cob_field * *f*, const int *opt*)

Definition at line 910 of file numeric.c.

```
{
        cob_field       temp;
        cob_field_attr  attr;
        double          val;
        float           fval;
        int             sign;
        unsigned char   data[64];

        if (unlikely(d->scale == DECIMAL_NAN)) {
                cob_set_exception (COB_EC_SIZE_OVERFLOW);
                return cob_exception_code;
```

```
	}

	/* work copy */
	if (d != &cob_d1) {
		cob_decimal_set (&cob_d1, d);
		d = &cob_d1;
	}

	/* rounding */
	if (opt & COB_STORE_ROUND) {
		if (COB_FIELD_SCALE(f) < d->scale) {
			sign = mpz_sgn (d->value);
			if (sign != 0) {
				shift_decimal (d, COB_FIELD_SCALE(f) - d->scale +
      1);
				if (sign > 0) {
					mpz_add_ui (d->value, d->value, 5);
				} else {
					mpz_sub_ui (d->value, d->value, 5);
				}
			}
		}
	}

	/* append or truncate decimal digits */
	shift_decimal (d, COB_FIELD_SCALE(f) - d->scale);

	/* store number */
	switch (COB_FIELD_TYPE (f)) {
	case COB_TYPE_NUMERIC_BINARY:
		return cob_decimal_get_binary (d, f, opt);
	case COB_TYPE_NUMERIC_PACKED:
		return cob_decimal_get_packed (d, f, opt);
	case COB_TYPE_NUMERIC_DISPLAY:
		return cob_decimal_get_display (d, f, opt);
	case COB_TYPE_NUMERIC_FLOAT:
		fval = (float) cob_decimal_get_double (d);
		memcpy (f->data, (ucharptr)&fval, sizeof (float));
		return 0;
	case COB_TYPE_NUMERIC_DOUBLE:
		val = cob_decimal_get_double (d);
		memcpy (f->data, (ucharptr)&val, sizeof (double));
		return 0;
	default:
		COB_ATTR_INIT (COB_TYPE_NUMERIC_DISPLAY, COB_FIELD_DIGITS(f),
			       COB_FIELD_SCALE(f), COB_FLAG_HAVE_SIGN, NULL);
		temp.size = COB_FIELD_DIGITS(f);
		temp.data = data;
		temp.attr = &attr;
		if (cob_decimal_get_display (d, &temp, opt) == 0) {
			cob_move (&temp, f);
		}
		return cob_exception_code;
	}
}
```

7.40.2.14 void cob_decimal_init (cob_decimal * d)

Definition at line 277 of file numeric.c.

```
{
        mpz_init2 (d->value, 256);
        d->scale = 0;
}
```

7.40.2.15 void cob_decimal_mul (cob_decimal * d1, cob_decimal * d2)

Definition at line 998 of file numeric.c.

```
{
        DECIMAL_CHECK (d1, d2);
        d1->scale += d2->scale;
        mpz_mul (d1->value, d1->value, d2->value);
}
```

7.40.2.16 void cob_decimal_pow (cob_decimal * d1, cob_decimal * d2)

Definition at line 1026 of file numeric.c.

```
{
        unsigned int    n;

        DECIMAL_CHECK (d1, d2);

        if (d2->scale == 0 && mpz_fits_ulong_p (d2->value)) {
                n = mpz_get_ui (d2->value);
                mpz_pow_ui (d1->value, d1->value, n);
                d1->scale *= n;
        } else {
                cob_decimal_set_double (d1, pow (cob_decimal_get_double (d1),
                                                 cob_decimal_get_double (d2)));
        }
}
```

7.40.2.17 void cob_decimal_set_field (cob_decimal * d, cob_field * f)

Definition at line 883 of file numeric.c.

```
{
        double  dval;
        float   fval;

        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_BINARY:
                cob_decimal_set_binary (d, f);
                break;
        case COB_TYPE_NUMERIC_PACKED:
                cob_decimal_set_packed (d, f);
                break;
        case COB_TYPE_NUMERIC_FLOAT:
                memcpy ((ucharptr)&fval, f->data, sizeof(float));
                cob_decimal_set_double (d, (double)fval);
```

```
                break;
        case COB_TYPE_NUMERIC_DOUBLE:
                memcpy ((ucharptr)&dval, f->data, sizeof(double));
                cob_decimal_set_double (d, dval);
                break;
        default:
                cob_decimal_set_display (d, f);
                break;
        }
}
```

7.40.2.18 void cob_decimal_sub (cob_decimal * *d1*, cob_decimal * *d2*)

Definition at line 990 of file numeric.c.

```
{
        DECIMAL_CHECK (d1, d2);
        align_decimal (d1, d2);
        mpz_sub (d1->value, d1->value, d2->value);
}
```

7.40.2.19 int cob_div_quotient (cob_field * *dividend*, cob_field * *divisor*, cob_field * *quotient*, const int *opt*)

Definition at line 1269 of file numeric.c.

```
{
        int     ret;

        cob_decimal_set_field (&cob_d1, dividend);
        cob_decimal_set_field (&cob_d2, divisor);
        cob_decimal_set (&cob_d3, &cob_d1);

        /* compute quotient */
        cob_decimal_div (&cob_d1, &cob_d2);
        if (cob_d1.scale == DECIMAL_NAN) {
                cob_d3.scale = DECIMAL_NAN;
                return cob_exception_code;
        }

        /* set quotient */
        cob_decimal_set (&cob_d4, &cob_d1);
        ret = cob_decimal_get_field (&cob_d1, quotient, opt);

        /* truncate digits from the quotient */
        shift_decimal (&cob_d4, COB_FIELD_SCALE(quotient) - cob_d4.scale);

        /* compute remainder */
        cob_decimal_mul (&cob_d4, &cob_d2);
        cob_decimal_sub (&cob_d3, &cob_d4);

        return ret;
}
```

7.40.2.20 int cob_div_remainder (cob_field * *fld_remainder*, const int *opt*)

Definition at line 1300 of file numeric.c.

```
{
        return cob_decimal_get_field (&cob_d3, fld_remainder, opt);
}
```

7.40.2.21 void cob_init_numeric (void)

Definition at line 1396 of file numeric.c.

```
{
        size_t  i;

        cob_decimal_init (&cob_d1);
        cob_decimal_init (&cob_d2);
        cob_decimal_init (&cob_d3);
        cob_decimal_init (&cob_d4);
        mpz_init2 (cob_mpzt, 256);
        mpz_init2 (cob_mexp, 512);
        for (i = 0; i < COB_MAX_BINARY; i++) {
                mpz_init (cob_mpze10[i]);
                mpz_ui_pow_ui (cob_mpze10[i], 10, i);
        }
        num_buff_ptr = cob_malloc (2048);
        memset (packed_value, 0, sizeof(packed_value));
}
```

7.40.2.22 int cob_numeric_cmp (cob_field * *f1*, cob_field * *f2*)

Definition at line 1324 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        cob_decimal_set_field (&cob_d2, f2);
        return cob_decimal_cmp (&cob_d1, &cob_d2);
}
```

7.40.2.23 void cob_set_packed_int (cob_field * *f*, const int *val*)

Definition at line 847 of file numeric.c.

```
{
        unsigned char   *p;
        size_t          sign = 0;
        int             n;

        if (val < 0) {
                n = -val;
```

```
                sign = 1;
        } else {
                n = val;
        }
        memset (f->data, 0, f->size);
        p = f->data + f->size - 1;
        *p = (n % 10) << 4;
        if (!COB_FIELD_HAVE_SIGN (f)) {
                *p |= 0x0f;
        } else if (sign) {
                *p |= 0x0d;
        } else {
                *p |= 0x0c;
        }
        n /= 10;
        p--;
        for (; n && p >= f->data; n /= 100, p--) {
                *p = packed_bytes[n % 100];
        }
        /* Fixme */
        if ((COB_FIELD_DIGITS(f) % 2) == 0) {
                *(f->data) &= 0x0f;
        }
}
```

7.40.2.24 void cob_set_packed_zero (cob_field * f)

Definition at line 836 of file numeric.c.

```
{
        memset (f->data, 0, f->size);
        if (!COB_FIELD_HAVE_SIGN (f)) {
                *(f->data + f->size - 1) = 0x0f;
        } else {
                *(f->data + f->size - 1) = 0x0c;
        }
}
```

7.40.2.25 int cob_sub (cob_field * f1, cob_field * f2, const int opt)

Definition at line 1224 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        cob_decimal_set_field (&cob_d2, f2);
        cob_decimal_sub (&cob_d1, &cob_d2);
        return cob_decimal_get_field (&cob_d1, f1, opt);
}
```

7.40.2.26 int cob_sub_int (cob_field * f, const int n)

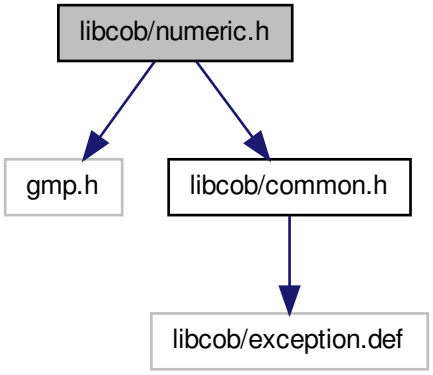
Definition at line 1260 of file numeric.c.

```
{
        if (unlikely(n == 0)) {
                return 0;
        }
        return cob_add_int (f, -n);
}
```

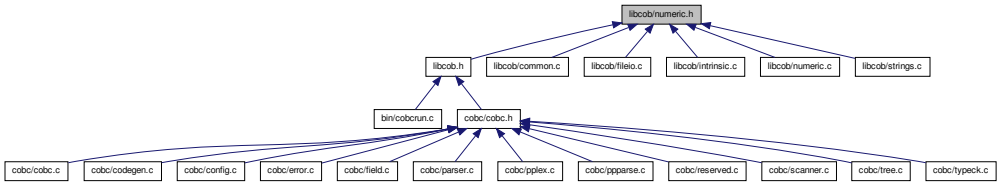
7.41 libcob/numeric.h File Reference

```
#include <gmp.h>
#include <libcob/common.h>
```

Include dependency graph for numeric.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct cob_decimal

Defines

- #define COB_STORE_ROUND 0x01
- #define COB_STORE_KEEP_ON_OVERFLOW 0x02
- #define COB_STORE_TRUNC_ON_OVERFLOW 0x04

Functions

- void cob_decimal_init (cob_decimal *)
- void cob_decimal_set_field (cob_decimal *, cob_field *)
- int cob_decimal_get_field (cob_decimal *, cob_field *, const int)
- void cob_decimal_add (cob_decimal *, cob_decimal *)
- void cob_decimal_sub (cob_decimal *, cob_decimal *)
- void cob_decimal_mul (cob_decimal *, cob_decimal *)
- void cob_decimal_div (cob_decimal *, cob_decimal *)
- void cob_decimal_pow (cob_decimal *, cob_decimal *)
- int cob_decimal_cmp (cob_decimal *, cob_decimal *)
- int cob_add (cob_field *, cob_field *, const int)
- int cob_sub (cob_field *, cob_field *, const int)
- int cob_add_int (cob_field *, const int)
- int cob_sub_int (cob_field *, const int)
- int cob_div_quotient (cob_field *, cob_field *, cob_field *, const int)
- int cob_div_remainder (cob_field *, const int)
- int cob_cmp_int (cob_field *, const int)
- int cob_cmp_uint (cob_field *, const unsigned int)
- int cob_cmp_packed (cob_field *, int)
- int cob_cmp_numdisp (const unsigned char *, const size_t, const int)
- int cob_cmp_sign_numdisp (const unsigned char *, const size_t, const int)
- int cob_cmp_long_numdisp (const unsigned char *, const size_t, const int)
- void cob_set_packed_zero (cob_field *)
- void cob_set_packed_int (cob_field *, const int)
- int cob_cmp_long_sign_numdisp (const unsigned char *, const size_t, const int)

7.41.1 Define Documentation

7.41.1.1 #define COB_STORE_KEEP_ON_OVERFLOW 0x02

Definition at line 28 of file numeric.h.

7.41.1.2 #define COB_STORE_ROUND 0x01

Definition at line 27 of file numeric.h.

7.41.1.3 #define COB_STORE_TRUNC_ON_OVERFLOW 0x04

Definition at line 29 of file numeric.h.

7.41.2 Function Documentation

7.41.2.1 int cob_add (cob_field *, cob_field *, const int)

Definition at line 1215 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        cob_decimal_set_field (&cob_d2, f2);
        cob_decimal_add (&cob_d1, &cob_d2);
        return cob_decimal_get_field (&cob_d1, f1, opt);
}
```

7.41.2.2 int cob_add_int (cob_field *, const int)

Definition at line 1233 of file numeric.c.

```
{
        if (unlikely(n == 0)) {
                return 0;
        }
        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_DISPLAY:
                return cob_display_add_int (f, n);
        case COB_TYPE_NUMERIC_PACKED:
                cob_add_packed (f, n);
                return 0;
        default:
                /* not optimized */
                cob_decimal_set_field (&cob_d1, f);
                mpz_set_si (cob_d2.value, n);
                cob_d2.scale = 0;
                if (cob_d1.scale) {
                        mpz_ui_pow_ui (cob_mexp, 10, (unsigned int)cob_d1.scale);

                        mpz_mul (cob_d2.value, cob_d2.value, cob_mexp);
                        cob_d2.scale = cob_d1.scale;
                }
                mpz_add (cob_d1.value, cob_d1.value, cob_d2.value);
                return cob_decimal_get_field (&cob_d1, f, 0);
        }
}
```

7.41.2.3 int cob_cmp_int (cob_field *, const int)

Definition at line 1306 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        mpz_set_si (cob_d2.value, n);
        cob_d2.scale = 0;
        return cob_decimal_cmp (&cob_d1, &cob_d2);
}
```

7.41.2.4 int cob_cmp_long_numdisp (const unsigned char *, const size_t , const int)

Definition at line 1431 of file numeric.c.

```
{
        const unsigned char     *p;
        long long               val = 0;
        size_t                  inc;

        p = data;
        for (inc = 0; inc < size; inc++, p++) {
                val = (val * 10) + (*p - (unsigned char)'0');
        }
        return (val < n) ? -1 : (val > n);
}
```

7.41.2.5 int cob_cmp_long_sign_numdisp (const unsigned char *, const size_t , const int)

Definition at line 1682 of file numeric.c.

```
{
        const unsigned char     *p;
        long long               val = 0;
        size_t                  inc;

        p = data;
        for (inc = 0; inc < size - 1; inc++, p++) {
                val = (val * 10) + (*p - (unsigned char)'0');
        }
        val *= 10;
        if (*p >= '0' && *p <= '9') {
                val += (*p - (unsigned char)'0');
        } else {
                if (unlikely(cob_current_module->display_sign)) {
                        if (cob_get_long_ebcdic_sign (p, &val)) {
                                val = -val;
                        }
                } else {
#ifdef  COB_EBCDIC_MACHINE
                        cob_get_long_ascii_sign (p, &val);
#else
                        val += (*p - (unsigned char)'p');
#endif
```

```
                        val = -val;
                }
        }
        return (val < n) ? -1 : (val > n);
}
```

7.41.2.6 int cob_cmp_numdisp (const unsigned char *, const size_t, const int)

Definition at line 1417 of file numeric.c.

```
{
        const unsigned char     *p;
        size_t                  inc;
        int                     val = 0;

        p = data;
        for (inc = 0; inc < size; inc++, p++) {
                val = (val * 10) + (*p - (unsigned char)'0');
        }
        return (val < n) ? -1 : (val > n);
}
```

7.41.2.7 int cob_cmp_packed (cob_field *, int)

Definition at line 1332 of file numeric.c.

```
{
        static int              lastval = 0;

        unsigned char           *p;
        size_t                  size;
        size_t                  inc = 0;
        int                     sign;
        unsigned char           val1[20];

        sign = cob_packed_get_sign (f);
        /* Field positive, value negative */
        if (sign >= 0 && n < 0) {
                return 1;
        }
        /* Field negative, value positive */
        if (sign < 0 && n >= 0) {
                return -1;
        }
        /* Both positive or both negative */
        p = f->data;
        for (size = 0; size < 20; size++) {
                if (size < 20 - f->size) {
                        val1[size] = 0;
                } else {
                        val1[size] = p[inc++];
                }
        }
        val1[19] &= 0xf0;
        if ((COB_FIELD_DIGITS(f) % 2) == 0) {
```

```
                val1[20 - f->size] &= 0x0f;
        }
        if (n != lastval) {
                lastval = n;
                if (n < 0) {
                        n = -n;
                }
                memset (&packed_value[14], 0, 6);
                if (n) {
                        p = &packed_value[19];
                        *p =  (n % 10) << 4;
                        p--;
                        n /= 10;
                        for (; n;) {
                                size = n % 100;
                                *p = (unsigned char)((size % 10) | ((size / 10) <
      < 4));
                                n /= 100;
                                p--;
                        }
                }
        }
        for (size = 0; size < 20; size++) {
                if (val1[size] != packed_value[size]) {
                        if (sign < 0) {
                                return packed_value[size] - val1[size];
                        } else {
                                return val1[size] - packed_value[size];
                        }
                }
        }
        return 0;
}
```

7.41.2.8 int cob_cmp_sign_numdisp (const unsigned char *, const size_t, const int)

Definition at line 1651 of file numeric.c.

```
{
        const unsigned char     *p;
        int                     val = 0;
        size_t                  inc;

        p = data;
        for (inc = 0; inc < size - 1; inc++, p++) {
                val = (val * 10) + (*p - (unsigned char)'0');
        }
        val *= 10;
        if (*p >= '0' && *p <= '9') {
                val += (*p - (unsigned char)'0');
        } else {
                if (unlikely(cob_current_module->display_sign)) {
                        if (cob_get_ebcdic_sign (p, &val)) {
                                val = -val;
                        }
                } else {
#ifdef  COB_EBCDIC_MACHINE
                        cob_get_ascii_sign (p, &val);
#else
```

```
                        val += (*p - (unsigned char)'p');
#endif
                        val = -val;
                }
        }
        return (val < n) ? -1 : (val > n);
}
```

7.41.2.9 int cob_cmp_uint (cob_field *, const unsigned *int*)

Definition at line 1315 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        mpz_set_ui (cob_d2.value, n);
        cob_d2.scale = 0;
        return cob_decimal_cmp (&cob_d1, &cob_d2);
}
```

7.41.2.10 void cob_decimal_add (cob_decimal *, cob_decimal *)

Definition at line 982 of file numeric.c.

```
{
        DECIMAL_CHECK (d1, d2);
        align_decimal (d1, d2);
        mpz_add (d1->value, d1->value, d2->value);
}
```

7.41.2.11 int cob_decimal_cmp (cob_decimal *, cob_decimal *)

Definition at line 1043 of file numeric.c.

```
{
        align_decimal (d1, d2);
        return mpz_cmp (d1->value, d2->value);
}
```

7.41.2.12 void cob_decimal_div (cob_decimal *, cob_decimal *)

Definition at line 1006 of file numeric.c.

```
{
        DECIMAL_CHECK (d1, d2);

        /* check for division by zero */
        if (unlikely(mpz_sgn (d2->value) == 0)) {
                d1->scale = DECIMAL_NAN;
```

```
                cob_set_exception (COB_EC_SIZE_ZERO_DIVIDE);
                return;
        }
        if (unlikely(mpz_sgn (d1->value) == 0)) {
                d1->scale = 0;
                return;
        }
        d1->scale -= d2->scale;
        shift_decimal (d1, 37 + ((d1->scale < 0) ? -d1->scale : 0));
        mpz_tdiv_q (d1->value, d1->value, d2->value);
}
```

7.41.2.13 int cob_decimal_get_field (cob_decimal *, cob_field *, const int)

Definition at line 910 of file numeric.c.

```
{
        cob_field       temp;
        cob_field_attr  attr;
        double          val;
        float           fval;
        int             sign;
        unsigned char   data[64];

        if (unlikely(d->scale == DECIMAL_NAN)) {
                cob_set_exception (COB_EC_SIZE_OVERFLOW);
                return cob_exception_code;
        }

        /* work copy */
        if (d != &cob_d1) {
                cob_decimal_set (&cob_d1, d);
                d = &cob_d1;
        }

        /* rounding */
        if (opt & COB_STORE_ROUND) {
                if (COB_FIELD_SCALE(f) < d->scale) {
                        sign = mpz_sgn (d->value);
                        if (sign != 0) {
                                shift_decimal (d, COB_FIELD_SCALE(f) - d->scale +
       1);
                                if (sign > 0) {
                                        mpz_add_ui (d->value, d->value, 5);
                                } else {
                                        mpz_sub_ui (d->value, d->value, 5);
                                }
                        }
                }
        }

        /* append or truncate decimal digits */
        shift_decimal (d, COB_FIELD_SCALE(f) - d->scale);

        /* store number */
        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_BINARY:
                return cob_decimal_get_binary (d, f, opt);
        case COB_TYPE_NUMERIC_PACKED:
```

```
                return cob_decimal_get_packed (d, f, opt);
        case COB_TYPE_NUMERIC_DISPLAY:
                return cob_decimal_get_display (d, f, opt);
        case COB_TYPE_NUMERIC_FLOAT:
                fval = (float) cob_decimal_get_double (d);
                memcpy (f->data, (ucharptr)&fval, sizeof (float));
                return 0;
        case COB_TYPE_NUMERIC_DOUBLE:
                val = cob_decimal_get_double (d);
                memcpy (f->data, (ucharptr)&val, sizeof (double));
                return 0;
        default:
                COB_ATTR_INIT (COB_TYPE_NUMERIC_DISPLAY, COB_FIELD_DIGITS(f),
                               COB_FIELD_SCALE(f), COB_FLAG_HAVE_SIGN, NULL);
                temp.size = COB_FIELD_DIGITS(f);
                temp.data = data;
                temp.attr = &attr;
                if (cob_decimal_get_display (d, &temp, opt) == 0) {
                        cob_move (&temp, f);
                }
                return cob_exception_code;
        }
}
```

7.41.2.14 void cob_decimal_init (cob_decimal *)

Definition at line 277 of file numeric.c.

```
{
        mpz_init2 (d->value, 256);
        d->scale = 0;
}
```

7.41.2.15 void cob_decimal_mul (cob_decimal *, cob_decimal *)

Definition at line 998 of file numeric.c.

```
{
        DECIMAL_CHECK (d1, d2);
        d1->scale += d2->scale;
        mpz_mul (d1->value, d1->value, d2->value);
}
```

7.41.2.16 void cob_decimal_pow (cob_decimal *, cob_decimal *)

Definition at line 1026 of file numeric.c.

```
{
        unsigned int    n;

        DECIMAL_CHECK (d1, d2);
```

```
        if (d2->scale == 0 && mpz_fits_ulong_p (d2->value)) {
                n = mpz_get_ui (d2->value);
                mpz_pow_ui (d1->value, d1->value, n);
                d1->scale *= n;
        } else {
                cob_decimal_set_double (d1, pow (cob_decimal_get_double (d1),
                                                 cob_decimal_get_double (d2)));
        }
}
```

7.41.2.17 void cob_decimal_set_field (cob_decimal *, cob_field *)

Definition at line 883 of file numeric.c.

```
{
        double  dval;
        float   fval;

        switch (COB_FIELD_TYPE (f)) {
        case COB_TYPE_NUMERIC_BINARY:
                cob_decimal_set_binary (d, f);
                break;
        case COB_TYPE_NUMERIC_PACKED:
                cob_decimal_set_packed (d, f);
                break;
        case COB_TYPE_NUMERIC_FLOAT:
                memcpy ((ucharptr)&fval, f->data, sizeof(float));
                cob_decimal_set_double (d, (double)fval);
                break;
        case COB_TYPE_NUMERIC_DOUBLE:
                memcpy ((ucharptr)&dval, f->data, sizeof(double));
                cob_decimal_set_double (d, dval);
                break;
        default:
                cob_decimal_set_display (d, f);
                break;
        }
}
```

7.41.2.18 void cob_decimal_sub (cob_decimal *, cob_decimal *)

Definition at line 990 of file numeric.c.

```
{
        DECIMAL_CHECK (d1, d2);
        align_decimal (d1, d2);
        mpz_sub (d1->value, d1->value, d2->value);
}
```

7.41.2.19 int cob_div_quotient (cob_field *, cob_field *, cob_field *, const int)

Definition at line 1269 of file numeric.c.

```
{
        int     ret;

        cob_decimal_set_field (&cob_d1, dividend);
        cob_decimal_set_field (&cob_d2, divisor);
        cob_decimal_set (&cob_d3, &cob_d1);

        /* compute quotient */
        cob_decimal_div (&cob_d1, &cob_d2);
        if (cob_d1.scale == DECIMAL_NAN) {
                cob_d3.scale = DECIMAL_NAN;
                return cob_exception_code;
        }

        /* set quotient */
        cob_decimal_set (&cob_d4, &cob_d1);
        ret = cob_decimal_get_field (&cob_d1, quotient, opt);

        /* truncate digits from the quotient */
        shift_decimal (&cob_d4, COB_FIELD_SCALE(quotient) - cob_d4.scale);

        /* compute remainder */
        cob_decimal_mul (&cob_d4, &cob_d2);
        cob_decimal_sub (&cob_d3, &cob_d4);

        return ret;
}
```

7.41.2.20 int cob_div_remainder (cob_field *, const int)

Definition at line 1300 of file numeric.c.

```
{
        return cob_decimal_get_field (&cob_d3, fld_remainder, opt);
}
```

7.41.2.21 void cob_set_packed_int (cob_field *, const int)

Definition at line 847 of file numeric.c.

```
{
        unsigned char   *p;
        size_t          sign = 0;
        int             n;

        if (val < 0) {
                n = -val;
                sign = 1;
        } else {
                n = val;
        }
        memset (f->data, 0, f->size);
        p = f->data + f->size - 1;
        *p = (n % 10) << 4;
        if (!COB_FIELD_HAVE_SIGN (f)) {
```

```
                *p |= 0x0f;
        } else if (sign) {
                *p |= 0x0d;
        } else {
                *p |= 0x0c;
        }
        n /= 10;
        p--;
        for (; n && p >= f->data; n /= 100, p--) {
                *p = packed_bytes[n % 100];
        }
        /* Fixme */
        if ((COB_FIELD_DIGITS(f) % 2) == 0) {
                *(f->data) &= 0x0f;
        }
}
```

7.41.2.22 void cob_set_packed_zero (cob_field *)

Definition at line 836 of file numeric.c.

```
{
        memset (f->data, 0, f->size);
        if (!COB_FIELD_HAVE_SIGN (f)) {
                *(f->data + f->size - 1) = 0x0f;
        } else {
                *(f->data + f->size - 1) = 0x0c;
        }
}
```

7.41.2.23 int cob_sub (cob_field *, cob_field *, const int)

Definition at line 1224 of file numeric.c.

```
{
        cob_decimal_set_field (&cob_d1, f1);
        cob_decimal_set_field (&cob_d2, f2);
        cob_decimal_sub (&cob_d1, &cob_d2);
        return cob_decimal_get_field (&cob_d1, f1, opt);
}
```

7.41.2.24 int cob_sub_int (cob_field *, const int)

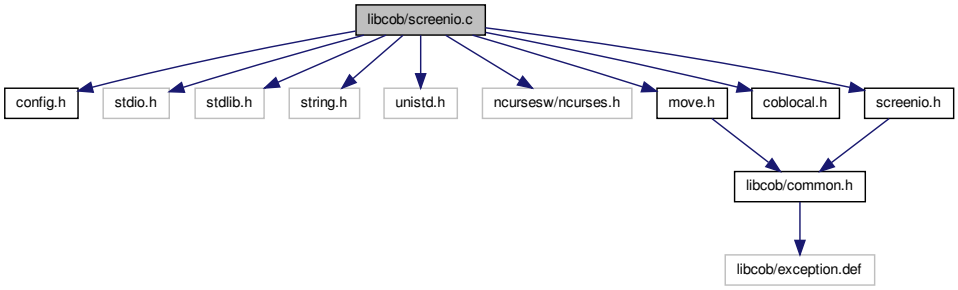
Definition at line 1260 of file numeric.c.

```
{
        if (unlikely(n == 0)) {
                return 0;
        }
        return cob_add_int (f, -n);
}
```

7.42 libcob/screenio.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ncursesw/ncurses.h>
#include "move.h"
#include "coblocal.h"
#include "screenio.h"
```

Include dependency graph for screenio.c:



Classes

- struct cob_inp_struct

Defines

- #define _XOPEN_SOURCE_EXTENDED
- #define COB_GEN_SCREENIO
- #define COB_INP_SIZE 1920 * sizeof(struct cob_inp_struct)

Functions

- void cob_screen_terminate (void)
- void cob_screen_display (cob_screen *s, cob_field *line, cob_field *column)
- void cob_screen_accept (cob_screen *s, cob_field *line, cob_field *column)

- void cob_field_display (cob_field *f, cob_field *line, cob_field *column, cob_field *fgc, cob_field *bgc, cob_field *scroll, const int attr)
- void cob_field_accept (cob_field *f, cob_field *line, cob_field *column, cob_field *fgc, cob_field *bgc, cob_field *scroll, const int attr)
- void cob_screen_line_col (cob_field *f, const int l_or_c)
- void cob_screen_set_mode (const size_t smode)

Variables

- int cob_screen_initialized = 0
- int cob_screen_mode = 0

7.42.1 Define Documentation

7.42.1.1 #define _XOPEN_SOURCE_EXTENDED

Definition at line 30 of file screenio.c.

7.42.1.2 #define COB_GEN_SCREENIO

Definition at line 33 of file screenio.c.

7.42.1.3 #define COB_INP_SIZE 1920 * sizeof(struct cob_inp_struct)

Definition at line 64 of file screenio.c.

7.42.2 Function Documentation

7.42.2.1 void cob_field_accept (cob_field * *f*, cob_field * *line*, cob_field * *column*, cob_field * *fgc*, cob_field * *bgc*, cob_field * *scroll*, const int *attr*)

Definition at line 855 of file screenio.c.

```
{
        unsigned char   *p;
        size_t          count;
        int             keyp;
        int             fret;
        int             sline;
        int             scolumn;
        int             cline;
        int             ccolumn;
        int             rightpos;
        int             ateof;
        int             gotbacksp;

        if (!cob_screen_initialized) {
                cob_screen_init ();
```

```
}

if (scroll) {
        keyp = cob_get_int (scroll);
        if (attr & COB_SCREEN_SCROLL_DOWN) {
                keyp = -keyp;
        }
        scrollok (stdscr, 1);
        scrl (keyp);
        scrollok (stdscr, 0);
        refresh ();
}
cob_exception_code = 0;
get_line_column (line, column, &sline, &scolumn);
move (sline, scolumn);
cob_screen_attr (fgc, bgc, attr);
p = f->data;
for (count = 0; count < f->size; count++) {
        if (attr & COB_SCREEN_SECURE) {
                addch ('*');
        } else if (attr & COB_SCREEN_UPDATE) {
                fret = *p++;
                addch ((unsigned int)fret);
        } else if (attr & COB_SCREEN_PROMPT) {
                addch ('_');
        } else {
                addch (' ');
        }
}
move (sline, scolumn);
if (!(attr & COB_SCREEN_UPDATE)) {
        if (COB_FIELD_IS_NUMERIC (f)) {
                cob_move (&cob_zero, f);
        } else {
                memset (f->data, ' ', f->size);
        }
}

fret = 0;
ateof = 0;
gotbacksp = 0;
rightpos = scolumn + f->size - 1;
p = f->data;
for (; ;) {
        refresh ();
        keyp = getch ();
        if (keyp == KEY_ENTER || keyp == '\n') {
                break;
        }
        if (keyp > KEY_F0 && keyp < KEY_F(65)) {
                fret = 1000 + keyp - KEY_F0;
                break;
        }
        if (cob_extended_status) {
                if (keyp == KEY_PPAGE) {
                        fret = 2001;
                        break;
                }
                if (keyp == KEY_NPAGE) {
                        fret = 2002;
                        break;
                }
```

```
        if (keyp == KEY_UP) {
                fret = 2003;
                break;
        }
        if (keyp == KEY_DOWN) {
                fret = 2004;
                break;
        }
        if (keyp == KEY_PRINT) {
                fret = 2006;
                break;
        }
        if (cob_use_esc) {
                if (keyp == 033) {
                        fret = 2005;
                        break;
                }
        }
}
getyx (stdscr, cline, ccolumn);
if (keyp == KEY_BACKSPACE || keyp == ('H' & 037) ||
    keyp == 0177) {
        if (ccolumn > scolumn) {
                if (gotbacksp || ccolumn != rightpos) {
                        ccolumn--;
                } else {
                        ateof = 0;
                }
                gotbacksp = 1;
                move (cline, ccolumn);
                if (attr & COB_SCREEN_SECURE) {
                        addch ('*');
                } else {
                        addch ('_');
                }
                move (cline, ccolumn);
                p = f->data + ccolumn - scolumn;
                *p = ' ';
                continue;
        }
}
if (keyp == KEY_HOME) {
        move (sline, scolumn);
        p = f->data;
        ateof = 0;
        gotbacksp = 0;
        continue;
}
if (keyp == KEY_END) {
        move (sline, rightpos);
        p = f->data + f->size - 1;
        ateof = 0;
        gotbacksp = 0;
        continue;
}
if (keyp == KEY_LEFT) {
        if (ccolumn > scolumn) {
                ccolumn--;
                move (cline, ccolumn);
                p = f->data + ccolumn - scolumn;
                continue;
        }
```

```
                        gotbacksp = 0;
                }
                if (keyp == KEY_RIGHT) {
                        if (ccolumn < rightpos) {
                                ccolumn++;
                                move (cline, ccolumn);
                                p = f->data + ccolumn - scolumn;
                                continue;
                        }
                        gotbacksp = 0;
                }
                if (keyp > 037 && keyp < (int)A_CHARTEXT) {
                        if (COB_FIELD_IS_NUMERIC (f)) {
                                if (keyp < '0' || keyp > '9') {
                                        beep ();
                                        continue;
                                }
                        }
                        gotbacksp = 0;
                        *p = keyp;
                        if (attr & COB_SCREEN_SECURE) {
                                addch ('*');
                        } else {
                                addch ((unsigned int)keyp);
                        }
                        if (ccolumn == rightpos) {
                                if (attr & COB_SCREEN_AUTO) {
                                        break;
                                }
                                move (cline, ccolumn);
                                if (ateof) {
                                        beep ();
                                } else {
                                        ateof = 1;
                                }
                        } else {
                                p++;
                        }
                        continue;
                }
                gotbacksp = 0;
                beep ();
        }
        cob_check_pos_status (fret);
        refresh ();
}
```

7.42.2.2 void cob_field_display (cob_field * *f,* cob_field * *line,* cob_field * *column,* cob_field * *fgc,* cob_field * *bgc,* cob_field * *scroll,* const int *attr*)

Definition at line 826 of file screenio.c.

```
{
        int sline;
        int scolumn;

        if (!cob_screen_initialized) {
                cob_screen_init ();
        }
```

```
        if (scroll) {
                sline = cob_get_int (scroll);
                if (attr & COB_SCREEN_SCROLL_DOWN) {
                        sline = -sline;
                }
                scrollok (stdscr, 1);
                scrl (sline);
                scrollok (stdscr, 0);
                refresh ();
        }
        get_line_column (line, column, &sline, &scolumn);
        move (sline, scolumn);
        cob_screen_attr (fgc, bgc, attr);
        addnstr ((char *)f->data, (int)f->size);
        refresh ();
}
```

7.42.2.3 void cob_screen_accept (cob_screen * *s,* cob_field * *line,* cob_field * *column*)

Definition at line 759 of file screenio.c.

```
{
        struct cob_inp_struct   *sptr;
        struct cob_inp_struct   *sptr2;
        size_t                  idx;
        size_t                  n;
        size_t                  posu;
        size_t                  posd;
        size_t                  prevy;
        size_t                  firsty;
        int                     starty;

        if (!cob_screen_initialized) {
                cob_screen_init ();
        }
        if (!cob_base_inp) {
                cob_base_inp = cob_malloc (COB_INP_SIZE);
        } else {
                memset (cob_base_inp, 0, COB_INP_SIZE);
        }
        cob_exception_code = 0;
        cob_current_y = 0;
        cob_current_x = 0;
        totl_index = 0;
        move (0, 0);
        cob_prep_input (s);
        /* No input fields is an error */
        if (!totl_index) {
                cob_check_pos_status (8000);
                return;
        }
        qsort (cob_base_inp, totl_index, sizeof(struct cob_inp_struct), compare_y
      x);
        sptr = cob_base_inp;
        starty = sptr->this_y;
        posu = 0;
        posd = 0;
        prevy = 0;
```

```
        firsty = 0;
        /* Set up array for Cursor UP/DOWN */
        for (n = 0; n < totl_index; n++) {
                sptr = cob_base_inp + n;
                if (sptr->this_y > starty) {
                        if (!firsty) {
                                firsty = n;
                        }
                        starty = sptr->this_y;
                        sptr2 = cob_base_inp + posd;
                        for (idx = posd; idx < n; idx++, sptr2++) {
                                sptr2->down_index = n;
                        }
                        posu = prevy;
                        prevy = n;
                        posd = n;
                }
                sptr->up_index = posu;
        }
        sptr = cob_base_inp;
        for (n = 0; n < firsty; n++, sptr++) {
                sptr->up_index = posd;
        }
        curr_index = 0;
        global_return = 0;
        cob_screen_get_all ();
        cob_check_pos_status (global_return);
}
```

7.42.2.4 void cob_screen_display (cob_screen * *s*, cob_field * *line*, cob_field * *column*)

Definition at line 726 of file screenio.c.

```
{
        int     n;

        if (!cob_screen_initialized) {
                cob_screen_init ();
        }

        switch (s->type) {
        case COB_SCREEN_TYPE_GROUP:
                for (s = s->child; s; s = s->next) {
                        cob_screen_display (s, line, column);
                }
                break;
        case COB_SCREEN_TYPE_FIELD:
                cob_screen_puts (s, s->field);
                break;
        case COB_SCREEN_TYPE_VALUE:
                cob_screen_puts (s, s->value);
                if (s->occurs) {
                        for (n = 1; n < s->occurs; ++n) {
                                cob_screen_puts (s, s->value);
                        }
                }
                break;
        case COB_SCREEN_TYPE_ATTRIBUTE:
```

```
                cob_screen_attr (s->foreg, s->backg, s->attr);
                break;
        }
        refresh ();
}
```

7.42.2.5 void cob_screen_line_col (cob_field * *f*, const int *l_or_c*)

Definition at line 1045 of file screenio.c.

```
{
        if (!cob_screen_initialized) {
                cob_screen_init ();
        }
        if (!l_or_c) {
                cob_set_int (f, (int)LINES);
        } else {
                cob_set_int (f, (int)COLS);
        }
}
```

7.42.2.6 void cob_screen_set_mode (const size_t *smode*)

Definition at line 1058 of file screenio.c.

```
{
        if (!smode) {
                refresh ();
                def_prog_mode ();
                endwin ();
        } else {
                reset_prog_mode ();
                refresh ();
        }
}
```

7.42.2.7 void cob_screen_terminate (void)

Definition at line 297 of file screenio.c.

```
{
        if (cob_screen_initialized) {
                cob_screen_initialized = 0;
                endwin ();
        }
}
```

7.42.3 Variable Documentation

7.42.3.1 int cob_screen_initialized = 0

Definition at line 51 of file screenio.c.

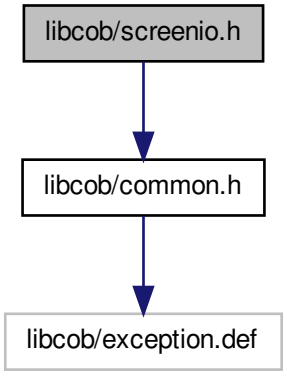
7.42.3.2 int cob_screen_mode = 0

Definition at line 52 of file screenio.c.

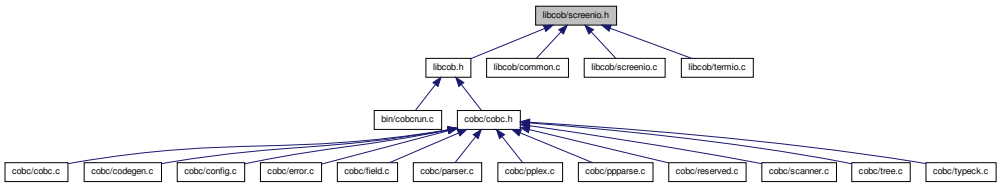
7.43 libcob/screenio.h File Reference

```
#include <libcob/common.h>
```

Include dependency graph for screenio.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct __cob_screen

Defines

- #define COB_SCREEN_BLACK 0
- #define COB_SCREEN_BLUE 1
- #define COB_SCREEN_GREEN 2
- #define COB_SCREEN_CYAN 3
- #define COB_SCREEN_RED 4
- #define COB_SCREEN_MAGENTA 5
- #define COB_SCREEN_YELLOW 6
- #define COB_SCREEN_WHITE 7
- #define COB_SCREEN_LINE_PLUS 0x00000001
- #define COB_SCREEN_LINE_MINUS 0x00000002
- #define COB_SCREEN_COLUMN_PLUS 0x00000004
- #define COB_SCREEN_COLUMN_MINUS 0x00000008
- #define COB_SCREEN_AUTO 0x00000010
- #define COB_SCREEN_BELL 0x00000020
- #define COB_SCREEN_BLANK_LINE 0x00000040
- #define COB_SCREEN_BLANK_SCREEN 0x00000080
- #define COB_SCREEN_BLINK 0x00000100
- #define COB_SCREEN_ERASE_EOL 0x00000200
- #define COB_SCREEN_ERASE_EOS 0x00000400
- #define COB_SCREEN_FULL 0x00000800
- #define COB_SCREEN_HIGHLIGHT 0x00001000
- #define COB_SCREEN_LOWLIGHT 0x00002000
- #define COB_SCREEN_REQUIRED 0x00004000
- #define COB_SCREEN_REVERSE 0x00008000
- #define COB_SCREEN_SECURE 0x00010000
- #define COB_SCREEN_UNDERLINE 0x00020000
- #define COB_SCREEN_OVERLINE 0x00040000
- #define COB_SCREEN_PROMPT 0x00080000
- #define COB_SCREEN_UPDATE 0x00100000
- #define COB_SCREEN_INPUT 0x00200000
- #define COB_SCREEN_SCROLL_DOWN 0x00400000
- #define COB_SCREEN_TYPE_GROUP 0
- #define COB_SCREEN_TYPE_FIELD 1
- #define COB_SCREEN_TYPE_VALUE 2
- #define COB_SCREEN_TYPE_ATTRIBUTE 3

Typedefs

- typedef struct __cob_screen cob_screen

Functions

- void cob_screen_line_col (cob_field *, const int)
- void cob_screen_display (cob_screen *, cob_field *, cob_field *)
- void cob_screen_accept (cob_screen *, cob_field *, cob_field *)
- void cob_field_display (cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, const int)
- void cob_field_accept (cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, const int)

Variables

- DLL_EXPIMP int cob_screen_mode

7.43.1 Define Documentation

7.43.1.1 #define COB_SCREEN_AUTO 0x00000010

Definition at line 40 of file screenio.h.

7.43.1.2 #define COB_SCREEN_BELL 0x00000020

Definition at line 41 of file screenio.h.

7.43.1.3 #define COB_SCREEN_BLACK 0

Definition at line 26 of file screenio.h.

7.43.1.4 #define COB_SCREEN_BLANK_LINE 0x00000040

Definition at line 42 of file screenio.h.

7.43.1.5 #define COB_SCREEN_BLANK_SCREEN 0x00000080

Definition at line 43 of file screenio.h.

7.43.1.6 #define COB_SCREEN_BLINK 0x00000100

Definition at line 44 of file screenio.h.

7.43.1.7 #define COB_SCREEN_BLUE 1

Definition at line 27 of file screenio.h.

7.43.1.8 #define COB_SCREEN_COLUMN_MINUS 0x00000008

Definition at line 38 of file screenio.h.

7.43.1.9 #define COB_SCREEN_COLUMN_PLUS 0x00000004

Definition at line 37 of file screenio.h.

7.43.1.10 #define COB_SCREEN_CYAN 3

Definition at line 29 of file screenio.h.

7.43.1.11 #define COB_SCREEN_ERASE_EOL 0x00000200

Definition at line 45 of file screenio.h.

7.43.1.12 #define COB_SCREEN_ERASE_EOS 0x00000400

Definition at line 46 of file screenio.h.

7.43.1.13 #define COB_SCREEN_FULL 0x00000800

Definition at line 47 of file screenio.h.

7.43.1.14 #define COB_SCREEN_GREEN 2

Definition at line 28 of file screenio.h.

7.43.1.15 #define COB_SCREEN_HIGHLIGHT 0x00001000

Definition at line 48 of file screenio.h.

7.43.1.16 #define COB_SCREEN_INPUT 0x00200000

Definition at line 57 of file screenio.h.

7.43.1.17 #define COB_SCREEN_LINE_MINUS 0x00000002

Definition at line 36 of file screenio.h.

7.43.1.18 #define COB_SCREEN_LINE_PLUS 0x00000001

Definition at line 35 of file screenio.h.

7.43.1.19 #define COB_SCREEN_LOWLIGHT 0x00002000

Definition at line 49 of file screenio.h.

7.43.1.20 #define COB_SCREEN_MAGENTA 5

Definition at line 31 of file screenio.h.

7.43.1.21 #define COB_SCREEN_OVERLINE 0x00040000

Definition at line 54 of file screenio.h.

7.43.1.22 #define COB_SCREEN_PROMPT 0x00080000

Definition at line 55 of file screenio.h.

7.43.1.23 #define COB_SCREEN_RED 4

Definition at line 30 of file screenio.h.

7.43.1.24 #define COB_SCREEN_REQUIRED 0x00004000

Definition at line 50 of file screenio.h.

7.43.1.25 #define COB_SCREEN_REVERSE 0x00008000

Definition at line 51 of file screenio.h.

7.43.1.26 #define COB_SCREEN_SCROLL_DOWN 0x00400000

Definition at line 58 of file screenio.h.

7.43.1.27 #define COB_SCREEN_SECURE 0x00010000

Definition at line 52 of file screenio.h.

7.43.1.28 #define COB_SCREEN_TYPE_ATTRIBUTE 3

Definition at line 63 of file screenio.h.

7.43.1.29 #define COB_SCREEN_TYPE_FIELD 1

Definition at line 61 of file screenio.h.

7.43.1.30 #define COB_SCREEN_TYPE_GROUP 0

Definition at line 60 of file screenio.h.

7.43.1.31 #define COB_SCREEN_TYPE_VALUE 2

Definition at line 62 of file screenio.h.

7.43.1.32 #define COB_SCREEN_UNDERLINE 0x00020000

Definition at line 53 of file screenio.h.

7.43.1.33 #define COB_SCREEN_UPDATE 0x00100000

Definition at line 56 of file screenio.h.

7.43.1.34 #define COB_SCREEN_WHITE 7

Definition at line 33 of file screenio.h.

7.43.1.35 #define COB_SCREEN_YELLOW 6

Definition at line 32 of file screenio.h.

7.43.2 Typedef Documentation

7.43.2.1 typedef struct __cob_screen cob_screen

Definition at line 65 of file screenio.h.

7.43.3 Function Documentation

7.43.3.1 void cob_field_accept (cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, const int)

Definition at line 855 of file screenio.c.

```
{
        unsigned char   *p;
        size_t          count;
        int             keyp;
        int             fret;
        int             sline;
        int             scolumn;
        int             cline;
        int             ccolumn;
        int             rightpos;
        int             ateof;
        int             gotbacksp;

        if (!cob_screen_initialized) {
                cob_screen_init ();
        }

        if (scroll) {
                keyp = cob_get_int (scroll);
                if (attr & COB_SCREEN_SCROLL_DOWN) {
                        keyp = -keyp;
                }
                scrollok (stdscr, 1);
                scrl (keyp);
                scrollok (stdscr, 0);
                refresh ();
        }
        cob_exception_code = 0;
        get_line_column (line, column, &sline, &scolumn);
        move (sline, scolumn);
        cob_screen_attr (fgc, bgc, attr);
        p = f->data;
        for (count = 0; count < f->size; count++) {
                if (attr & COB_SCREEN_SECURE) {
                        addch ('*');
                } else if (attr & COB_SCREEN_UPDATE) {
                        fret = *p++;
                        addch ((unsigned int)fret);
                } else if (attr & COB_SCREEN_PROMPT) {
                        addch ('_');
                } else {
                        addch (' ');
                }
        }
        move (sline, scolumn);
        if (!(attr & COB_SCREEN_UPDATE)) {
                if (COB_FIELD_IS_NUMERIC (f)) {
                        cob_move (&cob_zero, f);
                } else {
                        memset (f->data, ' ', f->size);
                }
        }

        fret = 0;
        ateof = 0;
        gotbacksp = 0;
```

```
rightpos = scolumn + f->size - 1;
p = f->data;
for (; ;) {
        refresh ();
        keyp = getch ();
        if (keyp == KEY_ENTER || keyp == '\n') {
                break;
        }
        if (keyp > KEY_F0 && keyp < KEY_F(65)) {
                fret = 1000 + keyp - KEY_F0;
                break;
        }
        if (cob_extended_status) {
                if (keyp == KEY_PPAGE) {
                        fret = 2001;
                        break;
                }
                if (keyp == KEY_NPAGE) {
                        fret = 2002;
                        break;
                }
                if (keyp == KEY_UP) {
                        fret = 2003;
                        break;
                }
                if (keyp == KEY_DOWN) {
                        fret = 2004;
                        break;
                }
                if (keyp == KEY_PRINT) {
                        fret = 2006;
                        break;
                }
                if (cob_use_esc) {
                        if (keyp == 033) {
                                fret = 2005;
                                break;
                        }
                }
        }
        getyx (stdscr, cline, ccolumn);
        if (keyp == KEY_BACKSPACE || keyp == ('H' & 037) ||
            keyp == 0177) {
                if (ccolumn > scolumn) {
                        if (gotbacksp || ccolumn != rightpos) {
                                ccolumn--;
                        } else {
                                ateof = 0;
                        }
                        gotbacksp = 1;
                        move (cline, ccolumn);
                        if (attr & COB_SCREEN_SECURE) {
                                addch ('*');
                        } else {
                                addch ('_');
                        }
                        move (cline, ccolumn);
                        p = f->data + ccolumn - scolumn;
                        *p = ' ';
                        continue;
                }
        }
```

```
if (keyp == KEY_HOME) {
        move (sline, scolumn);
        p = f->data;
        ateof = 0;
        gotbacksp = 0;
        continue;
}
if (keyp == KEY_END) {
        move (sline, rightpos);
        p = f->data + f->size - 1;
        ateof = 0;
        gotbacksp = 0;
        continue;
}
if (keyp == KEY_LEFT) {
        if (ccolumn > scolumn) {
                ccolumn--;
                move (cline, ccolumn);
                p = f->data + ccolumn - scolumn;
                continue;
        }
        gotbacksp = 0;
}
if (keyp == KEY_RIGHT) {
        if (ccolumn < rightpos) {
                ccolumn++;
                move (cline, ccolumn);
                p = f->data + ccolumn - scolumn;
                continue;
        }
        gotbacksp = 0;
}
if (keyp > 037 && keyp < (int)A_CHARTEXT) {
        if (COB_FIELD_IS_NUMERIC (f)) {
                if (keyp < '0' || keyp > '9') {
                        beep ();
                        continue;
                }
        }
        gotbacksp = 0;
        *p = keyp;
        if (attr & COB_SCREEN_SECURE) {
                addch ('*');
        } else {
                addch ((unsigned int)keyp);
        }
        if (ccolumn == rightpos) {
                if (attr & COB_SCREEN_AUTO) {
                        break;
                }
                move (cline, ccolumn);
                if (ateof) {
                        beep ();
                } else {
                        ateof = 1;
                }
        } else {
                p++;
        }
        continue;
}
gotbacksp = 0;
```

```
                beep ();
        }
        cob_check_pos_status (fret);
        refresh ();
}
```

7.43.3.2 void cob_field_display (cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, cob_field *, const int)

Definition at line 826 of file screenio.c.

```
{
        int sline;
        int scolumn;

        if (!cob_screen_initialized) {
                cob_screen_init ();
        }

        if (scroll) {
                sline = cob_get_int (scroll);
                if (attr & COB_SCREEN_SCROLL_DOWN) {
                        sline = -sline;
                }
                scrollok (stdscr, 1);
                scrl (sline);
                scrollok (stdscr, 0);
                refresh ();
        }
        get_line_column (line, column, &sline, &scolumn);
        move (sline, scolumn);
        cob_screen_attr (fgc, bgc, attr);
        addnstr ((char *)f->data, (int)f->size);
        refresh ();
}
```

7.43.3.3 void cob_screen_accept (cob_screen *, cob_field *, cob_field *)

Definition at line 759 of file screenio.c.

```
{
        struct cob_inp_struct   *sptr;
        struct cob_inp_struct   *sptr2;
        size_t                  idx;
        size_t                  n;
        size_t                  posu;
        size_t                  posd;
        size_t                  prevy;
        size_t                  firsty;
        int                     starty;

        if (!cob_screen_initialized) {
                cob_screen_init ();
        }
        if (!cob_base_inp) {
```

```
                cob_base_inp = cob_malloc (COB_INP_SIZE);
        } else {
                memset (cob_base_inp, 0, COB_INP_SIZE);
        }
        cob_exception_code = 0;
        cob_current_y = 0;
        cob_current_x = 0;
        totl_index = 0;
        move (0, 0);
        cob_prep_input (s);
        /* No input fields is an error */
        if (!totl_index) {
                cob_check_pos_status (8000);
                return;
        }
        qsort (cob_base_inp, totl_index, sizeof(struct cob_inp_struct), compare_y
      x);
        sptr = cob_base_inp;
        starty = sptr->this_y;
        posu = 0;
        posd = 0;
        prevy = 0;
        firsty = 0;
        /* Set up array for Cursor UP/DOWN */
        for (n = 0; n < totl_index; n++) {
                sptr = cob_base_inp + n;
                if (sptr->this_y > starty) {
                        if (!firsty) {
                                firsty = n;
                        }
                        starty = sptr->this_y;
                        sptr2 = cob_base_inp + posd;
                        for (idx = posd; idx < n; idx++, sptr2++) {
                                sptr2->down_index = n;
                        }
                        posu = prevy;
                        prevy = n;
                        posd = n;
                }
                sptr->up_index = posu;
        }
        sptr = cob_base_inp;
        for (n = 0; n < firsty; n++, sptr++) {
                sptr->up_index = posd;
        }
        curr_index = 0;
        global_return = 0;
        cob_screen_get_all ();
        cob_check_pos_status (global_return);
}
```

7.43.3.4 void cob_screen_display (cob_screen *, cob_field *, cob_field *)

Definition at line 726 of file screenio.c.

```
{
        int     n;

        if (!cob_screen_initialized) {
```

```
                cob_screen_init ();
        }

        switch (s->type) {
        case COB_SCREEN_TYPE_GROUP:
                for (s = s->child; s; s = s->next) {
                        cob_screen_display (s, line, column);
                }
                break;
        case COB_SCREEN_TYPE_FIELD:
                cob_screen_puts (s, s->field);
                break;
        case COB_SCREEN_TYPE_VALUE:
                cob_screen_puts (s, s->value);
                if (s->occurs) {
                        for (n = 1; n < s->occurs; ++n) {
                                cob_screen_puts (s, s->value);
                        }
                }
                break;
        case COB_SCREEN_TYPE_ATTRIBUTE:
                cob_screen_attr (s->foreg, s->backg, s->attr);
                break;
        }
        refresh ();
}
```

7.43.3.5 void cob_screen_line_col (cob_field *, const int)

Definition at line 1045 of file screenio.c.

```
{
        if (!cob_screen_initialized) {
                cob_screen_init ();
        }
        if (!l_or_c) {
                cob_set_int (f, (int)LINES);
        } else {
                cob_set_int (f, (int)COLS);
        }
}
```

7.43.4 Variable Documentation

7.43.4.1 DLL_EXPIMP int cob_screen_mode

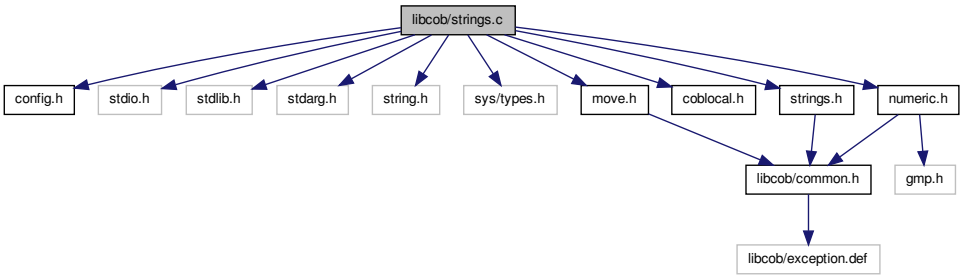
Definition at line 52 of file screenio.c.

7.44 libcob/strings.c File Reference

```
#include "config.h"

#include <stdio.h>
```

```
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <sys/types.h>
#include "move.h"
#include "coblocal.h"
#include "numeric.h"
#include "strings.h"
```

Include dependency graph for strings.c:



Classes

- struct dlm_struct

Defines

- #define INSPECT_ALL 0
- #define INSPECT_LEADING 1
- #define INSPECT_FIRST 2
- #define INSPECT_TRAILING 3
- #define DLM_DEFAULT_NUM 8

Functions

- void cob_inspect_init (cob_field *var, const int replacing)
- void cob_inspect_start (void)
- void cob_inspect_before (const cob_field *str)
- void cob_inspect_after (const cob_field *str)
- void cob_inspect_characters (cob_field *f1)

- void cob_inspect_all (cob_field *f1, cob_field *f2)
- void cob_inspect_leading (cob_field *f1, cob_field *f2)
- void cob_inspect_first (cob_field *f1, cob_field *f2)
- void cob_inspect_trailing (cob_field *f1, cob_field *f2)
- void cob_inspect_converting (cob_field *f1, cob_field *f2)
- void cob_inspect_finish (void)
- void cob_string_init (cob_field *dst, cob_field *ptr)
- void cob_string_delimited (cob_field *dlm)
- void cob_string_append (cob_field *src)
- void cob_string_finish (void)
- void cob_unstring_init (cob_field *src, cob_field *ptr, const size_t num_dlm)
- void cob_unstring_delimited (cob_field *dlm, const int all)
- void cob_unstring_into (cob_field *dst, cob_field *dlm, cob_field *cnt)
- void cob_unstring_tallying (cob_field *f)
- void cob_unstring_finish (void)
- void cob_init_strings (void)

7.44.1 Define Documentation

7.44.1.1 #define DLM_DEFAULT_NUM 8

Definition at line 39 of file strings.c.

7.44.1.2 #define INSPECT_ALL 0

Definition at line 34 of file strings.c.

7.44.1.3 #define INSPECT_FIRST 2

Definition at line 36 of file strings.c.

7.44.1.4 #define INSPECT_LEADING 1

Definition at line 35 of file strings.c.

7.44.1.5 #define INSPECT_TRAILING 3

Definition at line 37 of file strings.c.

7.44.2 Function Documentation

7.44.2.1 void cob_init_strings (void)

Definition at line 595 of file strings.c.

```
{
        inspect_mark = cob_malloc (COB_MEDIUM_BUFF);
        lastsize = COB_MEDIUM_BUFF;
        alpha_attr.type = COB_TYPE_ALPHANUMERIC;
        alpha_attr.digits = 0;
        alpha_attr.scale = 0;
        alpha_attr.flags = 0;
        alpha_attr.pic = NULL;
        alpha_fld.size = 0;
        alpha_fld.data = NULL;
        alpha_fld.attr = &alpha_attr;
}
```

7.44.2.2 void cob_inspect_after (const cob_field * str)

Definition at line 251 of file strings.c.

```
{
        unsigned char   *p;

        for (p = inspect_start; p < inspect_end - str->size + 1; p++) {
                if (memcmp (p, str->data, str->size) == 0) {
                        inspect_start = p + str->size;
                        return;
                }
        }
        inspect_start = inspect_end;
}
```

7.44.2.3 void cob_inspect_all (cob_field * f1, cob_field * f2)

Definition at line 297 of file strings.c.

```
{
        inspect_common (f1, f2, INSPECT_ALL);
}
```

7.44.2.4 void cob_inspect_before (const cob_field * str)

Definition at line 238 of file strings.c.

```
{
        unsigned char   *p;
```

```
        for (p = inspect_start; p < inspect_end - str->size + 1; p++) {
                if (memcmp (p, str->data, str->size) == 0) {
                        inspect_end = p;
                        return;
                }
        }
}
```

7.44.2.5 void cob_inspect_characters (cob_field * f1)

Definition at line 265 of file strings.c.

```
{
        int     *mark;
        int     i;
        int     n;
        int     len;

        mark = &inspect_mark[inspect_start - inspect_data];
        len = (int)(inspect_end - inspect_start);
        if (inspect_replacing) {
                /* INSPECT REPLACING CHARACTERS f1 */
                for (i = 0; i < len; i++) {
                        if (mark[i] == -1) {
                                mark[i] = f1->data[0];
                        }
                }
        } else {
                /* INSPECT TALLYING f1 CHARACTERS */
                n = 0;
                for (i = 0; i < len; i++) {
                        if (mark[i] == -1) {
                                mark[i] = 1;
                                n++;
                        }
                }
                if (n > 0) {
                        cob_add_int (f1, n);
                }
        }
}
```

7.44.2.6 void cob_inspect_converting (cob_field * f1, cob_field * f2)

Definition at line 321 of file strings.c.

```
{
        size_t  i;
        size_t  j;
        size_t  len;

        len = (size_t)(inspect_end - inspect_start);
        for (j = 0; j < f1->size; j++) {
                for (i = 0; i < len; i++) {
                        if (inspect_mark[i] == -1 && inspect_start[i] == f1->
```

```
      data[j]) {
                                inspect_start[i] = f2->data[j];
                                inspect_mark[i] = 1;
                        }
                }
        }
}
```

7.44.2.7 void cob_inspect_finish (void)

Definition at line 339 of file strings.c.

```
{
        size_t  i;

        if (inspect_replacing) {
                for (i = 0; i < inspect_size; i++) {
                        if (inspect_mark[i] != -1) {
                                inspect_data[i] = inspect_mark[i];
                        }
                }
        }

        cob_put_sign (inspect_var, inspect_sign);
}
```

7.44.2.8 void cob_inspect_first (cob_field * *f1*, cob_field * *f2*)

Definition at line 309 of file strings.c.

```
{
        inspect_common (f1, f2, INSPECT_FIRST);
}
```

7.44.2.9 void cob_inspect_init (cob_field * *var*, const int *replacing*)

Definition at line 205 of file strings.c.

```
{
        size_t          i;
        size_t          digcount;

        inspect_var_copy = *var;
        inspect_var = &inspect_var_copy;
        inspect_replacing = replacing;
        inspect_sign = cob_get_sign (var);
        inspect_size = COB_FIELD_SIZE (var);
        inspect_data = COB_FIELD_DATA (var);
        inspect_start = NULL;
        inspect_end = NULL;
        digcount = inspect_size * sizeof (int);
        if (digcount > lastsize) {
```

```
                free (inspect_mark);
                inspect_mark = cob_malloc (digcount);
                lastsize = digcount;
        }
        for (i = 0; i < inspect_size; i++) {
                inspect_mark[i] = -1;
        }
        cob_exception_code = 0;
}
```

7.44.2.10 void cob_inspect_leading (cob_field * *f1*, cob_field * *f2*)

Definition at line 303 of file strings.c.

```
{
        inspect_common (f1, f2, INSPECT_LEADING);
}
```

7.44.2.11 void cob_inspect_start (void)

Definition at line 231 of file strings.c.

```
{
        inspect_start = inspect_data;
        inspect_end = inspect_data + inspect_size;
}
```

7.44.2.12 void cob_inspect_trailing (cob_field * *f1*, cob_field * *f2*)

Definition at line 315 of file strings.c.

```
{
        inspect_common (f1, f2, INSPECT_TRAILING);
}
```

7.44.2.13 void cob_string_append (cob_field * *src*)

Definition at line 390 of file strings.c.

```
{
        size_t  src_size;
        int     i;
        int     size;

        if (cob_exception_code) {
                return;
        }
```

```
        src_size = src->size;
        if (string_dlm) {
                size = (int)(src_size - string_dlm->size + 1);
                for (i = 0; i < size; i++) {
                        if (memcmp (src->data + i, string_dlm->data, string_dlm->
      size) == 0) {
                                src_size = i;
                                break;
                        }
                }
        }

        if (src_size <= string_dst->size - string_offset) {
                memcpy (string_dst->data + string_offset, src->data, src_size);
                string_offset += (int) src_size;
        } else {
                size = (int)(string_dst->size - string_offset);
                memcpy (string_dst->data + string_offset, src->data, (size_t)size
      );
                string_offset += size;
                cob_set_exception (COB_EC_OVERFLOW_STRING);
        }
}
```

7.44.2.14 void cob_string_delimited (cob_field * *dlm*)

Definition at line 380 of file strings.c.

```
{
        string_dlm = NULL;
        if (dlm) {
                string_dlm_copy = *dlm;
                string_dlm = &string_dlm_copy;
        }
}
```

7.44.2.15 void cob_string_finish (void)

Definition at line 423 of file strings.c.

```
{
        if (string_ptr) {
                cob_set_int (string_ptr, string_offset + 1);
        }
}
```

7.44.2.16 void cob_string_init (cob_field * *dst*, cob_field * *ptr*)

Definition at line 359 of file strings.c.

```
{
        string_dst_copy = *dst;
```

```
        string_dst = &string_dst_copy;
        string_ptr = NULL;
        if (ptr) {
                string_ptr_copy = *ptr;
                string_ptr = &string_ptr_copy;
        }
        string_offset = 0;
        cob_exception_code = 0;

        if (string_ptr) {
                string_offset = cob_get_int (string_ptr) - 1;
                if (string_offset < 0 || string_offset >= (int)string_dst->size)
      {
                        cob_set_exception (COB_EC_OVERFLOW_STRING);
                }
        }
}
```

7.44.2.17 void cob_unstring_delimited (cob_field * *dlm,* const int *all*)

Definition at line 476 of file strings.c.

```
{
        dlm_list[unstring_ndlms].uns_dlm = dlm;
        dlm_list[unstring_ndlms].uns_all = all;
        unstring_ndlms++;
}
```

7.44.2.18 void cob_unstring_finish (void)

Definition at line 581 of file strings.c.

```
{
        if (unstring_offset < (int)unstring_src->size) {
                cob_set_exception (COB_EC_OVERFLOW_UNSTRING);
        }

        if (unstring_ptr) {
                cob_set_int (unstring_ptr, unstring_offset + 1);
        }
}
```

7.44.2.19 void cob_unstring_init (cob_field * *src,* cob_field * *ptr,* const size_t *num_dlm*)

Definition at line 435 of file strings.c.

```
{
        static size_t   udlmcount = 0;

        unstring_src_copy = *src;
        unstring_src = &unstring_src_copy;
        unstring_ptr = NULL;
```

```
        if (ptr) {
                unstring_ptr_copy = *ptr;
                unstring_ptr = &unstring_ptr_copy;
        }

        unstring_offset = 0;
        unstring_count = 0;
        unstring_ndlms = 0;
        cob_exception_code = 0;
        if (!dlm_list) {
                if (num_dlm <= DLM_DEFAULT_NUM) {
                        dlm_list = cob_malloc (DLM_DEFAULT_NUM * sizeof(struct
      dlm_struct));
                        udlmcount = DLM_DEFAULT_NUM;
                } else {
                        dlm_list = cob_malloc (num_dlm * sizeof(struct
      dlm_struct));
                        udlmcount = num_dlm;
                }
        } else {
                if (num_dlm > udlmcount) {
                        free (dlm_list);
                        dlm_list = cob_malloc (num_dlm * sizeof(struct
      dlm_struct));
                        udlmcount = num_dlm;
                }
        }

        if (unstring_ptr) {
                unstring_offset = cob_get_int (unstring_ptr) - 1;
                if (unstring_offset < 0 || unstring_offset >= (int)unstring_src->
      size) {
                        cob_set_exception (COB_EC_OVERFLOW_UNSTRING);
                }
        }
}
```

7.44.2.20 void cob_unstring_into (cob_field * *dst,* cob_field * *dlm,* cob_field * *cnt*)

Definition at line 484 of file strings.c.

```
{
        unsigned char   *p;
        unsigned char   *dp;
        unsigned char   *s;
        unsigned char   *dlm_data;
        unsigned char   *start;
        size_t          dlm_size = 0;
        int             i;
        int             srsize;
        int             dlsize;
        int             match_size = 0;
        int             brkpt = 0;

        if (cob_exception_code) {
                return;
        }

        if (unstring_offset >= (int)unstring_src->size) {
```

```
		return;
	}

	start = unstring_src->data + unstring_offset;
	dlm_data = NULL;
	if (unstring_ndlms == 0) {
		match_size = cob_min_int ((int)COB_FIELD_SIZE (dst),
					  (int)unstring_src->size - unstring_offs
et);
		cob_memcpy (dst, start, match_size);
		unstring_offset += match_size;
	} else {

		srsize = (int) unstring_src->size;
		s = unstring_src->data + srsize;
		for (p = start; p < s; p++) {
			for (i = 0; i < unstring_ndlms; i++) {
				dlsize = (int) dlm_list[i].uns_dlm->size;
				dp = dlm_list[i].uns_dlm->data;
				if (p + dlsize > s) {
					break;
				}
				if (!memcmp (p, dp, (size_t)dlsize)) {
					match_size = (int)(p - start);
					cob_memcpy (dst, start, match_size);
					unstring_offset += match_size + dlsize;
					dlm_data = dp;
					dlm_size = dlsize;
					if (dlm_list[i].uns_all) {
						for (p++ ; p < s; p++) {
							if (p + dlsize > s) {
								break;
							}
							if (memcmp (p, dp, (size_
t)dlsize)) {
								break;
							}
							unstring_offset += dlsize
;
						}
					}
					brkpt = 1;
					break;
				}
			}
			if (brkpt) {
				break;
			}
		}
		if (!brkpt) {
			/* no match */
			match_size = (int)(unstring_src->size - unstring_offset);

			cob_memcpy (dst, start, match_size);
			unstring_offset = (int) unstring_src->size;
			dlm_data = NULL;
		}
	}
	unstring_count++;

	if (dlm) {
		if (dlm_data) {
```

```
                        cob_memcpy (dlm, dlm_data, (int) dlm_size);
                } else if (COB_FIELD_IS_NUMERIC (dlm)) {
                        cob_move (&cob_zero, dlm);
                } else {
                        cob_move (&cob_space, dlm);
                }
        }

        if (cnt) {
                cob_set_int (cnt, match_size);
        }
}
```

7.44.2.21 void cob_unstring_tallying (cob_field * f)

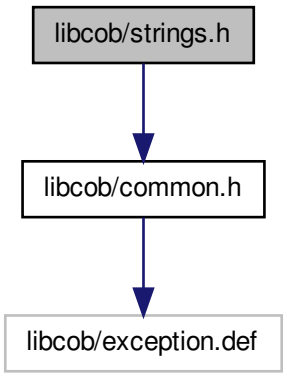
Definition at line 575 of file strings.c.

```
{
        cob_add_int (f, unstring_count);
}
```

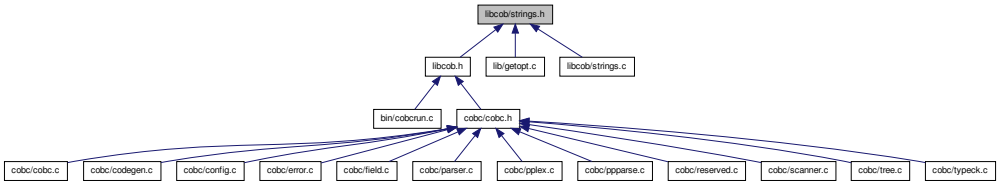
7.45 libcob/strings.h File Reference

```
#include <libcob/common.h>
```

Include dependency graph for strings.h:



This graph shows which files directly or indirectly include this file:



Functions

- void cob_inspect_init (cob_field *, const int)
- void cob_inspect_start (void)
- void cob_inspect_before (const cob_field *)
- void cob_inspect_after (const cob_field *)
- void cob_inspect_characters (cob_field *)
- void cob_inspect_all (cob_field *, cob_field *)
- void cob_inspect_leading (cob_field *, cob_field *)
- void cob_inspect_first (cob_field *, cob_field *)
- void cob_inspect_trailing (cob_field *, cob_field *)
- void cob_inspect_converting (cob_field *, cob_field *)
- void cob_inspect_finish (void)
- void cob_string_init (cob_field *, cob_field *)
- void cob_string_delimited (cob_field *)
- void cob_string_append (cob_field *)
- void cob_string_finish (void)
- void cob_unstring_init (cob_field *, cob_field *, const size_t)
- void cob_unstring_delimited (cob_field *, const int)
- void cob_unstring_into (cob_field *, cob_field *, cob_field *)
- void cob_unstring_tallying (cob_field *)
- void cob_unstring_finish (void)

7.45.1 Function Documentation

7.45.1.1 void cob_inspect_after (const cob_field *)

Definition at line 251 of file strings.c.

```
{
        unsigned char   *p;

        for (p = inspect_start; p < inspect_end - str->size + 1; p++) {
                if (memcmp (p, str->data, str->size) == 0) {
                        inspect_start = p + str->size;
                        return;
```

```
                }
        }
        inspect_start = inspect_end;
}
```

7.45.1.2 void cob_inspect_all (cob_field * , cob_field *)

Definition at line 297 of file strings.c.

```
{
        inspect_common (f1, f2, INSPECT_ALL);
}
```

7.45.1.3 void cob_inspect_before (const cob_field *)

Definition at line 238 of file strings.c.

```
{
        unsigned char   *p;

        for (p = inspect_start; p < inspect_end - str->size + 1; p++) {
                if (memcmp (p, str->data, str->size) == 0) {
                        inspect_end = p;
                        return;
                }
        }
}
```

7.45.1.4 void cob_inspect_characters (cob_field *)

Definition at line 265 of file strings.c.

```
{
        int     *mark;
        int     i;
        int     n;
        int     len;

        mark = &inspect_mark[inspect_start - inspect_data];
        len = (int)(inspect_end - inspect_start);
        if (inspect_replacing) {
                /* INSPECT REPLACING CHARACTERS f1 */
                for (i = 0; i < len; i++) {
                        if (mark[i] == -1) {
                                mark[i] = f1->data[0];
                        }
                }
        } else {
                /* INSPECT TALLYING f1 CHARACTERS */
                n = 0;
                for (i = 0; i < len; i++) {
                        if (mark[i] == -1) {
```

```
                                mark[i] = 1;
                                n++;
                        }
                }
                if (n > 0) {
                        cob_add_int (f1, n);
                }
        }
}
```

7.45.1.5 void cob_inspect_converting (cob_field *, cob_field *)

Definition at line 321 of file strings.c.

```
{
        size_t  i;
        size_t  j;
        size_t  len;

        len = (size_t)(inspect_end - inspect_start);
        for (j = 0; j < f1->size; j++) {
                for (i = 0; i < len; i++) {
                        if (inspect_mark[i] == -1 && inspect_start[i] == f1->data
      [j]) {
                                inspect_start[i] = f2->data[j];
                                inspect_mark[i] = 1;
                        }
                }
        }
}
```

7.45.1.6 void cob_inspect_finish (void)

Definition at line 339 of file strings.c.

```
{
        size_t  i;

        if (inspect_replacing) {
                for (i = 0; i < inspect_size; i++) {
                        if (inspect_mark[i] != -1) {
                                inspect_data[i] = inspect_mark[i];
                        }
                }
        }

        cob_put_sign (inspect_var, inspect_sign);
}
```

7.45.1.7 void cob_inspect_first (cob_field *, cob_field *)

Definition at line 309 of file strings.c.

```
{
        inspect_common (f1, f2, INSPECT_FIRST);
}
```

7.45.1.8 void cob_inspect_init (cob_field *, const int)

Definition at line 205 of file strings.c.

```
{
        size_t          i;
        size_t          digcount;

        inspect_var_copy = *var;
        inspect_var = &inspect_var_copy;
        inspect_replacing = replacing;
        inspect_sign = cob_get_sign (var);
        inspect_size = COB_FIELD_SIZE (var);
        inspect_data = COB_FIELD_DATA (var);
        inspect_start = NULL;
        inspect_end = NULL;
        digcount = inspect_size * sizeof (int);
        if (digcount > lastsize) {
                free (inspect_mark);
                inspect_mark = cob_malloc (digcount);
                lastsize = digcount;
        }
        for (i = 0; i < inspect_size; i++) {
                inspect_mark[i] = -1;
        }
        cob_exception_code = 0;
}
```

7.45.1.9 void cob_inspect_leading (cob_field *, cob_field *)

Definition at line 303 of file strings.c.

```
{
        inspect_common (f1, f2, INSPECT_LEADING);
}
```

7.45.1.10 void cob_inspect_start (void)

Definition at line 231 of file strings.c.

```
{
        inspect_start = inspect_data;
        inspect_end = inspect_data + inspect_size;
}
```

7.45.1.11 void cob_inspect_trailing (cob_field * , cob_field *)

Definition at line 315 of file strings.c.

```
{
        inspect_common (f1, f2, INSPECT_TRAILING);
}
```

7.45.1.12 void cob_string_append (cob_field *)

Definition at line 390 of file strings.c.

```
{
        size_t  src_size;
        int     i;
        int     size;

        if (cob_exception_code) {
                return;
        }

        src_size = src->size;
        if (string_dlm) {
                size = (int)(src_size - string_dlm->size + 1);
                for (i = 0; i < size; i++) {
                        if (memcmp (src->data + i, string_dlm->data, string_dlm->
      size) == 0) {
                                src_size = i;
                                break;
                        }
                }
        }

        if (src_size <= string_dst->size - string_offset) {
                memcpy (string_dst->data + string_offset, src->data, src_size);
                string_offset += (int) src_size;
        } else {
                size = (int)(string_dst->size - string_offset);
                memcpy (string_dst->data + string_offset, src->data, (size_t)size
      );
                string_offset += size;
                cob_set_exception (COB_EC_OVERFLOW_STRING);
        }
}
```

7.45.1.13 void cob_string_delimited (cob_field *)

Definition at line 380 of file strings.c.

```
{
        string_dlm = NULL;
        if (dlm) {
                string_dlm_copy = *dlm;
                string_dlm = &string_dlm_copy;
        }
}
```

7.45.1.14 void cob_string_finish (void)

Definition at line 423 of file strings.c.

```
{
        if (string_ptr) {
                cob_set_int (string_ptr, string_offset + 1);
        }
}
```

7.45.1.15 void cob_string_init (cob_field *, cob_field *)

Definition at line 359 of file strings.c.

```
{
        string_dst_copy = *dst;
        string_dst = &string_dst_copy;
        string_ptr = NULL;
        if (ptr) {
                string_ptr_copy = *ptr;
                string_ptr = &string_ptr_copy;
        }
        string_offset = 0;
        cob_exception_code = 0;

        if (string_ptr) {
                string_offset = cob_get_int (string_ptr) - 1;
                if (string_offset < 0 || string_offset >= (int)string_dst->size)
      {
                        cob_set_exception (COB_EC_OVERFLOW_STRING);
                }
        }
}
```

7.45.1.16 void cob_unstring_delimited (cob_field *, const int)

Definition at line 476 of file strings.c.

```
{
        dlm_list[unstring_ndlms].uns_dlm = dlm;
        dlm_list[unstring_ndlms].uns_all = all;
        unstring_ndlms++;
}
```

7.45.1.17 void cob_unstring_finish (void)

Definition at line 581 of file strings.c.

```
{
        if (unstring_offset < (int)unstring_src->size) {
```

```
                cob_set_exception (COB_EC_OVERFLOW_UNSTRING);
        }

        if (unstring_ptr) {
                cob_set_int (unstring_ptr, unstring_offset + 1);
        }
}
```

7.45.1.18 void cob_unstring_init (cob_field *, cob_field *, const size_t)

Definition at line 435 of file strings.c.

```
{
        static size_t   udlmcount = 0;

        unstring_src_copy = *src;
        unstring_src = &unstring_src_copy;
        unstring_ptr = NULL;
        if (ptr) {
                unstring_ptr_copy = *ptr;
                unstring_ptr = &unstring_ptr_copy;
        }

        unstring_offset = 0;
        unstring_count = 0;
        unstring_ndlms = 0;
        cob_exception_code = 0;
        if (!dlm_list) {
                if (num_dlm <= DLM_DEFAULT_NUM) {
                        dlm_list = cob_malloc (DLM_DEFAULT_NUM * sizeof(struct
      dlm_struct));
                        udlmcount = DLM_DEFAULT_NUM;
                } else {
                        dlm_list = cob_malloc (num_dlm * sizeof(struct
      dlm_struct));
                        udlmcount = num_dlm;
                }
        } else {
                if (num_dlm > udlmcount) {
                        free (dlm_list);
                        dlm_list = cob_malloc (num_dlm * sizeof(struct
      dlm_struct));
                        udlmcount = num_dlm;
                }
        }

        if (unstring_ptr) {
                unstring_offset = cob_get_int (unstring_ptr) - 1;
                if (unstring_offset < 0 || unstring_offset >= (int)unstring_src->
      size) {
                        cob_set_exception (COB_EC_OVERFLOW_UNSTRING);
                }
        }
}
```

7.45.1.19 void cob_unstring_into (cob_field *, cob_field *, cob_field *)

Definition at line 484 of file strings.c.

```
{
        unsigned char   *p;
        unsigned char   *dp;
        unsigned char   *s;
        unsigned char   *dlm_data;
        unsigned char   *start;
        size_t          dlm_size = 0;
        int             i;
        int             srsize;
        int             dlsize;
        int             match_size = 0;
        int             brkpt = 0;

        if (cob_exception_code) {
                return;
        }

        if (unstring_offset >= (int)unstring_src->size) {
                return;
        }

        start = unstring_src->data + unstring_offset;
        dlm_data = NULL;
        if (unstring_ndlms == 0) {
                match_size = cob_min_int ((int)COB_FIELD_SIZE (dst),
                                          (int)unstring_src->size - unstring_offs
      et);
                cob_memcpy (dst, start, match_size);
                unstring_offset += match_size;
        } else {

                srsize = (int) unstring_src->size;
                s = unstring_src->data + srsize;
                for (p = start; p < s; p++) {
                        for (i = 0; i < unstring_ndlms; i++) {
                                dlsize = (int) dlm_list[i].uns_dlm->size;
                                dp = dlm_list[i].uns_dlm->data;
                                if (p + dlsize > s) {
                                        break;
                                }
                                if (!memcmp (p, dp, (size_t)dlsize)) {
                                        match_size = (int)(p - start);
                                        cob_memcpy (dst, start, match_size);
                                        unstring_offset += match_size + dlsize;
                                        dlm_data = dp;
                                        dlm_size = dlsize;
                                        if (dlm_list[i].uns_all) {
                                                for (p++ ; p < s; p++) {
                                                        if (p + dlsize > s) {
                                                                break;
                                                        }
                                                        if (memcmp (p, dp, (size_
      t)dlsize)) {
                                                                break;
                                                        }
                                                        unstring_offset += dlsize
      ;
                                                }
```

```
                                        }
                                        brkpt = 1;
                                        break;
                                }
                        }
                        if (brkpt) {
                                break;
                        }
                }
                if (!brkpt) {
                        /* no match */
                        match_size = (int)(unstring_src->size - unstring_offset);

                        cob_memcpy (dst, start, match_size);
                        unstring_offset = (int) unstring_src->size;
                        dlm_data = NULL;
                }
        }
        unstring_count++;

        if (dlm) {
                if (dlm_data) {
                        cob_memcpy (dlm, dlm_data, (int) dlm_size);
                } else if (COB_FIELD_IS_NUMERIC (dlm)) {
                        cob_move (&cob_zero, dlm);
                } else {
                        cob_move (&cob_space, dlm);
                }
        }

        if (cnt) {
                cob_set_int (cnt, match_size);
        }
}
```

7.45.1.20 void cob_unstring_tallying (cob_field *)

Definition at line 575 of file strings.c.

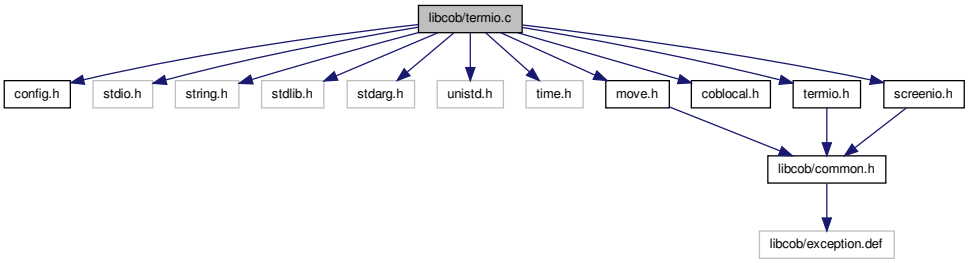
```
{
        cob_add_int (f, unstring_count);
}
```

7.46 libcob/termio.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
#include <unistd.h>
```

```
#include <time.h>
#include "move.h"
#include "coblocal.h"
#include "termio.h"
#include "screenio.h"
```

Include dependency graph for termio.c:



Functions

- void cob_display (const int outorerr, const int newline, const int varcnt,...)
- void cob_accept (cob_field *f)
- void cob_init_termio (void)

7.46.1 Function Documentation

7.46.1.1 void cob_accept (cob_field * f)

Definition at line 234 of file termio.c.

```
{
/* RXW
        size_t          size;
*/
        cob_field_attr  attr;
        cob_field       temp;

        if (cob_screen_initialized) {
                cob_field_accept (f, NULL, NULL, NULL, NULL, NULL, 0);
                return;
        }
        temp.data = term_buff;
        temp.attr = &attr;
        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        /* read a line */
        if (fgets ((char *)term_buff, COB_MEDIUM_BUFF, stdin) == NULL) {
```

```
                temp.size = 1;
                term_buff[0] = ' ';
                term_buff[1] = 0;
        } else {
                temp.size = strlen ((char *)term_buff) - 1;
        }
        if (COB_FIELD_TYPE(f) == COB_TYPE_NUMERIC_DISPLAY) {
                if (temp.size > f->size) {
                        temp.size = f->size;
                }
        }
        cob_move (&temp, f);
/* RXW
        if (isatty (fileno (stdin))) {
                temp.size = strlen ((char *)term_buff) - 1;
                cob_move (&temp, f);
        } else {
                size = strlen ((char *)term_buff) - 1;
                if (size > f->size) {
                        size = f->size;
                }
                memcpy (f->data, term_buff, size);
                memset (f->data + size, ' ', f->size - size);
        }
*/
}
```

7.46.1.2 void cob_display (const int *outorerr,* const int *newline,* const int *varcnt,* ...)

Definition at line 205 of file termio.c.

```
{
        FILE            *fp;
        cob_field       *f;
        int             i;
        va_list         args;

        if (!outorerr && !cob_screen_initialized) {
                fp = stdout;
        } else {
                fp = stderr;
        }
        va_start (args, varcnt);
        for (i = 0; i < varcnt; ++i) {
                f = va_arg (args, cob_field *);
                display (f, fp);
        }
        va_end (args);
        if (newline) {
                putc ('\n', fp);
                fflush (fp);
        }
}
```

7.46.1.3 void cob_init_termio (void)

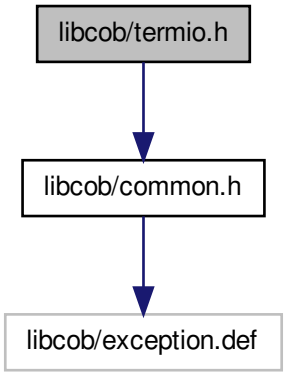
Definition at line 279 of file termio.c.

```
{
        term_buff = cob_malloc (COB_MEDIUM_BUFF);
}
```

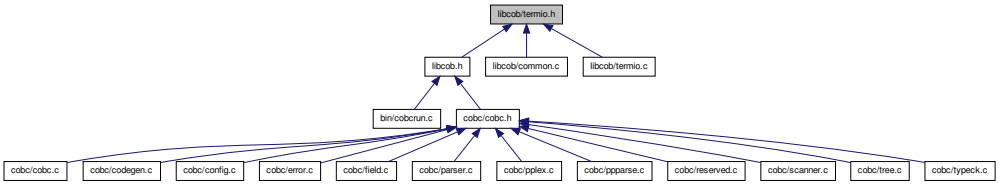
7.47 libcob/termio.h File Reference

```
#include <libcob/common.h>
```

Include dependency graph for termio.h:



This graph shows which files directly or indirectly include this file:



Functions

- void cob_display (const int, const int, const int,...)
- void cob_accept (cob_field *)

7.47.1 Function Documentation

7.47.1.1 void cob_accept (cob_field *)

Definition at line 234 of file termio.c.

```
{
/* RXW
        size_t          size;
*/
        cob_field_attr  attr;
        cob_field       temp;

        if (cob_screen_initialized) {
                cob_field_accept (f, NULL, NULL, NULL, NULL, NULL, 0);
                return;
        }
        temp.data = term_buff;
        temp.attr = &attr;
        COB_ATTR_INIT (COB_TYPE_ALPHANUMERIC, 0, 0, 0, NULL);
        /* read a line */
        if (fgets ((char *)term_buff, COB_MEDIUM_BUFF, stdin) == NULL) {
                temp.size = 1;
                term_buff[0] = ' ';
                term_buff[1] = 0;
        } else {
                temp.size = strlen ((char *)term_buff) - 1;
        }
        if (COB_FIELD_TYPE(f) == COB_TYPE_NUMERIC_DISPLAY) {
                if (temp.size > f->size) {
                        temp.size = f->size;
                }
        }
        cob_move (&temp, f);
/* RXW
        if (isatty (fileno (stdin))) {
                temp.size = strlen ((char *)term_buff) - 1;
                cob_move (&temp, f);
        } else {
                size = strlen ((char *)term_buff) - 1;
                if (size > f->size) {
                        size = f->size;
                }
                memcpy (f->data, term_buff, size);
                memset (f->data + size, ' ', f->size - size);
        }
*/
}
```

7.47.1.2 void cob_display (const int , const int , const int , ...)

Definition at line 205 of file termio.c.

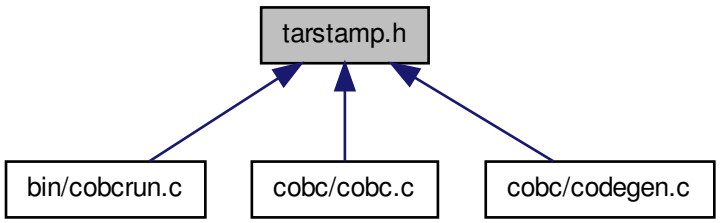
```
{
        FILE            *fp;
        cob_field       *f;
        int             i;
```

```
        va_list         args;

        if (!outorerr && !cob_screen_initialized) {
                fp = stdout;
        } else {
                fp = stderr;
        }
        va_start (args, varcnt);
        for (i = 0; i < varcnt; ++i) {
                f = va_arg (args, cob_field *);
                display (f, fp);
        }
        va_end (args);
        if (newline) {
                putc ('\n', fp);
                fflush (fp);
        }
}
```

7.48 mainpage.h File Reference

7.49 tarstamp.h File Reference

This graph shows which files directly or indirectly include this file:



Index