Published: 07/23/69

Identification

# Internal Representation of Declarations

R. Freiburghouse

## 1. Overview

This document describes the internal representation of PL/I declarations during compilation. It does not discuss the various intermediate steps which transform the original declarations into the form shown here. These intermediate states are internal to the declaration processor and are discussed in BZ.8.04. The form described here is the form of the declarations after the execution of the declaration processor. No further transformations are made on the declarations until the execution of the code generator.

The internal representation of declarations is a structure consisting of various kinds of components (nodes) which are linked to each other by pointers. The entire structure is known as the Symbol Table. It represents all source language declarations and all compiler produced declarations. The major types of nodes in this structure are briefly described below:

block nodes - represent the block structure of the program, they are created for each procedure, begin block, and ON unit in the source program. Each block node points to a list of executable statements and to symbol table nodes each of which represents a declaration made in that block. The block node is discussed more fully in BZ.8.09.

token table entries - each source program token (identifier, constant, or operator) is represented by an entry in this table.

symbol table nodes - each declaration is represented by a symbol table node which contains those attributes which are common to all classes of declarations.

attribute blocks - each symbol table node contains a pointer to an attribute block which provides attributes which are unique to a particular class of declarations. Separate attribute blocks exist for: variables, entry names, statement labels, and condition names.

constant blocks - each source program or compiler created constant is represented by a constant node. All constant nodes are threaded to form a uni-directional chain and are not connected to any block node.

Additional nodes are used to represent descriptors, initial values, and array attributes. Many nodes contain pointers to expressions which represent sizes or addressing offsets. All such expressions have the same representation as the source program expressions discussed in Section BZ.8.09.

The relationship between these nodes is shown in the example which follows. Note that the arrows represent pointers and also that the example is somewhat simplified to retain some measure of clarity.

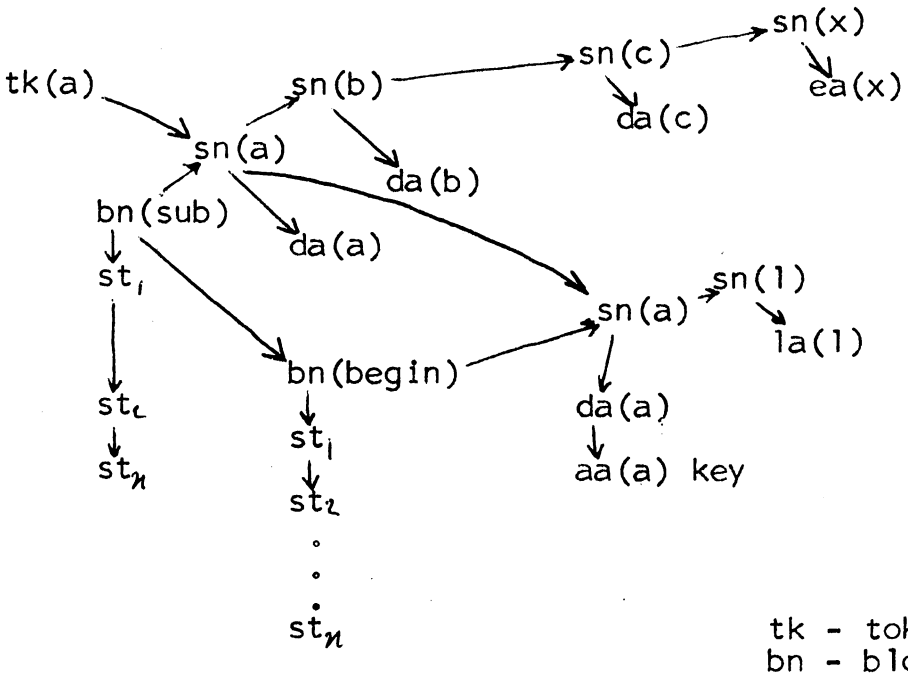
A Simplified Example of the Symbol Table Structure

```
sub:  proc;

      dcl (a,b,c) fixed, x entry;
       .
       .
       .
      begin;
       .
       .
       .
      dcl a(5);

       .
       .
       .
 1:   a(5) = a(3);
      end;

       .
       .
       .
      end;
```




tk - token table entry
bn - block node
sn - symbol table node
da - data attribute block
ea - entry name attribute block
la - lable attribute block
st - statement node
aa - array block

## 2. Implementation

### 2.1 The Token Table Entries

Each source program token (identifier, constant, or operator) is represented by an entry in this table. Each entry contains the token, its length expressed in characters, a pointer to a chain of context nodes, a pointer to a chain of symbol table nodes, and a pointer to another entry in the token table. All symbol table nodes which represent separate declarations of an identifier contain a pointer back to the appropriate token table entry.

A Definition of a Token Table Entry:

```
dcl  1  token_table      based(p),
     2  node_type        fixed bin(15),
     2  size             fixed bin(15),
     2  context          ptr,
     2  declaration      ptr,
     2  next             ptr,
     2  type             fixed bin(15),
     2  string           char(n);
```

node_type - is a constant 13 indicating that this is a token table entry.

size - is the length of the token in characters and is equivalent to n.

context - is a pointer used by the context recorder and context processor.

declaration - is a pointer to a symbol table node representing a declaration of the token.

next - is a pointer to another token table node.

type - is an integer code which describes the lexical type of the token. Its value is one of the codes listed in Appendix A.

string - is the acutal token.

## 2.2 Symbol Table Node

A symbol table node is created for each distinct use (declaration) of an identifier in the program. Each node contains only that information which is common to all declarations of all identifiers. Symbol table nodes also contain several pointers which point to other symbol table nodes and to an attribute block.

Definition of a Symbol Table Node:

```
dcl  1  symbol_table     based(p),
     2  node_type        fixed bin(15),
     2  dcl_type         fixed bin(15),
     2  block_node       ptr,
     2  reference_list   ptr,
     2  token            ptr,
     2  next             ptr,
     2  multi_use        ptr,
     2  attributes       ptr;
```

node type - is a constant 6 which indicates that this is a symbol table node.

dcl type - has one of the following values:

1. declare statement declaration
2. label constant or the entry name of an entry in this program
3. contextual declaration other than above
4. implicit declaration
5. compiler created declaration

block node - is a pointer to the block node which represents the block to which this declaration belongs.

reference list - is a pointer to a chain of cross reference nodes. Since this feature is not yet implemented the reference_list pointer must be null.

token - is a pointer to the token table entry for the identifier to which this declaration applies.

next - is a pointer to the next declaration which belongs to this block. The pointer is null if no further declarations exist in this block.

multi use - is a pointer to a symbol table node which represents another declaration of this same identifier. If no further multiple declarations exist this pointer is null.

attributes - is a pointer to the attribute block for this declaration.

## 2.3 The Data Attribute Block

All declarations of variables are represented by a symbol table node which contains a pointer to a data attribute block. The data attribute block contains information which is unique to declarations of variables.

Definition of a Data Atrribute Block:

```
dcl  1  data_attribute              based(p),
     2  node_type                   fixed bin(15),
     2  const_bit_size              fixed bin(15),
     2  statement                   fixed bin(31),
     2  level                       fixed bin(15),
     2  type                        fixed bin(15),
     2  class                       fixed bin(15),
     2  precision                   fixed bin(15),
     2  scale                       fixed bin(15),
     2  position                    fixed bin(15),
     2  size                        fixed bin(31),
     2  unit_size                   fixed bin(15),
     2  boundary                    fixed bin(15),
     2  allocation_units            fixed bin(15),
     2  const_storage               fixed bin(31),
     2  storage                     ptr,
     2  dcl_size                    ptr,
     2  back                        ptr,
     2  father                      ptr,
     2  brother                     ptr,
     2  son                         ptr,
     2  initial                     ptr,
     2  array                       ptr,
     2  reference                   ptr,
     2  equivalence                 ptr,
     2  descriptor                  ptr,
     2  escape                      ptr,
     2  bits,
     3  abnormal                    bit(1),
     3  packed                      bit(1),
     3  aligned                     bit(1),
     3  parameter                   bit(1),
     3  referenced                  bit(1),
     3  set                         bit(1),
     3  desc_image_reg              bit(1),
     3  refer_option                bit(1);
```

node_type - is a constant 8 which indicates that this is a data attribute block.

const_bit_size - If the data is of constant size, then this field contains the size measured in bits, otherwise the field is zero.

statement - is the identification number of the source statement from which this declaration was derived.

level - is the structure level.

type - is a code which describes the data type of the variable. Appendix B contains a list of the data types.

class - is a code which describes the storage class of the variable. Appendix C contains a list of the storage classes.

precision - is the arithmetic precision of the variable.

scale - is the arithmetic scale factor of the variable.

position - is used by the declaration processor and semantic translator. During declaration processing it contains the declared position for defined data. During semantic translation it contains the parameter position number.

size - contains the declared size of strings if that size was declared as a decimal integer constant.

unit size - is used by the declaration processor to remember the units in which the current size is expressed. If the current size is in bits the value is 1, if it is in characters the value is 2.

boundary - contains a code indicating the storage boundary alignment requirements of the variable. It may be one of the following codes:

1. bit
2. character
3. word
4. mod 2 word
5. mod 4 word
6. mod 8 word
7. mod 16 word

allocation units - is used by the declaration processor to remember the units in which the allocated size is expressed. If the allocated size is in bits the value is 1, if it is in characters the value is 2.

const storage - contains the amount of storage required by this variable. It is zero for variable size data and it is always in terms of words for level 1 variables.

storage - points to an expression which describes the amount of storage needed by this variable. The amount is always measured in words for level 1 variables and is null for constant size data.

dcl_size - points to an expression which is the declared length of a string or the declared size of an area. It is null if these values were declared as constant.

back - is a pointer to the symbol table node whch owns this attribute block.

father - is a pointer to the attribute block of the immediately containing structure. If the variable is not a member of a structure this pointer is null.

brother - is a pointer to the attribute block of the next structure member at this level. If the variable is not a member of a structure or if no more members exist at this level, this pointer is null.

son - is a pointer to the first element of a structure. If the variable is not a structure, this pointer is null.

initial - is a pointer to the internal representation of an initial value. If the call form of the initial attribute was used in the declaration, this points to an expression which describes the call. If the initial attribute was declared, this pointer points to an initial_link node as described in Section 2.3.1.

array - points to an array block if the variable was declared with dimensions, otherwise the pointer is null. The array block is described in Section 2.3.2.

reference - points to an accessing expression which describes the accessing function associated with this variable. If the variable is an array, the accessing function describes an access to the entire array. Accessing functions are described in Section 2.3.3.

equivalence - is a null pointer reserved for the implementation of the defined attribute.

descriptor - is a null pointer used by the code generator and storage allocator. If the desc_image_req bit is on, the storage allocator creates an argument descriptor image for this variable. The code generator uses that image when necessary.

escape - this is a general purpose pointer used by PL/I for three functions.

a) Label variables declared with a list of label values use this pointer to reference the list.

b) The structure created for varying strings uses this pointer to directly access the data attribute block of the string.

c) Declarations of offset variables which contain an area reference use this pointer to access the area.

abnormal - if on this bit indicates that the value of this variable may change without explicit indication of that change. The code generator and optimizer will not attempt to eliminate common subexpressions involving this variable.

packed - if on this bit indicates that the variable consists entirely of unaligned bit strings or entirely of unaligned character strings.

aligned - if on this bit indicates that the variable was declared aligned.

parameter - if on this bit indicates that the variable is a formal parameter.

referenced - if on this bit indicates that the variable has been referenced somewhere in the program. If this bit is not on, no storage will be allocated for the variable by the storage allocator.

set - reserved for future use.

desc image req - if on this bit indicates that the variable has been used as an argument to a function which requires descriptors. The bit causes the storage allocator to create a descriptor for the variable.

refer option - if on this bit indicates that the refer option was used in the declaration of this variable.

### 2.3.1 Definition of an Initial Link Node

```
dcl  1  initial_link     based(p),
     2  node_type        fixed bin(15),
     2  factor           ptr,
     2  value            ptr,
     2  next             ptr;
```

node_type - is a constant 11 which indicates that this is an initial link node.

factor - is a pointer to an expression which descirbes the number of times this value is to be used.

value - is a pointer to the internal representation of the initial value. A null pointer indicates no initialization. The pointer may point to a constant or to another initial_link_node.

next - is a pointer to the next initial link at this factoring level. A null pointer indicates that no more values exist at this level.

NOTE - All expressions must be constants if the storage class of the variable is static. The call option is not allowed for static variables.

2.3.2 Definition of an Array Block:

```
dcl  1 array_block                  based(p),
     2 node_type                    fixed bin(15),
     2 number_of_dimensions         fixed bin(15),
     2 dimensioned_ancestor         ptr,
     2 virtual_origin               ptr,
     2 bounds                       ptr,
     2 number_of_elements           ptr,
     2 const_number_of_elements     fixed bin(15),
     2 constant_virtual_origin      fixed bin(31),
     2 units                        fixed bin(15);
```

node_type - is a constant 19 which indicates that this is an array attribute block.

number_of_dimensions - is the number of dimensions declared for this variable.

dimensioned_ancestor - is a pointer to the data attribute block of the first dimensioned containing structure. If no such ancestors exist, this pointer is null.

virtual_origin - is a pointer to the expression

$$\sum_{j=1}^{n} l_j m_j$$

where n is the number of dimensions

lj is the jth lower bound
mj is the jth multiplier

The pointer is null if the expression is a constant.

bounds - is a pointer to a chain of bound pair nodes. Each dimension is represented by a bound pair node of the form descirbed in Section 2.3.2.1.

number of elements - is a pointer to an expression which describes the total size of the array prior to any rounding. This value may be smaller than the allocated size by some fraction of a word. This pointer is null if the value is a constant. This expression is used to initialize automatic or based arrary.

const number of elements - the same as the previous except the value is a constant.

constant virtual origin - same as the virtual origin except this value is a constant.

units - indicates the units of the multiplier. 1 indicates bits - other than 1 indicates words.

2.3.2.1 Definition of a Bound Pair

```
dcl  1  bound_pair                 based(p),
     2  lower_variable             ptr,
     2  upper_variable             ptr,
     2  variable_multiplier        ptr,
     2  next                       ptr,
     2  lower_constant             fixed bin(31),
     2  upper_constant             fixed bin(31),
     2  constant_multiplier        fixed bin(31);
```

lower variable - points to an expression which describes the lower bound. If the lower bound is constant, this pointer is null.

upper variable - points to an expression which describes the upper bound. If the upper bound is constant, this pointer is null.

variable multiplier - points to an expression which describes the multiplier associated with this dimension. This pointer is never null.

lower constant
upper constant
constant multiplier

these fields correspond to the variable fields previously described. If the bounds or multiplier are constant, their values are contained in these fields.

### 2.3.3 Accessing Functions

An accessing function consists of a reference node or string reference node possibly qualified by a pointer operator.

If the declared variable has a pointer qualified accessing function then the reference pointer of the data attribute block points to a pointer operator node which in turn points to a reference or string reference node.

The reference or string reference node is completely described in BZ.8.09. These two nodes contain size and offset expressions and constants used to address variables.

## 2.4 The Entry Attribute Block

All declarations of entry names are represented by a symbol table node which contains a pointer to an entry attribute block. The entry attribute block contains information which is unique to declarations of entries.

Definition of an Entry Attribute Block:

```
dcl  1  entry_attribute          based(p),
     2  node_type                fixed bin(15),
     2  entry_type               fixed bin(15),
     2  last_usage               fixed bin(31),
     2  location                 fixed bin(31),
     2  address                  ptr,
     2  returns                  ptr,
     2  list                     ptr,
     2  bits,
     3  external                 bit(1),
     3  desc_list_req            bit(1),
     3  referenced               bit(1),
     3  irreducible              bit(1),
     2  back                     ptr;
```

node type - is a constant 12 that indicates that this is an entry attribute block.

entry type - is a code which describes the type of entry name. The code may be any of the following values:

1. An entry name either belonging to this program or declared in this program.
2. An entry name parameter.
3. A generic entry name.

100 to 200 a builtin function name.

last_usage - must be zero. Used by the code generator.

location - must be zero. Used by the code generator.

address - points to the entry or procedure statement on which this name appeared. The pointer is null if the name is not an entry to this program.

returns - points to a symbol table node which describes the properties of the return value of this entry. The pointer may be null.

list - points to a chain of link nodes. Each link node consists of two pointers: the first points to a symbol table node, the second points to the next link in the chain. The symbol table nodes describe the properties of the parameters declared for this entry or they describe members of a generic family of entries.

external - if on this bit indicates that the entry is external. If off the bit indicates that the entry is internal.

desc_list_req - if on this bit indicates that this entry requires a descriptor list. If off the code generator will not create a descriptor list for calls to this entry.

referenced - if on this bit indicates that the entry is referenced somewhere in this program. External entries which are not referenced will not result in links.

irreducible - if on this bit indicates that the entry is irreducible. The bit is ignored by the compiler

back - points to the symbol table node which owns this attribute block.

## 2.5 The Label Attribute Block

All declarations of statement labels are represented by a symbol table node which contains a pointer to a label attribute block contains information whch is unique to declarations of statement labels. Note that label variables are represented as variables not as statement labels.

Definition of a Label Attribute Block:

```
dcl  1  label_attribute          based(p),
     2  node_type                fixed bin(15),
     2  label_type               fixed bin(15),
     2  last_usage               fixed bin(15),
     2  location                 fixed bin(31),
     2  address                  ptr,
     2  back                     ptr;
```

node type - is a constant 20 which indicates that this is a label attribute block.

label type - is a constant 1. Other values are reserved for format statements and future expansion.

last usage -
must be zero. Used by the code generator.

location -

address - points to the statement node on which the label was defined.

back - points to the symbol table node which owns this attribute block.

## 2.6 The Condition Attribute Block

All declarations of condition names are represented by a symbol table node which points to a condition attribute block.

Definition of a Condition Attribute Block:

```
dcl  1  cond_attribute          based(p),
     2  node_type               fixed bin(15),
     2  location                fixed bin(15),
     2  name                    ptr,
     2  enabled                 bit(1);
```

node type - is a constant 9 which indicates that this is a condition attribute block.

location - must be zero. Used by the code generator.

name - a pointer to a constant node which describes the character string representation of the condition name.

enabled - if on this bit indicates that this condition is enabled by an on statement somewhere in this program.

## 2.7 Constant Nodes

All arithmetic and string constants used by the program are represented by constant nodes. These nodes are connected into a chain whose origin is the external static pointer "constant_list".

Definition of a Constant Node:

```
dcl  1  constant_node                  based(p),
     2  node_type                      fixed bin(15),
     2  data_type                      fixed bin(15),
     2  size                           fixed bin(31),
     2  scale                          fixed bin(15),
     2  boundary                       fixed bin(15),
     2  class_offset                   fixed bin(31),
     2  value                          ptr,
     2  next                           ptr,
     2  last                           ptr
     2  equivalence                    ptr,
     2  last_usage                     fixed bin(31),
     2  location                       fixed bin(31);
```

node type - is a constant 16 which indicates that this is a constant node.

data type - one of the arithmetic or string data types given in Appendix B. Pointer and offset are also valid codes.

size - arithmetic precision or string length.

scale - arithmetic scale factor.

boundary - a code which describes the storage boundary alignment requirements of the constant.

1. bit
2. character
3. word
4. mod 2

class offset - the amount of storage, measured in words, required for this constant.

value - a pointer to the actual value of the constant.

next - a pointer to the next constant node.

last - unused.

equivalence - a pointer to another constant node. If this pointer is null, then the storage allocator will create a unique constant in the text segment. If this pointer points to another constant node, then that node's value has a binary representation which is equivalent to this constant. No storage is allocated for equivalenced constants.

last usage -
these fields are zero and are used by the code generator.

location -

# APPENDIX A

## LEXICAL TOKEN TYPES

| | |
|---|---|
| no_token | 0 |
| identifier | 1 |
| bit_string | 3 |
| char_string | 4 |
| float_bin | 5 |
| float_dec | 6 |
| bin_integer | 7 |
| dec_integer | 8 |
| isub | 10 |
| plus | 11 |
| minus | 12 |
| asterisk | 13 |
| expon | 14 |
| slash | 15 |
| not | 16 |
| eq | 17 |
| assignment | 17 |
| ne | 18 |
| lt | 19 |
| gt | 20 |
| le | 21 |
| ge | 22 |
| ngt | 23 |
| nlt | 24 |
| or | 25 |
| cat | 26 |
| and | 27 |
| colon | 28 |
| left_parn | 29 |
| right_parn | 30 |
| arrow | 31 |
| period | 32 |
| comma | 33 |
| semi_colon | 34 |
| i_dec_integer | 36 |
| i_bin_integer | 37 |
| i_float_bin | 38 |
| i_float_dec | 39 |
| i_fixed_dec | 40 |
| i_fixed_bin | 41 |
| fixed_bin | 42 |
| fixed_dec | 43 |

# APPENDIX B

## DATA TYPE CODES

| | | |
|---|---|---|
| rfb1 | 1 | real fixed binary single |
| rfb2 | 2 | real fixed binary double |
| rfd1 | 3 | real fixed decimal single |
| rfd2 | 4 | real fixed decimal double |
| rflb1 | 5 | real float binary single |
| rflb2 | 6 | real float binary double |
| rfld1 | 7 | real float decimal single |
| rfld2 | 8 | real float decimal double |
| | | |
| cfb1 | 21 | complex fixed binary single |
| cfb2 | 22 | complex fixed binary double |
| cfd1 | 23 | complex fixed decimal single |
| cfd2 | 24 | complex fixed decimal double |
| cflb1 | 25 | complex float binary single |
| cflb2 | 26 | complex float binary double |
| cfld1 | 27 | complex float decimal single |
| cfld2 | 28 | complex float decimal double |
| | | |
| cs | 31 | non-varying character string |
| vcs | 32 | varying character string |
| | | |
| bs | 41 | non-varying bit string |
| vbs | 42 | varying bit string |
| | | |
| lbv1 | 51 | label variable local values only |
| lbv | 52 | label variable any value |
| | | |
| ptr | 60 | pointer variable |
| off | 61 | offset variable |
| | | |
| entvar | 72 | entry variable |
| | | |
| struct | 80 | structure |
| v_struct | 81 | structure created for var strings |
| cell | 82 | cell data type |
| filen | 83 | file name |
| area | 84 | area data |

# APPENDIX C

## STORAGE CLASS CODES

| | |
|---|---|
| auto | 1 |
| auto_adj | 2 |
| based | 3 |
| static_int | 4 |
| static_ext | 5 |
| ctl_int | 6 |
| ctl_ext | 7 |
| param | 8 |
| def | 9 |
| temp | 10 |
| stack_header | 11 |
| text_ref | 12 |
| link_ref | 13 |
| ctl_param | 14 |