

Space Flight Operations Contract

HAL/S-FC

SDL INTERFACE CONTROL DOCUMENT

September 2005

DRD- 1.4.3.8-a

Contract NAS9-20000



HAL/S-FC / SDL
INTERFACE CONTROL DOCUMENT

Prepared by

Peter Koester
USA/Application Tools, PASS Build and
Reconfiguration

Approved by

Monica Leone, Director
USA/Application Tools, PASS Build and
Reconfiguration

DRD – 1.4.3.8-a

Contract NAS9-18817

REVISION LOG

Rev. letter	Change no.	Description	Date
Baseline		Total rewrite. Supercedes OB30029 due to SSCR 14217	09/2005

LIST OF EFFECTIVE PAGES

The status of all pages in this document is shown below:

<u>Page No.</u>	<u>Change No.</u>
32.0/17.0	Baseline

PREFACE

The HAL/S-FC/SDL Interface Control Document was prepared by the United Space Alliance (USA), Flight Operations.

The primary responsibility is with USA, FSW Applications Tools and Recon, D/0163500.

Questions concerning the technical content of this document should be directed to Danny Strauss, (281) 282-2647, Mailcode USH-635L, Department 01635A7.

“This page intentionally left blank.”

CONTENTS

Section	Page
1.0 HAL/S-FC / SDL¹ ICD.....	1-1
1.1 INTRODUCTION.....	1-1
1.1.1 Purpose.....	1-1
1.1.2 Scope.....	1-1
1.1.3 Precedence of Documents.....	1-1
2.0 COMPILER/SPF.....	2-1
2.1 PROGRAM MANAGEMENT FACILITY (PMF).....	2-1
2.1.1 Dynamic Invocation of HAL/S-FC Compiler.....	2-3
2.1.2 Compile-Time Options.....	2-5
2.1.3 Inputs.....	2-5
2.1.3.1 Primary Input.....	2-5
2.1.3.2 Included Input.....	2-6
2.1.4 Outputs.....	2-7
2.1.4.1 Listings.....	2-7
2.1.4.2 Object Code.....	2-8
2.1.4.3 Templates.....	2-8
2.1.4.4 Simulation Data Files (SDFs).....	2-8
2.1.4.5 Return Codes.....	2-8
2.1.5 Access Rights.....	2-9
2.2 SIMULATION DATA FILES (SDFS).....	2-10
2.2.1 Simulation Data File Directory.....	2-14
2.2.2 Master Directory Cell.....	2-15
2.2.2.1 SDF Free Space.....	2-15
2.2.2.2 Directory Root Cell.....	2-17
2.2.2.2.1 Compiler Data.....	2-27
2.2.2.2.1.1 Title Data Cell.....	2-27
2.2.2.2.1.2 CARDTYPE Data Cell.....	2-28
2.2.2.2.1.3 Initialization Table.....	2-29
2.2.2.2.2 Include Text Data.....	2-30
2.2.2.2.3 Block Data Structures.....	2-33
2.2.2.2.3.1 Block Index Table.....	2-33
2.2.2.2.3.2 HAL/S Block Data Cell.....	2-34
2.2.2.2.3.3 Block Symbol Extent Cell.....	2-43
2.2.2.2.4 Symbol Data Structures.....	2-46
2.2.2.2.4.1 Symbol Index Table.....	2-46
2.2.2.2.4.2 Symbol Data Cell.....	2-48
2.2.2.2.4.3 Constant Value Cells.....	2-69
2.2.2.2.4.3.1 String Constant Value Cells.....	2-70
2.2.2.2.4.3.2 Scalar/Integer Constant Value Cells.....	2-71
2.2.2.2.4.4 Replace Text Cells.....	2-71

2.2.2.2.4.5	Procedure/Function Formal Parameter Cell.....	2-74
2.2.2.2.4.6	Name Terminal Initialization Cell.....	2-76
2.2.2.2.5	Statement Data Structures.....	2-82
2.2.2.2.5.1	Statement Index Table.....	2-84
2.2.2.2.5.2	Statement Data Cells.....	2-86
2.2.2.2.5.2.1	Executable Statement Data Cell.....	2-86
2.2.2.2.5.2.2	DECLARE Statement Data Cell.....	2-93
2.2.2.2.5.3	Statement Extent Cell.....	2-94
2.2.2.2.5.4	Procedure/Function Invocation Cell.....	2-98
2.2.2.2.6	Expression Variables Cell.....	2-100
2.2.2.2.7	Variable Reference Cell.....	2-103
2.2.2.2.8	Function Tables.....	2-111
2.2.2.2.8.1	Function Index Tables.....	2-112
2.2.2.2.8.2	Function XREF Data Cell.....	2-115
2.2.2.2.9	HALMAT Data Structures.....	2-116
2.2.2.2.9.1	HALMAT Cells.....	2-116
2.2.2.2.9.2	Literal Data.....	2-119
2.2.2.2.9.2.1	Literal Extent Table.....	2-120
2.2.2.2.9.2.2	Literal Tables.....	2-121
2.2.2.2.9.2.2.1	Character Literal.....	2-122
2.2.2.2.9.2.2.2	Arithmetic Literal.....	2-123
2.2.2.2.9.2.2.3	Bit Literal.....	2-124
2.2.2.2.9.2.2.4	Template Subscript Literal Cell.....	2-125
2.3	OBJECT CODE.....	2-127
3.0	AP-101 EXECUTION ENVIRONMENT.....	3-1
3.1	AP-101 REGISTER USE.....	3-1
3.2	HAL/S STACK.....	3-2
3.3	STACK AND LOCAL BLOCK DATA ORGANIZATION.....	3-5
3.4	PROCEDURE AND FUNCTION CALLS.....	3-7
4.0	CSECT/MEMBER NAMING CONVENTIONS.....	4-1

Appendix

APPENDIX A EXAMPLE PROGRAM AND SDF DATA STRUCTURES...A-1

APPENDIX B CHANGE HISTORY.....B-1

FIGURES

Figure

Figure 2-1 HAL/S-FC Interface With Program Libraries.....	2-2
Figure 2-2 Partitioned Data Set Directory Entry.....	2-7
Figure 2-3 Naming Convention Cross-Reference Table.....	2-11
Figure 2-4 SDF Pointer.....	2-12
Figure 2-5 Simulation Data File Member Organization (Not all interconnections are shown).....	2-13
Figure 2-6 PDS-Level Organization of the Simulation Data Files.....	2-14
Figure 2-7 Master Directory Cell.....	2-15
Figure 2-8 Free Cell Linkage.....	2-16
Figure 2-9 SDF Free Cell Linked Lists.....	2-17
Figure 2-10 Master Directory/Directory Root Cell Overview.....	2-18
Figure 2-11 Directory Root Cell (Part 1 of 3).....	2-19
Figure 2-12 Title Data Cell Overview.....	2-27
Figure 2-13 Title Data Cell.....	2-28
Figure 2-14 Cardtype Data Cell Overview.....	2-28
Figure 2-15 CARDTYPE Data Cell.....	2-29
Figure 2-16 Initialization Table Overview.....	2-30
Figure 2-17 Initialization Table.....	2-30
Figure 2-18 Include Data Overview.....	2-31
Figure 2-19 Include Data Cell.....	2-31
Figure 2-20 Block Data Structures Overview.....	2-33
Figure 2-21 Block Index Table.....	2-34
Figure 2-22 All Symbols Contained on One SDF Page for Block.....	2-35
Figure 2-23 Symbols Contained on Multiple SDF Pages for Block.....	2-36
Figure 2-24 Example of Block Symbol Extent Cell.....	2-37
Figure 2-25 Block Data Cell.....	2-38
Figure 2-26 Alphabetic Name Tree.....	2-40
Figure 2-27 Hierarchical Block Tree.....	2-41
Figure 2-28 Relationship of Block Data Cells, Block Symbol Extent Cells, and Symbol Index Table.....	2-44
Figure 2-29 Block Symbol Extent Cell.....	2-45
Figure 2-30 Symbol Data Structures Overview.....	2-46
Figure 2-31 Symbol Index Table.....	2-47
Figure 2-32 Symbol Data Cell (Part 1 of 3).....	2-49
Figure 2-33 Structures and Templates for a Single Structure (Part 1 of 2)	2-52
Figure 2-34 Structures and Templates for Nested Structures (Part 1 of 2)	2-54
Figure 2-35 Structure Symbol Cross-Reference Information (Part 1 of 2).	2-56
Figure 2-36 Symbol Data Cell Linked Lists.....	2-57
Figure 2-37 Stack Variable Character String Format.....	2-66

Figure 2-38 Array of Character Strings.....	2-67
Figure 2-39 Algorithm for Calculating the Bias Factor.....	2-68
Figure 2-40 Constant Value Cell Overview.....	2-70
Figure 2-41 String Constant Value Cell.....	2-70
Figure 2-42 Scalar/Integer Constant Value Cell.....	2-71
Figure 2-43 Replace Text Overview.....	2-72
Figure 2-44 Replace Text Examples.....	2-72
Figure 2-45 Replace Text Parameter Cell.....	2-73
Figure 2-46 Replace Text Parameter Cell Pseudo Descriptor.....	2-73
Figure 2-47 Replace Text Macro Cell.....	2-74
Figure 2-48 Procedure/Function Formal Parameter Cell Override.....	2-75
Figure 2-49 Procedure/Function Formal Parameter Cell.....	2-76
Figure 2-50 Name Terminal Initialization Cell Overview.....	2-77
Figure 2-51 Name Terminal Initialization Cell.....	2-78
Figure 2-52 Initial Pointer Value Operator.....	2-79
Figure 2-53 Initialization Loop Start Operator.....	2-80
Figure 2-54 Initialization Loop End Operator.....	2-81
Figure 2-55 End of Initialization (Cell) Operator.....	2-81
Figure 2-56 Name Terminal Initialization Extension Cell.....	2-82
Figure 2-57 Statement Data Structures Overview.....	2-83
Figure 2-58 Statement/Symbol Relationship Overview.....	2-84
Figure 2-59 Statement Index Table.....	2-85
Figure 2-60 Example of Non-unique SRNs.....	2-86
Figure 2-61 Block Statement Nesting.....	2-86
Figure 2-62 Executable Statement Data Cell.....	2-87
Figure 2-63 Statement Type.....	2-91
Figure 2-64 Left Hand Side (LHS) Indexes.....	2-92
Figure 2-65 DECLARE Statement Data Cell.....	2-93
Figure 2-66 Statement Extent Cell Overview.....	2-95
Figure 2-67 Relationship of Statement Extent Cells and Statement Index Table.....	2-96
Figure 2-68 Statement Extent Cell.....	2-97
Figure 2-69 Procedure/Function Invocation Cell Overview.....	2-98
Figure 2-70 Procedure/Function Invocation Cell.....	2-99
Figure 2-71 Expression Variables Cell Overview.....	2-101
Figure 2-72 Expression Variables Cell.....	2-102
Figure 2-73 Variable Reference Cell Overview (Expression Variables Cell).....	2-104
Figure 2-74 Variable Reference Cell Overview (Name Terminal Initialization Cell).....	2-105
Figure 2-75 Variable Reference Cell Overview (Symbol Data Cell).....	2-106
Figure 2-76 Variable Reference Cell.....	2-107
Figure 2-77 Structure Reference Diagram.....	2-108
Figure 2-78 Function Data Overview.....	2-112
Figure 2-79 Function Index Table.....	2-113
Figure 2-80 Function XREF Data Cell.....	2-115

Figure 2-81 Function XREF Extension Cell.....	2-115
Figure 2-82 HALMAT Data Cells Overview.....	2-117
Figure 2-83 HALMAT Cell.....	2-118
Figure 2-84 HALMAT Extension Cell.....	2-118
Figure 2-85 Literal Data Overview.....	2-120
Figure 2-86 Literal Extent Table.....	2-121
Figure 2-87 Literal Table.....	2-122
Figure 2-88 Character Literal Cell.....	2-123
Figure 2-89 Arithmetic Literal Cell.....	2-124
Figure 2-90 Bit Literal Cell.....	2-125
Figure 2-91 Template Subscript Literal Cell.....	2-126
Figure 2-92 ESD Output Record (Card Image).....	2-127
Figure 2-93 ESD Data Item.....	2-128
Figure 2-94 Text Output Record (Card Image).....	2-129
Figure 2-95 RLD Output Record (Card Image).....	2-130
Figure 2-96 END Output Record - Type 1 (Card Image).....	2-131
Figure 2-97 END Output Record - Type 2 (Card Image).....	2-131
Figure 2-98 IDR Data in a Object Module END Record.....	2-132
Figure 2-99 TESTRAN (SYM) Output Record - (Card Image).....	2-132
Figure 2-100 SYM Variable Field Data.....	2-133
Figure 3-1 Stack Elements.....	3-3
Figure 3-2 Stack Organization Cell.....	3-5
Figure 3-3 Local Block Data.....	3-6
Figure 3-4 Error Vector.....	3-8

“This page intentionally left blank.”

1.0 HAL/S-FC / SDL¹ ICD

1.1 INTRODUCTION

1.1.1 Purpose

The purpose of the HAL/S-FC / SDL Interface Control Document (ICD) is to define the specific interfaces that exist between the HAL/S-FC compiler and Software Production Facility (SPF) software systems. This document is necessary to control and track changes in the interfaces since parallel HAL/S-FC and SPF maintenance efforts are taking place. Its contents impose requirements on the HAL/S-FC compilers.

1.1.2 Scope

The scope of this document covers the following two major HAL/S-FC / SPF interface areas:

- HAL/S-FC compiler with the SPF
- HAL/S-FC compiler with the AP-101/S Linkage Editor

1.1.3 Precedence of Documents

The precedence governing the applicability of various controlling documents is as follows:

HAL/S Language Specification (USA003088)
HAL/S-FC Compiler System Specification (USA003089)
HAL/FCOS Interface Control Document (USA001460)
HAL/S-FC / SDL Interface Control Document (USA001556)

1. Since this document was originally written, the term Software Development Laboratory (SDL) has been superceded by the term Software Production Facility (SPF). SPF will be used throughout this document, except for the document Title.

“This page intentionally left blank.”

2.0 COMPILER/SPF

This portion of the ICD defines the interfaces that exist between the HAL/S-FC compiler and the SPF. It contains the following major subsections:

- 2.1 PROGRAM MANAGEMENT FACILITY (PMF), page 1
- 2.2 SIMULATION DATA FILES (SDFs), page 10
- 2.3 OBJECT CODE, page 129

2.1 PROGRAM MANAGEMENT FACILITY (PMF)

The system that maintains and controls the disk resident libraries (source, object, and load) for the SPF is called the PMF. The interface considerations between the PMF and the HAL/S-FC compiler arise due to the following factors:

- Dynamic invocation of the HAL/S-FC compiler by the PMF
- HAL/S-FC compiler's need to access and/or create elements within the program libraries.

Figure 2-1 on page 2 shows the relationship between the program libraries and the HAL/S compiler.

In the sections that follow, the detailed interface between the PMF and the HAL/S-FC compiler is established. Section 2.1.1, "Dynamic Invocation of HAL/S-FC Compiler" on page 3 discusses dynamic invocation of the HAL/S-FC compiler. Section 2.1.2, "Compile-Time Options" on page 5 discusses Compile-Time options. Section 2.1.3, "Inputs" on page 5 discusses the source input and how it is formatted. Section 2.1.4, "Outputs" on page 7 discusses the output from the compiler and how it is formatted. Section 2.1.5, "Access Rights" on page 9 discusses access rights.

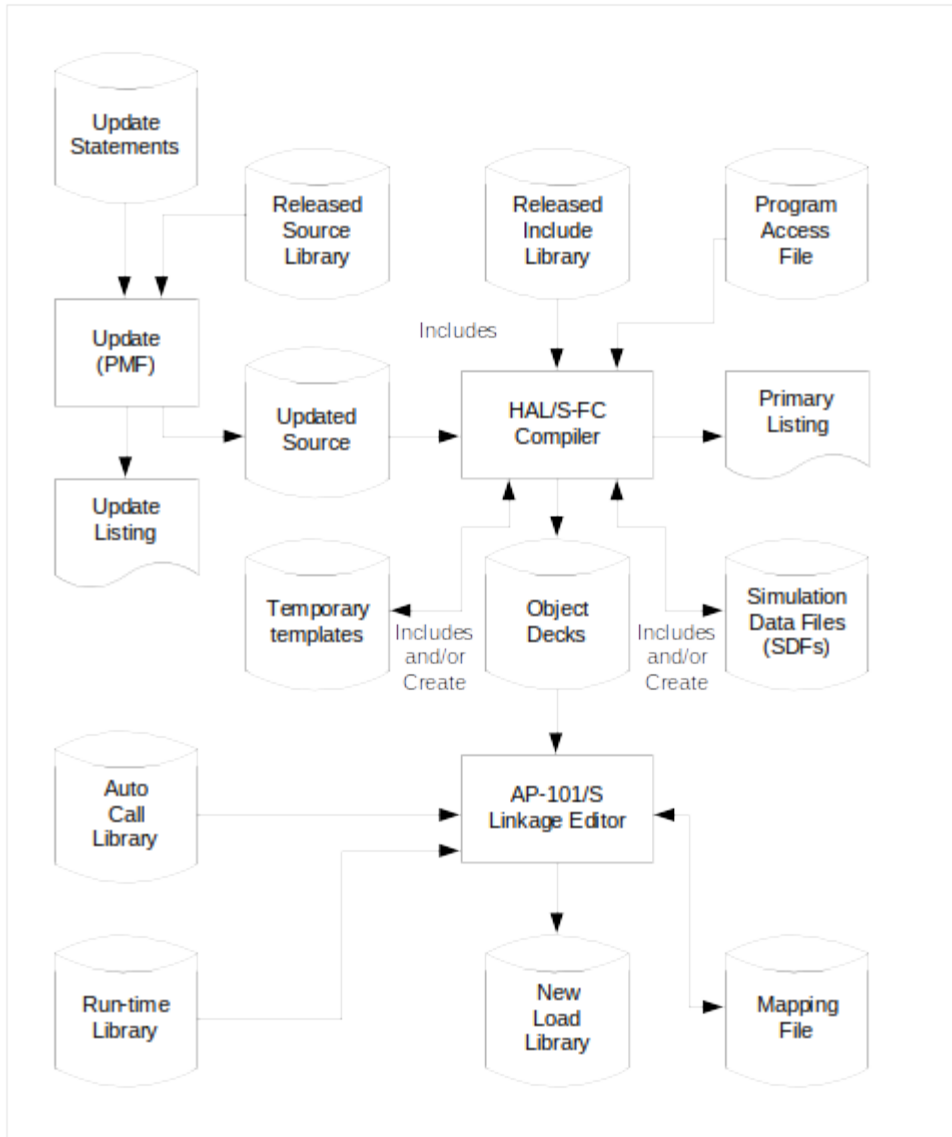


Figure 2-1 HAL/S-FC Interface With Program Libraries

2.1.1 Dynamic Invocation of HAL/S-FC Compiler

The HAL/S compiler can be invoked by the problem program at execution time through the use of the CALL, LINK, XCTL, or ATTACH macro instructions. If the XCTL macro instruction is used to invoke the compiler, then no user options may be specified. The compiler will use the standard default, as set during system generation, for each option.

If the compiler is invoked by CALL, LINK, or ATTACH, the user may supply:

1. The compiler options
2. The DDNAMES of the data sets to be used during processing
3. Field for the compiler to return the control section (CSECT) name generated for this unit of compilation

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
symbol	CALL	MONITOR, (optionlist {[, ddnamelist] [, ddnamelist, csectname]}), VL
	LINK OR ATTACH	EP=MONITOR, PARAM=(optionlist {[, ddnamelist] [, ddnamelist, csectname]}), VL= 1

EP - specifies the symbolic name of the compiler. The entry point at which execution is to begin is determined by the control program (from the library directory entry).

PARAM - specifies, as a sublist, the address parameters to be passed from the problem program to the compiler. The first word in the address parameter list contains the address of the option list. The second word contains the address of the DDNAME list. The third word contains the address of the field used by the compiler to return the control section (CSECT) name.

optionlist - specifies the address of a variable length list containing the options. This address must be provided even if no option list is provided.

The option list must begin on a halfword boundary. The first two bytes contain a count of the number of bytes in the remainder of the list. If no options are specified, the count must be zero. The option list is free form with each field separated by a comma.

ddnamelist - specifies the address of a variable length list containing alternate DDNAMEs for the data sets used during compiler processing. If standard DDNAMEs are used and the CSECT name return field is not provided, then this operand may be omitted. If standard DDNAMEs are used, but the CSECT name return field is provided, this address must be provided and point to halfword count of zero.

When the standard DDNAMEs are not to be used, the alternate DDNAME list must begin on a halfword boundary. The first two bytes contain a count of the number of bytes in the remainder of the list. If any name is less than eight bytes long, it must be left-justified and padded with blanks. If an alternate DDNAME is omitted, the standard name will be assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end merely by shortening the list. The sequence of the 8-byte entries in the DDNAME list is as follows:

<u>Entry</u>	<u>Alternate Name For</u>	<u>Data Set Organ.</u>	<u>Description</u>
1	SYSIN	PS	Primary Input
2	INCLUDE	PO	Include Library
3	ERROR	PO	Error Messages
4	ACCESS	PO	Program Access File
5	SYSPRINT	PS	Primary Listing
6	LISTING2	PS	Secondary Listing
7	OUTPUT3	PS	Object Deck
8	OUTPUT4	PS	Duplicate Object Deck
9	OUTPUT5	PO	Simulation Data Files (SDFs)
10	OUTPUT6	PO	Templates
11	OUTPUT7	PS	AP-101 Assembly Listing
12	FILE1	PS	<u>Work File</u>
13	FILE2	PS	↓
14	FILE3	PS	↓
15	FILE4	PS	↓
16	FILE5	PS	↓
17	FILE6	PS	↓
18	PROGRAM	PS	Compiler Program Library
19	OUTPUT8	PO	Define Block Library
20	HALSDF	PO	Simulation Data Files (SDFs)
21	UNUSED	PO	
22	UNUSED	PS	
23	UNUSED	PS	
24	FILE7	PS	Work File
25	UNUSED	PO	
26	UNUSED	PO	

Where PS = Physical Sequential

PO = Partitioned Organization

csectname - specifies the address of an eight byte field into which the compiler moves the generated control section (CSECT) name of the primary unit of compilation. If the program invoking the compiler does not need this information, this operand may be omitted.

VL - specifies that the sign bit is to be set to 1 in the last word of the address parameter list.

2.1.2 Compile-Time Options

The Compile-Time Options, including special compiler processing for the SPF, are listed in Section 5.1 of the HALS/S-FC User's Manual.

An example of special compiler processing for the SPF is:

- Special output requirements on compiler's primary listing (see Section 2.1.4.1, "Listings" on page 7).

2.1.3 Inputs

Source data will come from two major areas. The primary area contains units of compilation (PROGRAM, PROCEDURE, or COMPOOL) that are passed from an update step. The other area is the HALSDF library which contains simulation data files (SDFs) and the INCLUDE library which contains source code that are to be included. Both the primary input and INCLUDE library may contain concatenated data sets. If data sets are concatenated, they must have identical characteristics.

2.1.3.1 Primary Input

The following items relate to the format of the source data coming in via the primary input stream:

Data Set Organization (DSORG): Sequential

Record Format (RECFM): Fixed Blocked (FB)

Logical Record Length (LRECL): 80

Record Sequence Number: Positions 73 thru 78 of every record are assigned and controlled by PMF

Record Revision Level: Positions 79 and 80 of every record are assigned and controlled by PMF

The source margins on the input records are positions 2 thru 72 of each record. Positions 73-80 are valid for source data if the NOSRN option is specified and PMF is not used to compile the source.

2.1.3.2 Included Input

The following items relate to the format of the data coming in via the HALSDF library input stream:

Data Set Organization (DSORG): Partitioned

Record Format (RECFM): Fixed (F)

Logical Record Length (LRECL): 1680

The following items relate to the format of the data coming in via the INCLUDE library input stream:

Data Set Organization (DSORG): Partitioned

Record Format (RECFM): Fixed Block (FB)

Logical Record Length (LRECL): 80

When the compiler is retrieving a data set member to be included in the source, the source revision level is obtained from the PDS directory entry and used on the output listing (see Section 2.1.4, "Outputs" on page 7). The location of the 2 byte revision level is shown in Figure 2-2 on page 7.

In the optional user data portion of a PDS directory entry, any user supplied pointers (TTRNs) must come first.

Bits 1 and 2 of the "C" byte specifies the number of TTRNs that are present. Each TTRN is 4 bytes long. The source revision level will follow immediately after the last TTRN. If no TTRNs exist, the source revision level will follow immediately after the "C" byte.

Note: If no, revision level field exists, (i.e., PMF was not used to update the included input) assume the revision level is zero.

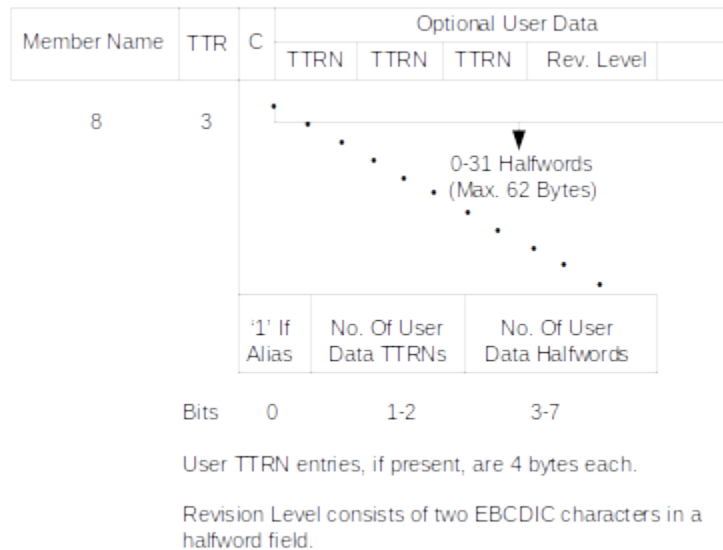


Figure 2-2 Partitioned Data Set Directory Entry

2.1.4 Outputs

Outputs from the compiler have been categorized into the following five classes:

- Listings (Primary, Secondary – Not Used, and Tertiary)
- Object Code
- Templates
- Simulation Data Files
- Return Codes

2.1.4.1 Listings

The primary listing is the standard HAL/S compiler output listing as described in the HAL/S-FC User's Manual (USA003090) with the following modifications:

The Statement Reference Number (SRN) (input positions 73 thru 78) and source record revision level (input positions 79 and 80) are printed adjacent to the compiler's statement number.

Note: If the statement spans more than one input record, the compiler only prints the statement reference number and record revision level from the first record.

After encountering an "INCLUDE" statement, the compiler shall print informative messages. Message content depends on the source of inclusion. Inclusion from an SDF results in the following:

"INCLUDED FROM SDF member"
"RVL xx CATENATION NUMBER n"

Local inclusion and inclusion from a file that is not an SDF results in:

“START OF INCLUDED MEMBER, RVL xx, CATENATION NUMBER n”.

“END OF INCLUDED MEMBER, RVL xx, CATENATION NUMBER n”.

Where xx is alphanumeric and n is numeric. If the LIST option is on, the “INCLUDED” source statements are printed between the appropriate message sets described above.

No requirements have been identified for the secondary listing from the HAL/S compiler since the type of listing needed (i.e., an accurate reflection of the source records) is available from the PMF program.

The tertiary listing consists of AP-101 code that is emitted by the HAL/S-FC compiler.

2.1.4.2 Object Code

When producing AP101/S object modules, each object module generated by the HAL/S compiler is written to a sequential data set for input to the AP101/S linkage editor. Multiple compilations produce object decks “stacked” in the order they were compiled.

2.1.4.3 Templates

NOTE: The PMF default action is to specify that templates are not generated.

Each template generated by the HAL/S-FC compiler is output to a partitioned data set defined by the OUTPUT6 DD card. The member name is derived by eliminating underscore characters from the source (“unit of compilation”) label, taking the first six characters (or all of the characters, if there are fewer than six characters) from the resulting string, and then appending two “@” characters to the beginning of the string. For example, a compilation unit named MY_PROGRAM generates a Template member named @@MYPROG.

Since the templates go into a dataset which may also contain source code members, the templates are created in the same format as source code (see Section 2.1.3.2, “Included Input” on page 6), and written using the block size of the existing data set.

All template directory entries, for new or revised templates, are created with two bytes of user data initialized to X'F0F0'.

2.1.4.4 Simulation Data Files (SDFs)

Each SDF member created by the HAL/S compiler is written to a partitioned data set defined by the OUTPUT5 DD card. The member name is derived as described in Section 2.1.4.3, “Templates” on page 8, except that it is preceded by two “#” characters (e.g., MY_PROGM becomes ##MYPROG).

2.1.4.5 Return Codes

The compiler passes the results of the compilation process via register 15. The low order three bytes of register 15 contain the highest severity code encountered during compilation. The high order byte of register 15 is used as a flag byte with the following bit settings defined:

- 1 This unit of compilation has a template
- .1 Template for this unit of compilation was either changed or newly created
- . .1 This unit of compilation has an SDF

2.1.5 Access Rights

The HAL/S language allows managerial restrictions to be placed upon the usage of user-defined variables and external routines. The existence of such a restriction is indicated by the use of the ACCESS attribute as described in the HAL/S Language Specification (USA003088). A detailed description of the manner in which these restrictions are enforced can be found in HAL/S Compiler System Specification (USA003089). For additional information, refer to the HAL/S-FC User's Manual (USA003090).

2.2 SIMULATION DATA FILES (SDFS)

Simulation Data Files (SDFs) provide the information about symbols and statements necessary to conduct simulation processes and to reduce simulation output into a convenient and readable form. An SDF member is produced by the compiler for each unit of compilation, including COMPOOLS. It is stored, as a member of a PDS, separate from the associated object code, and therefore can be retrieved as needed by the simulation processors and DASS/HALSTAT tools. SDFs are also used by the HAL/S Compiler to retrieve included COMPOOL symbol data. Naming conventions are described in Section 2.1.4.4, "Simulation Data Files (SDFs)" on page 8.

Several Tables/Cells have been renamed in this document in an effort to standardize the names of the Tables/Cells between the HAL/SDL ICD, the HAL/S-360 Compiler System Specification, and the documentation to SDFPKG. Figure 2-3 on page 11 contains a cross-reference between the old names and the new standard ones.

The logical organization of an SDF member for a unit of compilation is portrayed in Figure 2-5 on page 13. The SDF member is logically divided into three major parts as follows:

Directory - which provides the locations of the various component parts of the SDF member.

Symbol Data - which provides attribute information about the symbols in the compilation. Also it supplies information on relative memory locations of symbols, structure template linkages for structure elements, and the statements in which symbols are declared, referenced, used as a subscript, or modified.

Statement Data - which provides attribute information about the statements in a compilation. It also provides information on relative memory locations of the first and last machine instructions in a statement, statement labels, and the variables that are used and/or modified.

HAL/S Compiler Specification and SDFPKG Terminology	HAL/SDL ICD Terminology (Revision 8)	Standardized Table/Cell Names	HAL/SDL ICD Figure No.
Directory Root Cell	Simulation Table or Directory Header	Master Directory Cell	Section 2.2.2 (Figure 2-7)
Directory Root Cell	Simulation Table or Directory Header	Directory Root Cell	Section 2.2.2.2 (Figure 2-11)
Block Data Cell	HAL/S Block List Member	Block Data Cell	Section 2.2.2.2.3.2 (Figure 2-23)
Symbol Data Cell	Symbol Data Entry	Symbol Data Cell	Section 2.2.2.2.4.2 (Figure 2-30)
Statement Data Cell (Executable)	Statement Data Entry	Executable Statement Data Cell	Section 2.2.2.2.5.2.1 (Figure 2-60)
Statement Data Cell (Declare)	Statement Data Entry	Declare Statement Data Cell	Section 2.2.2.2.5.2.2 (Figure 2-63)
Block Node	Block Index Table Entry	Block Index Table Entry	Section 2.2.2.2.3.1 (Figure 2-19)
Symbol Node	Symbol Names and Pointers Table Entry	Symbol Index Table Entry	Section 2.2.2.2.4.1 (Figure 2-29)
Symbol Block Extent Cell	Symbol Block Extent Cell	Block Symbol Extent Cell	Section 2.2.2.2.3.3 (Figure 2-27)
Function Node Table	Function Node Table	Function Index Table	Section 2.2.2.2.8.1 (Figure 2-77)
Function XREF Cell	Function XREF Cell	Function XREF Data Cell	Section 2.2.2.2.8.2 (Figure 2-78)
_____	Card Type Cell	CARDTYPE Data Cell	Section 2.2.2.2.1.2 (Figure 2-15)
_____	Include Library Member Cell	Include Data Cell	Section 2.2.2.2.2 (Figure 2-17)
_____	Replace Text Cell	Replace Text Parameter Cell	Section 2.2.2.2.4.4 (Figure 2-43)
_____	Replace Text (Extension) Cell	Replace Text Macro Cell	Section 2.2.2.2.4.4 (Figure 2-45)

Figure 2-3 Naming Convention Cross-Reference Table

Figure 2-5 on page 13 identifies the various components of the SDF member and depicts most of the interconnections between these components. One of the relationships not shown is the connection between the Statement Extent Cells and Block Symbol Extent Cells and their corresponding Index Tables. This connection is too complex to portray; therefore the user should reference the sections for the Statement Extent Cells and Block Symbol Extent Cells to gain an understanding of the connection. The figure should be referred to as later sections are reviewed in order to keep in mind the relationships of the various components of the SDF member.

The SDF member for a unit of compilation is blocked into fixed-length physical records in the SDF partitioned data set (PDS). The organization of these records and of the PDS directory entry for the member is shown in Figure 2-6 on page 14. Each physical record (“page”) is 1680 bytes long and contains logical records dedicated to specific functions. The physical records are numbered from zero.

PMF places the revision level of the HAL/S source code member into the first two bytes of the user-data field of the directory entry. Figure 2-6 on page 14 illustrates the storage of the revision level for an SDF member.

The logical data segments are fixed or variable in size, but are always fullword aligned. A logical data segment is referenced by a 4-byte pointer (fullword aligned) which consists of two 2-byte fields; the first field contains the record number (beginning at 0) of the physical record to which the logical record belongs, and the second field contains the offset (also beginning at 0) of the logical record within the physical record (see Figure 2-4 on page 12 for a pointer illustration). It should be noted that the record number is synonymous with the page number. A pointer is represented in the ICD figures by a vertical arrow “↑” and a notational reference.

All of the data contained within an SDF member is organized into Cells and Tables. A Cell is a contiguous block of space in a record of an SDF member that is addressable only by an SDF pointer. A Cell cannot cross a page boundary so it must be ≤ 1680 bytes in length. Tables, on the other hand, consist of multiple entries and may cross SDF page boundaries, though no individual table entry can cross a boundary. Table entries may be accessed by either SDF pointers or by halfword indexes.

Table indexes are two bytes in size and are always halfword aligned. The first index of any table is always one.

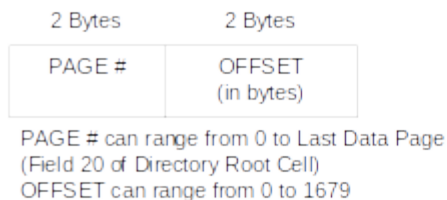


Figure 2-4 SDF Pointer

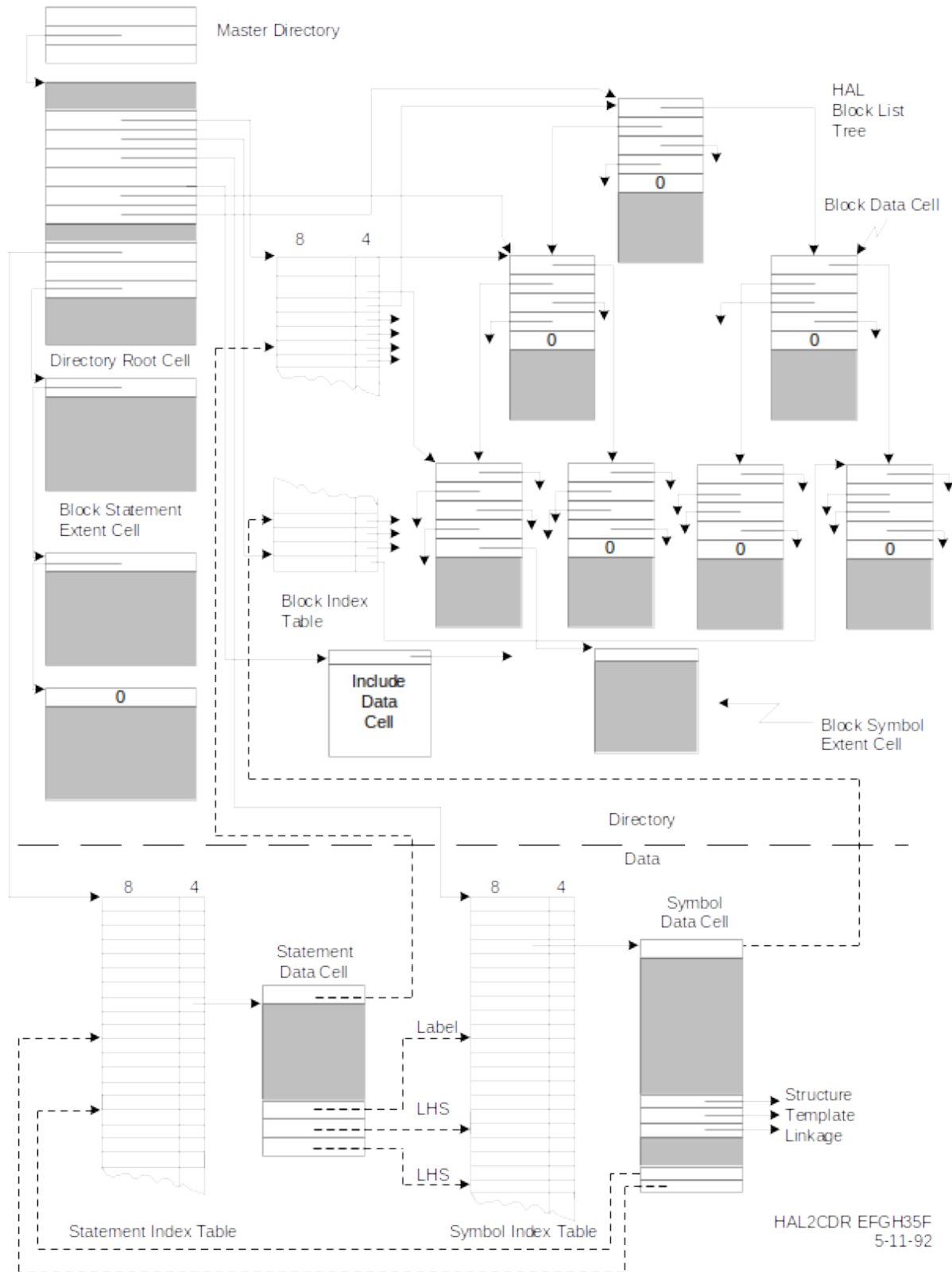


Figure 2-5 Simulation Data File Member Organization (Not all interconnections are shown)

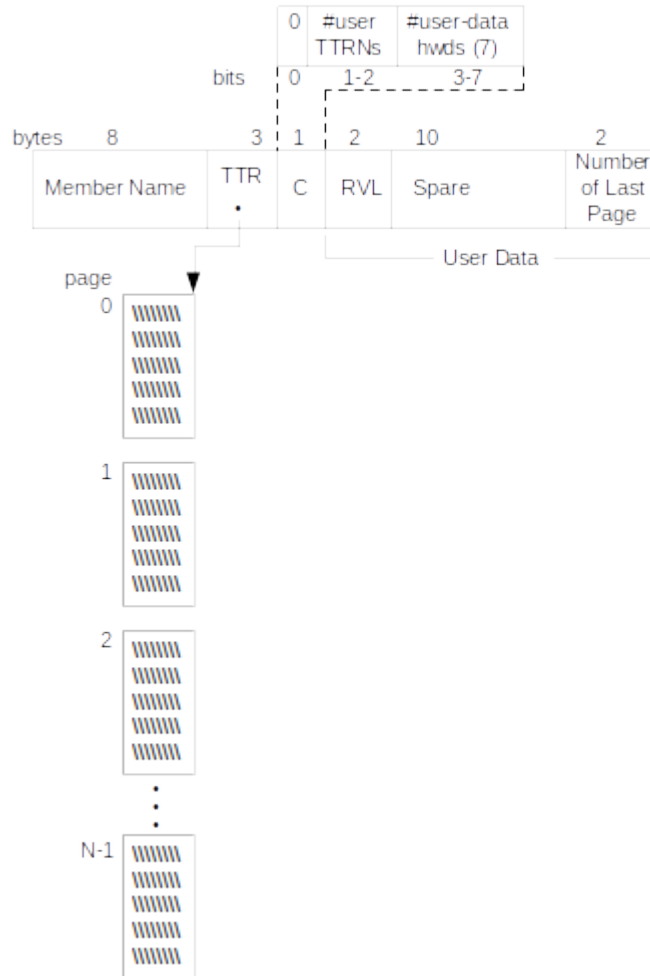


Figure 2-6 PDS-Level Organization of the Simulation Data Files

2.2.1 Simulation Data File Directory

The SDF Directory provides information about the organization and location of the various lists and component tables that make up the SDF member for a unit of compilation.

The directory consists of:

- Master Directory Cell (Figure 2-7 on page 16)
- Directory Root Cell (Figure 2-11 on page 20)
- Block Index Table (Figure 2-21 on page 35)
- Block Data Cell (Figure 2-25 on page 39)
- Block Symbol Extent Cell (Figure 2-29 on page 46)
- Statement Extent Cell (Figure 2-68 on page 99)

The directory also serves as the means for locating major groupings of data contained in the physical records of the data set member for the unit of compilation.

2.2.2 Master Directory Cell

The Master Directory Cell (Figure 2-7 on page 16) is always found at the very beginning (Record 0, Byte Offset 0) of the SDF member. It is the initial entry point into the SDF member. Among other things, this cell is used to determine the SDF Version Number and the location of the Directory Root Cell (Section 2.2.2.2, “Directory Root Cell” on page 18).

Fullword Offset	Halfword Offset	Byte Offset Decimal (Hex)	Field Number		Bytes
0	0	0 (0)	1	Phase 3 Version Number	2
-	1	2 (2)	2	Unused X'0000'	2
1	2	4 (4)	3	▲ First Cell of Directory Free Cell Chain	4
2	4	8 (8)	4	▲ Directory Root Cell	4
3	6	12 (C)	5	▲ First Cell of Data Free Cell Chain	4

First Physical Record of File

Figure 2-7 Master Directory Cell

The fields contained in the Master Directory Cell are described below:

Field No.	Description
1	This field contains the version number of Phase 3 (the SDF creation phase) of the HAL/S compiler used to compile this Compilation Unit. Every time a significant change is made to the SDFs, this Version Number is incremented by one.
2	Unused. Contains X'0000'.
3	SDF pointer to the Directory Free Cell Linked List. This list identifies space that was allocated for directory information, but was never used.
4	Pointer to the Directory Root Cell (Figure 2-11 on page 20)
5	SDF pointer to the Data Free Cell Linked List. Data Cells are cells like the Symbol Data Cell and Statement Data Cell. This list identifies space that was allocated for block, symbol, or statement data, but never used.

2.2.2.1 SDF Free Space

SDF members contain unused free space which is divided into two classes: Directory Free Space and Data Free Space (see Figure 2-8 on page 17). Each Free Space group is organized into cells of contiguous space and is a part of either the Data or the Directory Free Cell Linked List. Both the Directory Free Cell Linked List and the Data Free Cell Linked List terminate with a zero pointer (hex '00000000').

Figure 2-9 on page 18 illustrates the linked lists used in both the Directory Free Cell and Data Free Cell lists.

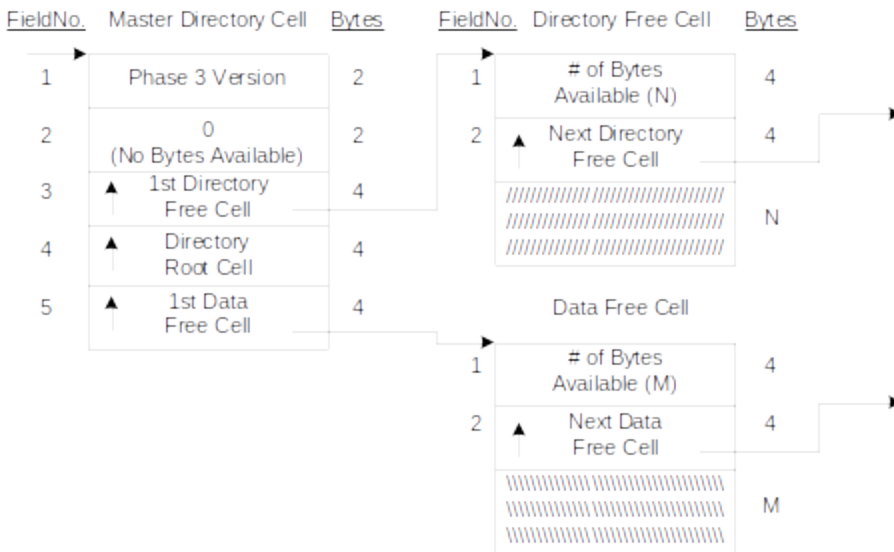


Figure 2-8 Free Cell Linkage

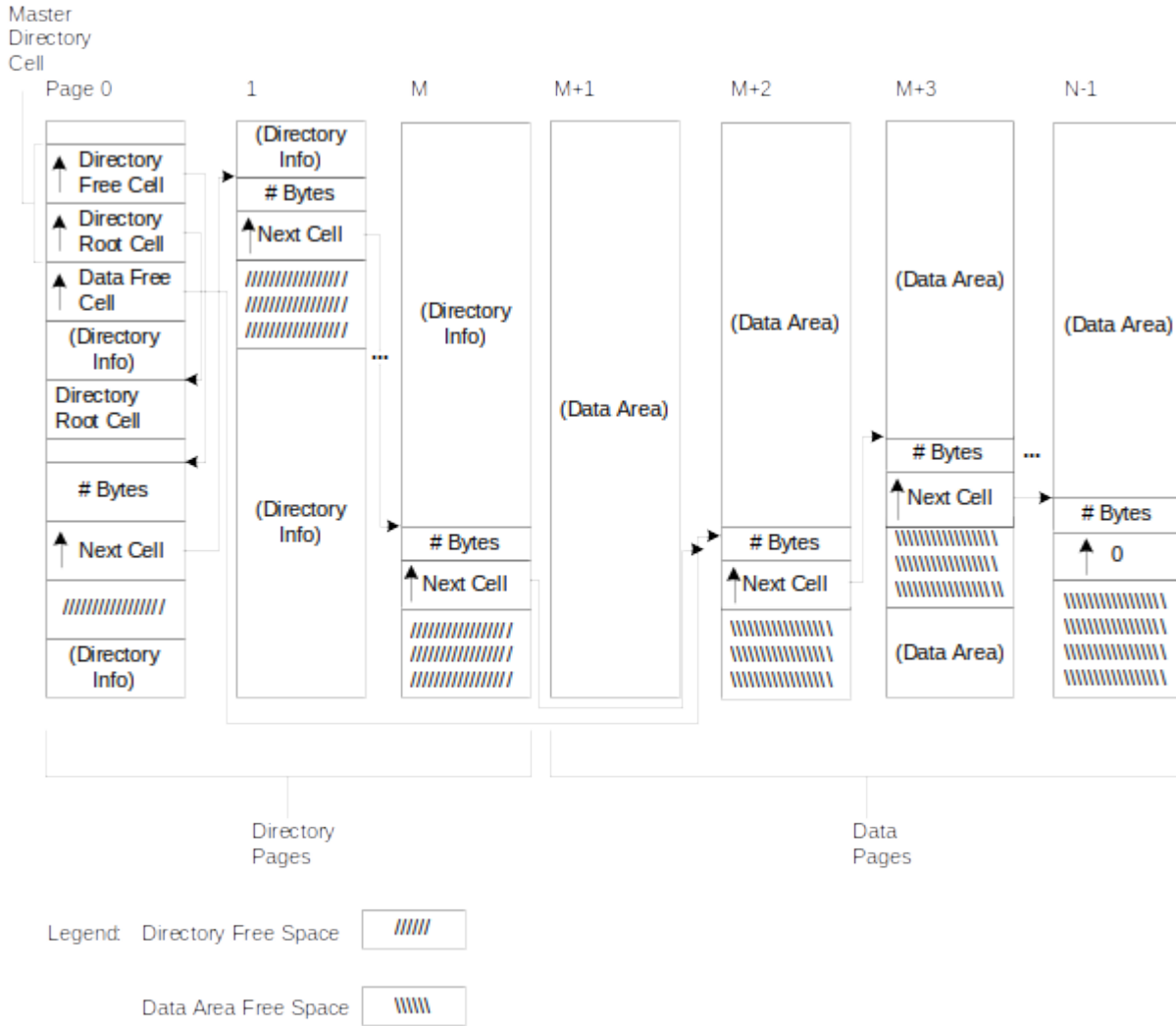


Figure 2-9 SDF Free Cell Linked Lists

2.2.2.2 Directory Root Cell

As shown in Figure 2-10 on page 19, the Directory Root Cell is pointed to by Field 4 of the Master Directory Cell. The Directory Root Cell (Figure 2-11 on page 20) locates subordinate Cells, locates the Symbol Index Table, and provides general information needed for statement processing. This latter information includes the location of the Statement Index Table and the values of the first and last internal statement numbers (ISNs). This ISN information is used in conjunction with the Statement Extent List to determine the pertinent physical records of the Statement Index Table. However, the information can be used directly to locate statement data pointers by means of a binary search whenever sufficient memory space exists to contain the complete Statement Index Table.

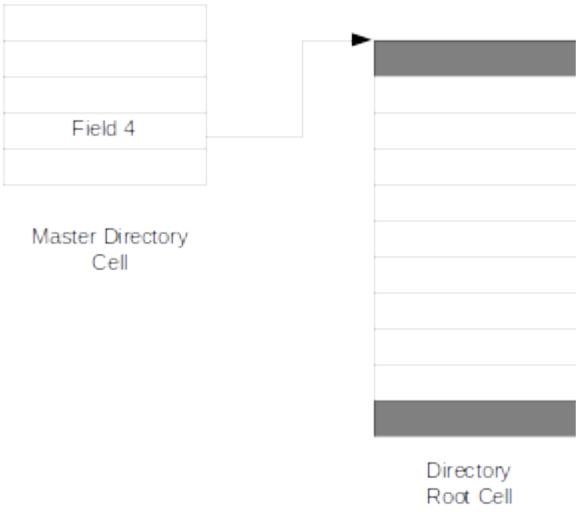


Figure 2-10 Master Directory/Directory Root Cell Overview

Fullword Offset	Halfword Offset	Byte Offset Decimal (Hex)	Field Number		Bytes
0	0	0 (0)	1	Flag Field	2
-	1	2 (2)	2	Number of Last Physical Record	2
1	2	4 (4)	3	Date of File Creation	4
2	4	8 (8)	4	Time of File Creation	4
3	6	12 (C)	5	Number of Last Directory Physical Record	2
-	7	14 (E)	6	Number of EXTERNAL Blocks	2
4	8	16 (10)	7	Number of Block Indices	2
-	9	18 (12)	8	Number of Symbols	2
5	10	20 (14)	9	▲ Head of Block Index Table	4
6	12	24 (18)	10		4
7	14	28 (1C)	11a		2
-	15	30 (1E)	11b		2
8	16	32 (20)	12a	#D or #P List Head of Compilation Unit Internal Symbols (Address Order) Within #D or #P	2
-	17	34 (22)	12b	#R List Head of Compilation Unit Remote Data	2
9	18	36 (24)	13	▲ First SYMBOL Index Table Entry	4
10	20	40 (28)	14a	Number of Stack Walkback Loops	2
-	21	42 (2A)	14b	Relative Address of Literal Area in #D CSECT	2
11	22	44 (2C)	15	▲ Compilation Unit Block Data Cell (Hierarchical Tree)	4
12	24	48 (30)	16	▲ Head of the HAL/S Block Tree (Alphabetic Tree)	4
13	26	52 (34)	17	Value of the First ISN in File	2
				⋮	

Figure 2-11 Directory Root Cell (Part 1 of 3)

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset Decimal (Hex)</u>	<u>Field Number</u>		<u>Bytes</u>
				• • •	
-	27	54 (36)	18	Value of Last ISN in this File	2
14	28	56 (38)	19	Number of Executable Statements	2
-	29	58 (3A)	20	Number of Statements	2
15	30	60 (3C)	21	▲ First Statement Index Table Entry	4
16	32	64 (40)	22	▲ First Cell of the INCLUDE Data Cell List	4
17	34	68 (44)	23	▲ First Cell of the Statement Extent Cell List	4
18	36	72 (48)	24a	First Statement Reference Number (SRN)	6
-	39	78 (4E)	24b	Include Count for First SRN	2
20	40	80 (50)	25a	Last Statement Reference Number (SRN)	6
-	43	86 (56)	25b	Include Count for Last SRN	2
22	44	88 (58)	26	Index of Compilation Unit Block Data Cell	2
-	45	90 (5A)	27	User Defined Compilation Unit Number (COMPUNIT)	2
23	46	92 (5C)	28	▲ Title Data Cell	4
24	48	96 (60)	29	Reserved (User Data)	8
26	52	104 (68)	30	Total Number of Symbols	4
27	54	108 (6C)	31	Total Number of Bytes of REPLACE Text	4
28	56	112 (70)	32	Total Number of Characters In Literal Table	4
29	58	116 (74)	33	Total Free Cell Space	2
-	59	118 (76)	34	COMSUB Parameter End	2
				• • •	

Figure 2-11 Directory Root Cell (Part 2 of 3)

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset Decimal (Hex)</u>	<u>Field Number</u>		<u>Bytes</u>
				⋮	
30	60	120 (78)	35	Number of XREF Table Entries	4
31	62	124 (7C)	36	Number of Symbols Specified in PARM Field	4
32	64	128 (80)	37	No. Bytes Specified for REPLACE Text in PARM Field	4
33	66	132 (84)	38	No. Characters Specified for Literals In PARM Field	4
34	68	136 (88)	39	Number of XREF Entries Specified in PARM Field	4
35	70	140 (8C)	40	Compiler Identification	12
38	76	152 (98)	41	▲ CARDTYPE Data	4
39	78	156 (9C)	42	▲ Initialization Table	4
40	80	160 (A0)	43	# Halfwords in Initialization Table	4
41	82	164 (A4)	44..47	Unused	4
42	84	168 (A8)	48	Offset to Start of Top Literal Area in #D	2
-	85	170 (AA)	49	Offset to End of Top Literal Area in #D	2
43	86	172 (AC)	50	▲ Literal Extent Table	4
44	88	176 (B0)	51	Number of Entries in Literal Extent Table	4
45	90	180 (B4)	52	▲ Function Index Table	4
46	92	184 (B8)	53	Number of Entries in Function Index Table	2
-	93	186 (BA)	54	Unused	2
47..50	94..100	188..200 (BC...C8)	55..62	Unused	16

Figure 2-11 Directory Root Cell (Part 3 of 3)

The meanings of these fields are as follows:

Field No. Description

- 1 This is a “flag” field containing binary flags which describe the various compilation conditions. The placement and meaning of these bits are as follows:

<u>Bit No.</u>	<u>Flag Name</u>	<u>Meaning When Set</u>
0	SRN_FLAG	File contains Statement Reference Numbers (SRNs). Statement Index Table entries are 12 bytes in size.
1	ADDRS_FLAG	Statement Data Cells contain 6 bytes of address information: relative addresses are for the first and last lines of the emitted code for that statement.

<u>Field No.</u>	<u>Description</u>		
1 (Cont'd)	<u>Bit No.</u>	<u>Flag Name</u>	<u>Meaning When Set</u>
	2	COMPOOL_FLAG	SDF was produced for a COMPOOL compilation.
	3	FC_FLAG	Identifies the SDF member as belonging to an FC compilation.
	4	OVERFLOW_FLAG	Indicates one or more overflow directory cells were allocated from the Data Free Cell Chain due to insufficient space in the initial record(s) of the file which were pre-allocated for directory data (i.e., not all directory information is on the initial physical records).
	5	NON_MONOTONIC_SRN_FLAG	SRNs are not monotonic (i.e., one or more SRNs have values that are less than that of their predecessor).
	6	NON_UNIQUE_SRN_FLAG	SRNs are not unique (i.e., one or more SRNs have equal values).
	7	NOTRACE_FLAG	Unused for HAL/S-FC compilations.
	8	HIGHOPT	Allows the compiler to perform optimizations that may not be valid when the programmer uses %MACROs to bypass the type checking protection provided by the HAL/S language.
	9	BIT_FLAG	The current compilation unit contains an instance of a BIT variable which is assigned from a multi-instruction masking operation.
	10	HALMAT_FLAG	SDF includes HALMAT.
	11	FCDATA_FLAG	Unused for HAL/S-FC compilations.
	12	SDL_FLAG	Identifies the compilation as being compiled using the SDL option.
	13	DATA_REMOTE	SDF produced by compiler with the DATA_REMOTE directive in effect.

<u>Field No.</u>	<u>Description</u>	
1 (Cont'd)	<u>Bit No.</u>	<u>Flag Name</u>
		<u>Meaning When Set</u>
	14	REL6_FLAG
		Identifies the SDF member as being of the format specified in Revision 6 of the HAL/SDL ICD.
	15	NEW_FLAG
		Always set. Used to maintain upward compatibility of old SDFs.
2	The number (starting with 0) of the last physical record (page) in the SDF file.	
3	The date the file was created, in the format: Day of the year + (1000 *(year-1900)).	
4	The time the file was created: the number of the centiseconds since midnight of the creation date.	
5	The number of the last directory page. If the OVERFLOW FLAG is set, however, some directory information is located in the data area. All directory information in the data area is referred to by pointers.	
6	The number of EXTERNAL blocks. The total number of EXTERNAL blocks (COMPOOLS, PROGRAMS, PROCEDURES, NON-HAL) referred to (Included) in the compilation.	
7	The number of HAL/S blocks in the unit of compilation. Also, the number of entries in the HAL/S Block Index Table.	
8	The total number of legitimate symbols for which the SDF file contains data. Also, the number of entries in the Symbol Index Table.	
9	Pointer to the first entry in the Block Index Table.	
10	The total number of emitted AP-101 instructions generated in the compilation (zero for COMPOOL).	
11a	Index in Symbol Index Table for the name of the Unit of Compilation.	
11b	List Head (Index into Symbol Index Table) for Linked List of all Internal Symbols ordered alphabetically.	
12a	List Head (Index into Symbol Index Table) for Linked List of all internal symbols residing in #D or #P CSECTS ordered by increasing address.	

Field No. Description

- 12b List Head (Index into Symbol Index Table) for Linked List of all internal symbols defined in a #R CSECT ordered by increasing address. Note that #R CSECT cannot exist if the SDL compiler option is specified (SDL_FLAG is TRUE).
- 13 Pointers to the first entry (symbol) in the Symbol Index Table.
- 14a The number of stack walkback loops generated by Phase 2.
- 14b The relative address of the literal area from the beginning of the #D CSECT. This field contains X“FFFF” when the literal area does not exist.
- 15 Pointer to the Block Data Cell which represents the primary block in the unit of compilation (e.g., the program block in a PROGRAM compilation). Also, this is a pointer to the root block of the Block Tree Hierarchy (see Section 2.2.2.2.3.2, “HAL/S Block Data Cell” on page 35).
- 16 Pointer to the Block Data Cell containing the largest number of defined symbols. Also, this is the pointer to the root block of the Symbol Quantity and Alphabetic Name Tree (see Section 2.2.2.2.3.2, “HAL/S Block Data Cell” on page 35).
- 17 First statement number. This is the internal statement number (ISN), as assigned by Phase 1, corresponding to the first executable statement in the compilation unit. Together with Field 21, this value provides the direct correspondence between internal statement numbers (ISNs) and pointers to the pertinent statement entries in the Statement Index Table.
- 18 Last statement number. This will always be the internal statement number of the final CLOSE statement.
- 19 Number of declare and executable statements. Field 19 is \leq Field 20 and represents the number of statement entries that contain data (i.e., number of entries which point to a Statement Data Cell).
- 20 Number of Statements (Field 18 – Field 17 + 1) of all types, beginning with the first executable statement. Note that comments and HAL/S compiler directives are not considered statements and, so, are not counted.
- 21 Pointers to the first entry in the Statement Index Table (see description for Field 17).
- 22 Pointer to the first cell of the Include Data Cell List.
- 23 Pointer to the first cell of the Statement Extent Cell List.

Field No. Description

- 24a First Statement Reference Number (SRN) contained within the SDF member.
- 24b Include Count associated with the first SRN contained in the SDF member. See Section 2.2.2.2.5.1, "Statement Index Table" on page 86, for a description of Include Counts.
- 25a Last Statement Reference Number contained within the SDF member.
- 25b Include Count associated with the last SRN contained in the SDF member. See Section 2.2.2.2.5.1, "Statement Index Table" on 86, for a description of Include Counts.
- 26 Index in Block Index Table for the Unit of Compilation.
- 27 User supplied compilation unit number (COMPUNIT). This number specifies the BLOCK ID for a block within a unit of compilation. The compilation unit number ranges in value from 0 to 511 and is set only if the COMPUNIT Parameter is specified at compile time.
- 28 Pointer to Title Data Cell (see Figure 2-13 on page 29 for description of contents).
- 29 Reserved.
- 30 Total number of symbols in the unit of compilation. Actual size of compiler symbol table.
- 31 Actual number of bytes of REPLACE text.
- 32 Actual number of characters in the Compiler Literal Table.
- 33 Number of unused bytes in the SDF accounted for in the Free Cell Linked Lists.
- 34 This field is zero unless the compilation is a COMSUB. In that case, this field is the index in the Symbol Index Table of the last symbol entered into the Phase 1 symbol table at the point in the COMSUB compilation when the last parameter to the COMSUB is formally declared.
- 35 Actual number of XREF Table entries.
- 36 Maximum number of symbols (may be specified by the user in the compile-time parameter string.)

Field No. Description

- 37 Maximum number of bytes of REPLACE text (may be specified by the user in the compile-time parameter string.)
- 38 Maximum number of characters of literal text (may be specified by the user in the compile-time parameter string.)
- 39 Maximum number of XREF entries (may be specified by the user in the compile-time parameter string.)
- 40 Contains the 10 character EBCDIC string obtained from the ID field of the File Control Block from the first phase of the compiler. The last 2 bytes of this field are unused and are 0.
- 41 Pointer to the CARDTYPE Data Cell (see Figure 2-15 on page 29 for description of contents).
- 42 Pointer to the Initialization Table.
- 43 Number of halfwords in the Initialization Table.
- 44 Unused
- 45 Unused
- 46 Unused
- 47 Unused
- 48 Offset to start of the literal area from the beginning of the #D CSECT.
- 49 Offset to end of the literal area from the beginning of the #D CSECT.
- 50 Pointer to Literal Extent Table within SDF (see Section 2.2.2.2.9.2.1, "Literal Extent Table" on page 122).
- 51 Number of entries in Literal Extent Table (see Section 2.2.2.2.9.2.1, "Literal Extent table" on page 122).
- 52 Pointer to Function Index Table for cross-reference of shaping functions. This field contains X'FFFFFFFF' if the Function Index Table does not exist (see Section 2.2.2.2.8.1, "Function Index Tables" on page 114).
- 53 Number of entries in Function Index Table (see Section 2.2.2.2.8.1, "Function Index Tables" on page 114).
- 54-62 Unused

Notes:

1. Fields 17-25 may have values for COMPOOLS as with other compilation units.
2. If SRN_FLAG=0 (i.e., no SRNs are present), the fields 23-25 contain zero.
3. If HALMAT_FLAG=0 then fields 50 and 51 are zero.
4. Fields 48 and 49 contain X'FFFF' if there is no #D CSECT or if the literal area does not exist, within the #D CSECT.

2.2.2.2.1 Compiler Data

This section describes the SDF Cells which contain the data specified for the TITLE and CARDTYPE parameters in the compile-time parameter string.

2.2.2.2.1.1 Title Data Cell

This Cell contains the information specified for the TITLE compile-time option.

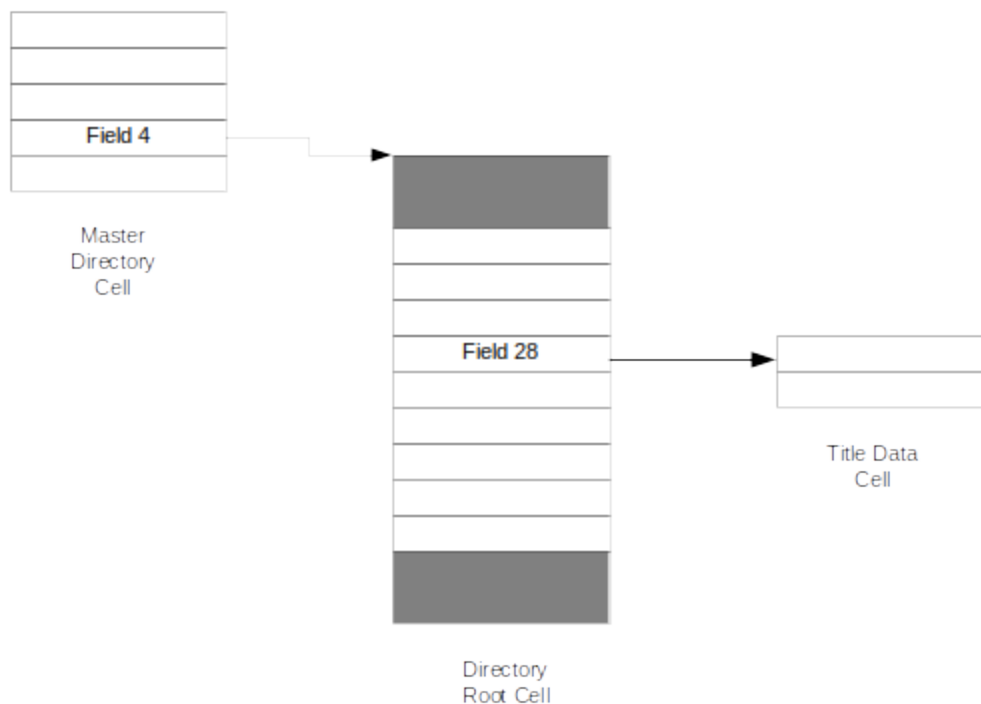


Figure 2-12 Title Data Cell Overview

The fields contained in this cell are described below:

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <p>No. of Characters</p> <hr/> <p>Title Contents</p> </div>	1
-	-	1	2		Up to 60 bytes

Figure 2-13 Title Data Cell

<u>Field No.</u>	<u>Description</u>
1	This field specifies the number of characters contained in Field 2 below.
2	This field contains the title specified by the TITLE option. This field may contain up to 60 characters.

2.2.2.2.1.2 CARDTYPE Data Cell

This Cell contains the information specified for the CARDTYPE compile-time option.

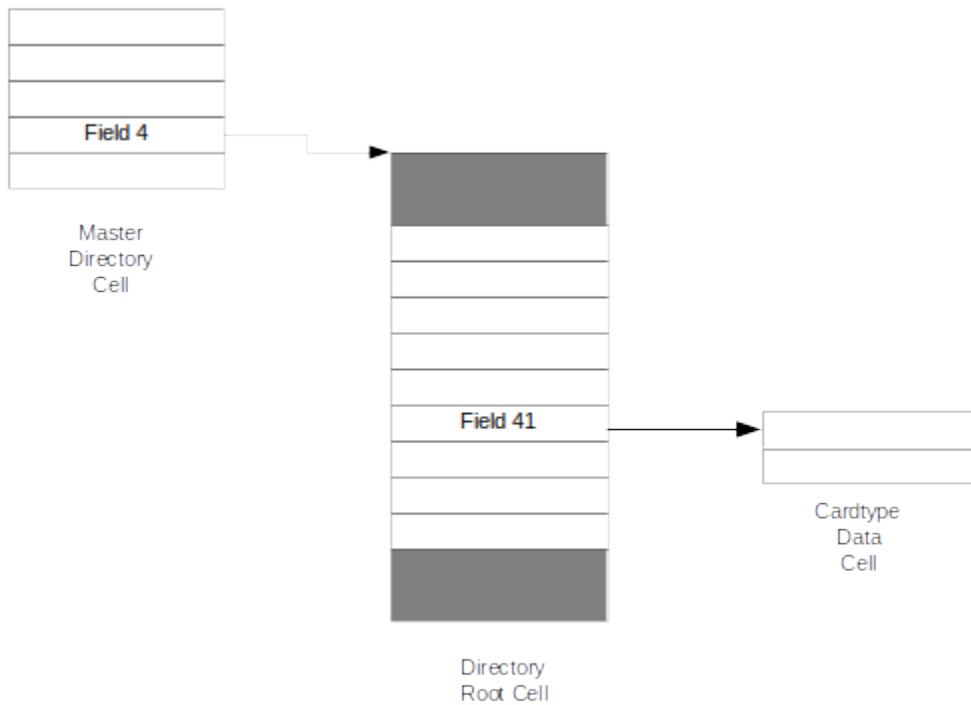


Figure 2-14 Cardtype Data Cell Overview

The CARDTYPE Compiler Option allows HAL/S statements with non-standard Card Types to be mapped into the standard types listed in Field 2 below.

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> No. of Characters Text </div>	1
-	-	1	2		Up to 100 bytes

Figure 2-15 CARDTYPE Data Cell

<u>Field No.</u>	<u>Description</u>
------------------	--------------------

- 1 This field specifies the number of characters contained in Field 2 below.
- 2 This field contains the character data specified by the CARDTYPE option. The standard Card Types are:

- E – Exponent statement line
- M – Main statement line
- S – Subscript statement line
- C – Comment line
- D – Compiler Directive line

The Statement types are mapped using the following format:

CT=VEWMXSYCZD

In this example, statements containing V, W, X, Y, and Z in the first columns are mapped to the types of E, M, S, C, and D, respectively. It is necessary to specify only those Card Types which are non-standard. This field may contain up to 100 characters.

2.2.2.2.1.3 Initialization Table

This table contains the initialization data for non-NAME variables. The table is formatted the same as the #D or #P CSECT. #R data is not supported.

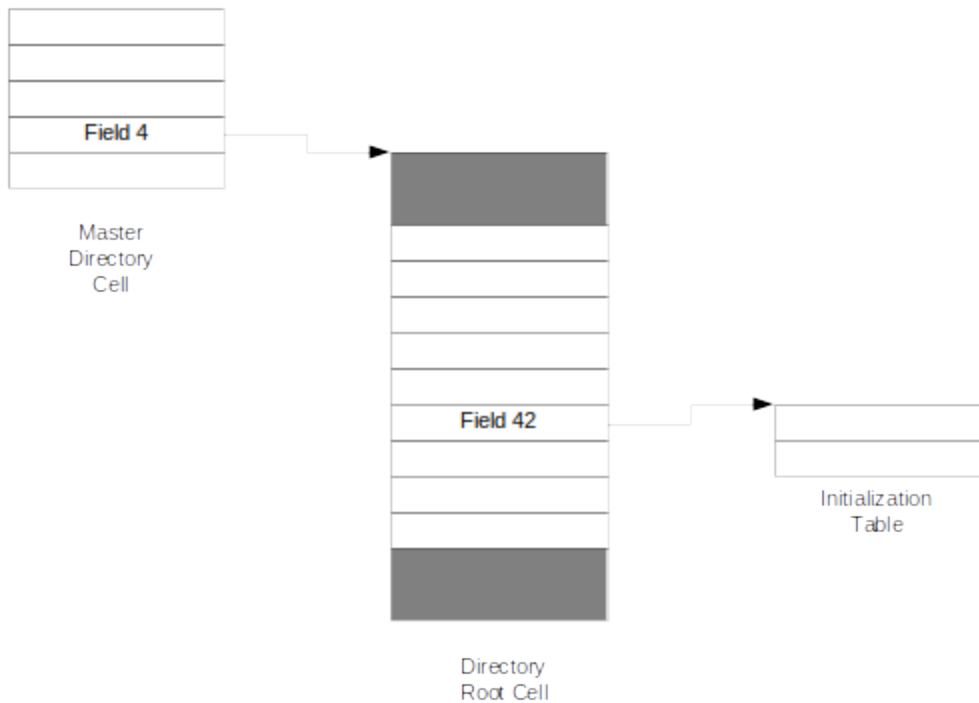


Figure 2-16 Initialization Table Overview

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	Initialization Value 1	2
-	1	2	1	Initialization Value 2	2
				⋮	
			1	Initialization Value N	2

Field No.	Description
1	Each field is a halfword of data formatted like the #D or #P CSECT. The Symbol data Cell's field 10A is an offset into the Initialization Table, pointing to the beginning of the initialization data for that symbol. The INITIAL flag can be used to determine if data was initialized.

Figure 2-17 Initialization Table

2.2.2.2.2 Include Text Data

As shown in Figure 2-18 on page 32, the Include Text Data consists of a linked list of Include Library Member Cells that is pointed to by Field 22 of the Directory Root Cell. Each cell in the list provides information about the name, revision level, and catenation number of a distinct include library member. The cell also indicates the JCL DDname associated with the PDS library in which the member is located, and a list of the SRNs containing compiler directives that INCLUDE the member.

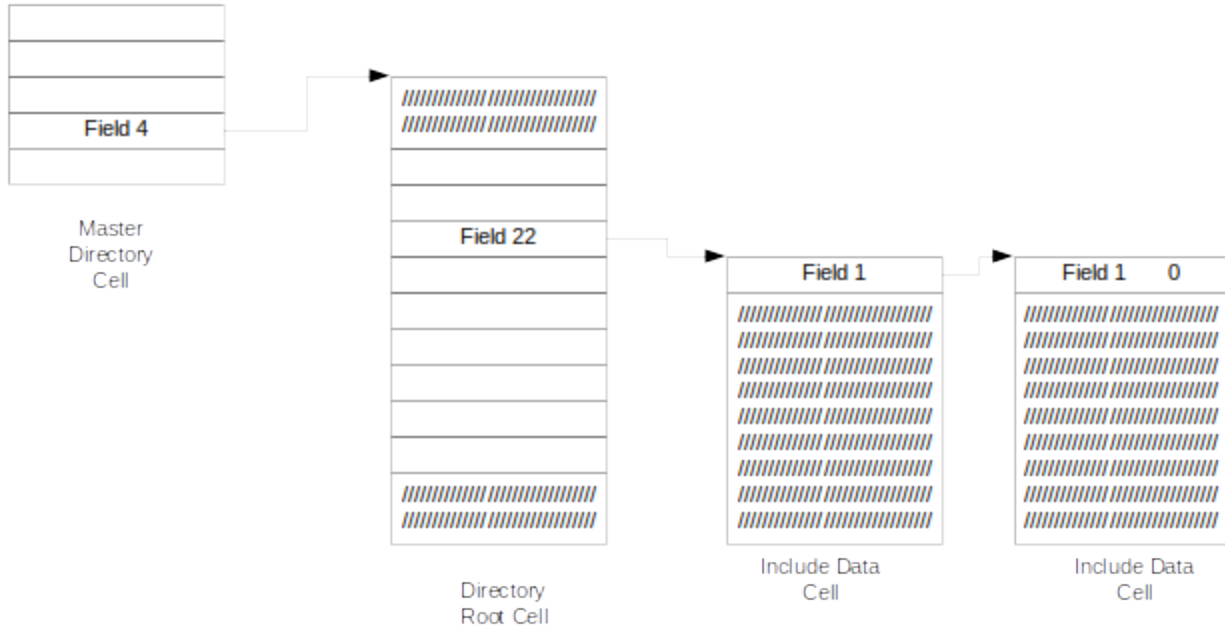


Figure 2-18 Include Data Overview

Fullword Offset	Halfword Offset	Byte Offset Decimal (Hex)	Field Number		
0	0	0 (0)	1	▲	4
1	2	4 (4)	2		8
3	6	12 (C)	3		2
-	7	14 (E)	4		2
-	-	15 (F)	5	0 1 2 3 4 5 6 7	1 Flag Bits
4	8	16 (10)	6		1
-	-	17 (11)	7		6
			7		6

Pointer to Next Member in the Include Cell Chain	4
Include Library Member Name	8
Revision Number	2
Catenation Number	2
Number of SRNs	1
SRN 1	6
•	
•	
SRN N	6

Figure 2-19 Include Data Cell

The meanings of the fields of the Include Data Cell are as follows:

<u>Field No.</u>	<u>Description</u>																		
1	A pointer to the next cell in the list. The cells are linked alphabetically by member name.																		
2	The name of the Include Library Member, in EBCDIC.																		
3	The Revision Level consists of two EBCDIC characters which are set to 00 (X"F0F0") when the member is created.																		
4	An Include Library defined by a JCL DD statement may consist of a concatenated list of PDS libraries. The catenation number is the index in that list of the data set in which the member was located.																		
5	The flag bits indicate the type of the INCLUDE directive, and which JCL DD statement defines the library in which the member was located. The flag bits are as follows:																		
	<table border="1"> <thead> <tr> <th><u>Bit</u></th> <th><u>Meaning When Set</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>The member was an SDF.</td> </tr> <tr> <td>1</td> <td>The member was located in the library specified by the OUTPUT5 DD statement.</td> </tr> <tr> <td>2</td> <td>The member was located in the library specified by the HALSDF DD statement.</td> </tr> <tr> <td>3</td> <td>The member was located in the library specified by the OUTPUT8 DD statement.</td> </tr> <tr> <td>4</td> <td>The member was located in the library specified by the OUTPUT6 DD statement.</td> </tr> <tr> <td>5</td> <td>The member was located in the library specified by the INCLUDE DD statement.</td> </tr> <tr> <td>6</td> <td>TEMPLATE flag. The directive has the form: D INCLUDE TEMPLATE.</td> </tr> <tr> <td>7</td> <td>REMOTE flag. The directive specifies the keyword REMOTE.</td> </tr> </tbody> </table>	<u>Bit</u>	<u>Meaning When Set</u>	0	The member was an SDF.	1	The member was located in the library specified by the OUTPUT5 DD statement.	2	The member was located in the library specified by the HALSDF DD statement.	3	The member was located in the library specified by the OUTPUT8 DD statement.	4	The member was located in the library specified by the OUTPUT6 DD statement.	5	The member was located in the library specified by the INCLUDE DD statement.	6	TEMPLATE flag. The directive has the form: D INCLUDE TEMPLATE.	7	REMOTE flag. The directive specifies the keyword REMOTE.
<u>Bit</u>	<u>Meaning When Set</u>																		
0	The member was an SDF.																		
1	The member was located in the library specified by the OUTPUT5 DD statement.																		
2	The member was located in the library specified by the HALSDF DD statement.																		
3	The member was located in the library specified by the OUTPUT8 DD statement.																		
4	The member was located in the library specified by the OUTPUT6 DD statement.																		
5	The member was located in the library specified by the INCLUDE DD statement.																		
6	TEMPLATE flag. The directive has the form: D INCLUDE TEMPLATE.																		
7	REMOTE flag. The directive specifies the keyword REMOTE.																		
6	The number of SRN entries (field 7).																		
7	The SRNs of the compiler directives which INCLUDE this member. Each SRN consists of 6 EBCDIC characters.																		

2.2.2.2.3 Block Data Structures

The Block Data Structures consist of the Block Index Table, Block Data Cell, and the Block Symbol Extent Cell. These cells and tables provide the means by which the proper symbol data can be located in the SDF pages. The HAL/S Block Data Cell also provides information about the blocks in a unit of compilation. As shown in Figure 2-20 on page 34, several different fields of the Directory Root Cell point to the different HAL/S Block Data Structures. The next three sections describe the Block Data Structures in more detail.

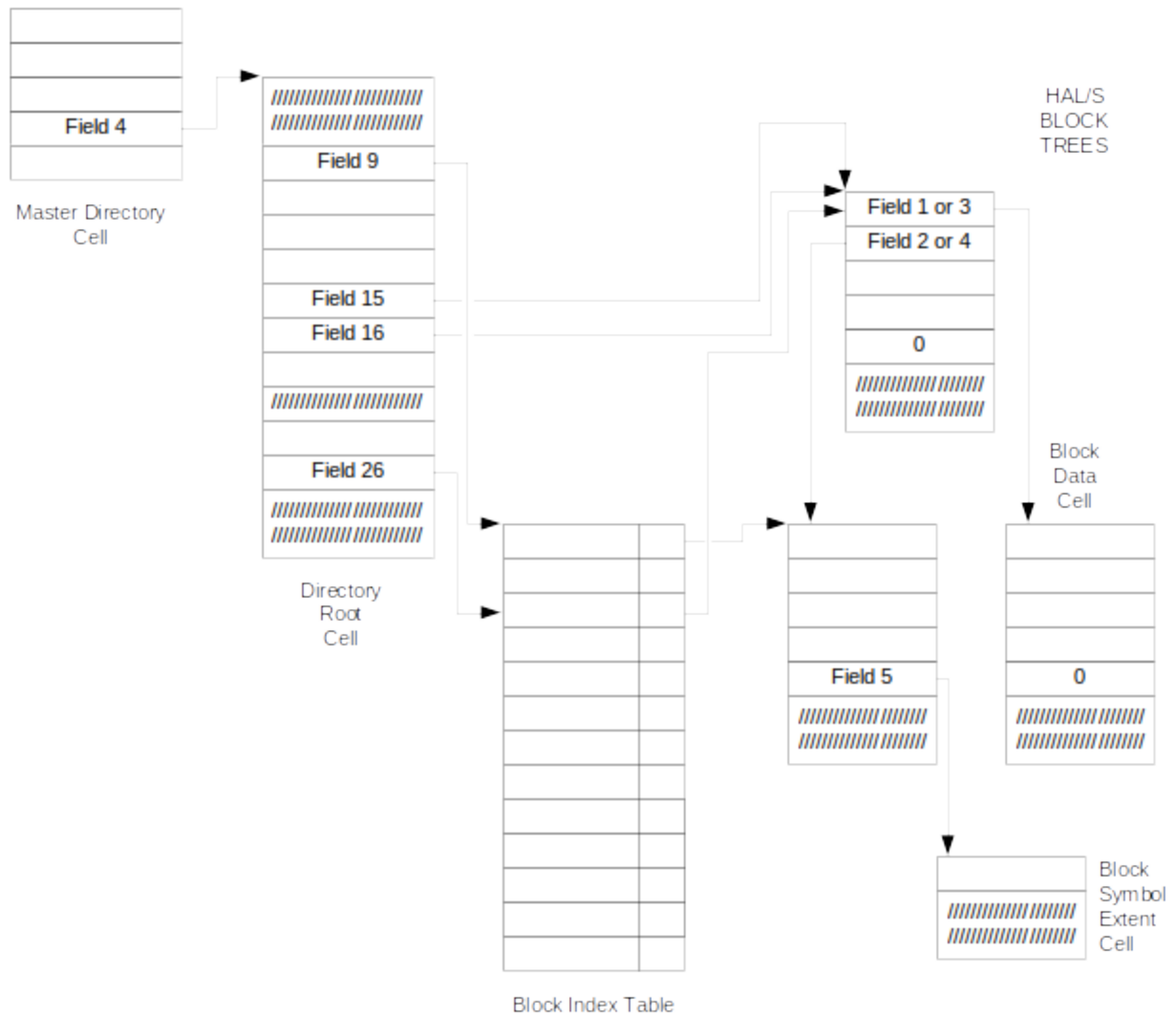


Figure 2-20 Block Data Structures Overview

2.2.2.2.3.1 Block Index Table

The Block Index Table (see Figure 2-21 on page 35) locates the various HAL/S Block Data Cells. It is ordered in accordance with the alphabetic order of the block CSECT

names. The Block Index serves as a convenient reference to identify the HAL/S block to which a statement or symbol belongs.

Except for COMPOOLs, the CSECT names contained in the Block Index Table are the names of the Code CSECT generated for each Block. When the Block represents a COMPOOL, the CSECT name is the name of the COMPOOL CSECT (e. g., #PNNNNNN). The CSECT naming conventions are described in Section 4.0, "CSECT/MEMBER NAMING CONVENTIONS" on page 1 of this document.

A binary search on the Block Index Table can be used to locate a particular block. However, a direct search of the HAL/S Block List using the linkages which are based upon symbol definition frequency and alphabetic order (see the following section) is faster.

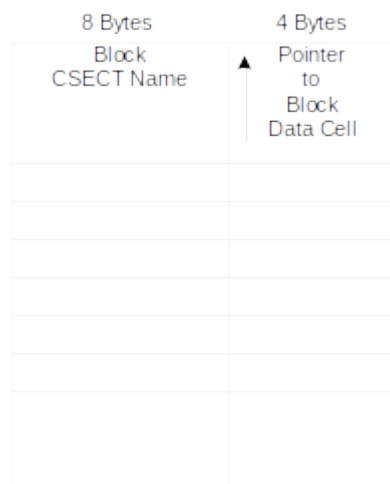


Figure 2-21 Block Index Table

2.2.2.2.3.2 HAL/S Block Data Cell

The HAL/S Block Data Cell provides the means by which the information about a symbol unique to a HAL/S Block can be found; it also identifies and supplies information about the HAL/S Blocks themselves. The HAL/S Block Data Cell, shown in Figure 2-25 on page 39, corresponds to the HAL/S Blocks (COMPOOL, PROGRAM, PROCEDURE, FUNCTION, TASK, UPDATE) within a unit of compilation. The cells are logically organized in two different tree structures: one based on the symbol frequency and the alphabetic block name order of its members (see Figure 2-26 on page 41 for an example), and the other based upon the hierarchical block structure of its members (see Figure 2-27 on page 42 for an example). The first tree structure provides an easy and efficient way to locate a particular block in a unit of compilation. The second tree structure provides an easy way to locate the variables of a block which are within the name scope of a block but not in the block where they are being referenced (e.g., the hierarchical linkages would provide a way to SNAP the active variables of any encompassing blocks at the time a block terminated). Entry to the list is in one of three ways: from the Directory Root Cell to the root block of the Alphabetic Name Tree; from

the Directory Root Cell to the root block of the Hierarchical Block Tree; and from a pointer in the Block Index Table.

The HAL/S Block Data Cell and its corresponding Block Symbol Extent Cell serve to identify the regions of the Symbol Index Table that are pertinent to a HAL/S Block. If all of the symbols within a unit of compilation lie on a single physical record of the Symbol Index Table, no Block Symbol Extent Cell is referenced and indexes exist in the Block Data Cell to identify the first and last symbols in the Symbol Index Table (see Figure 2-22 on page 36). However, if the symbols do not lie on a single physical record, a four-byte pointer exists to the Block Symbol Extent Cell which then identifies the regions pertinent to the block (see Figure 2-23 on page 37 and Figure 2-24 on page 38).

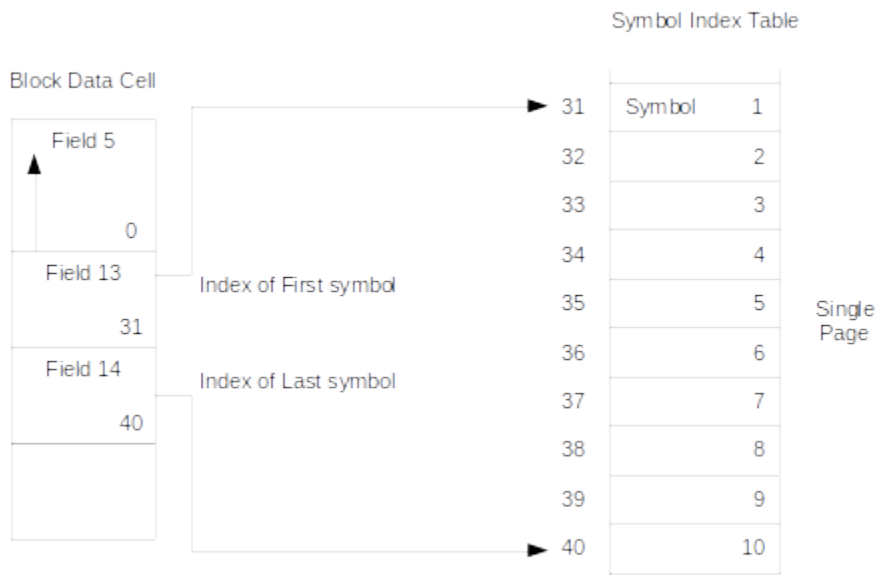


Figure 2-22 All Symbols Contained on One SDF Page for Block

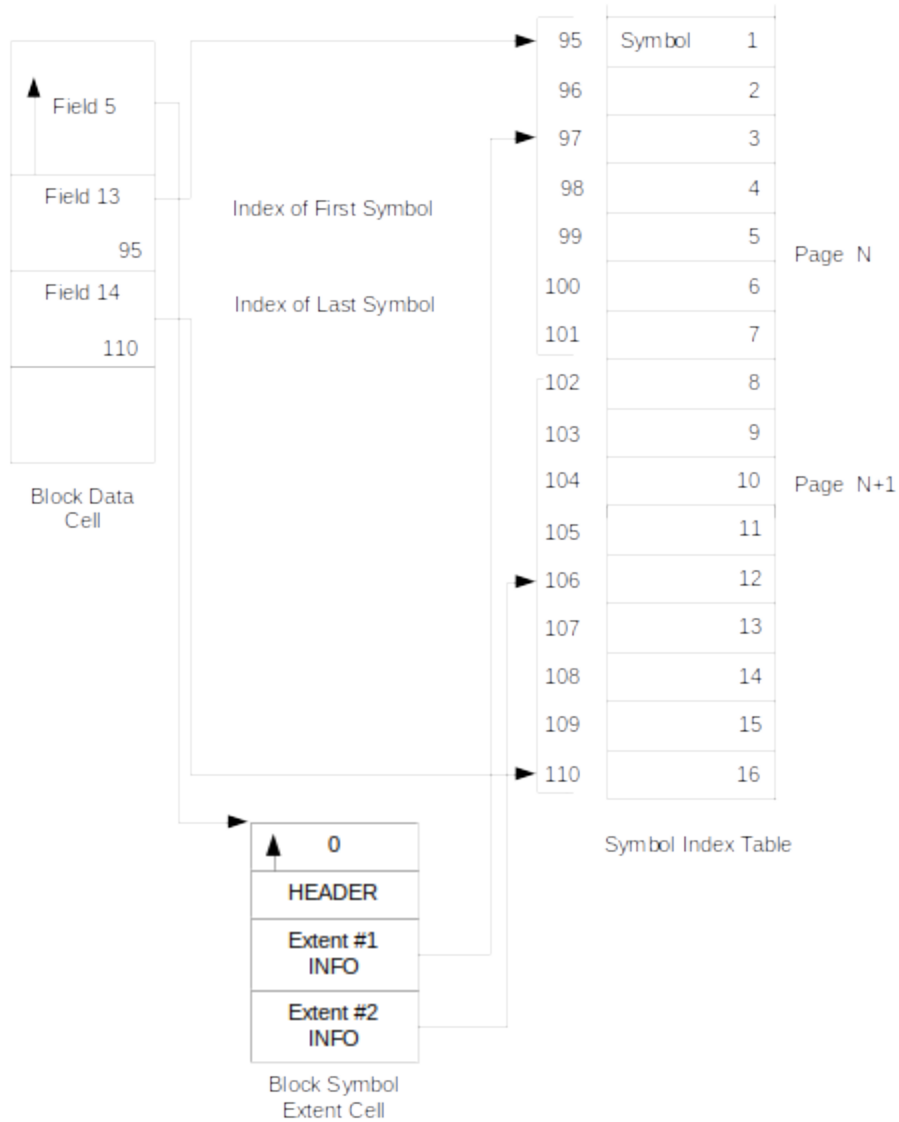


Figure 2-23 Symbols Contained on Multiple SDF Pages for Block

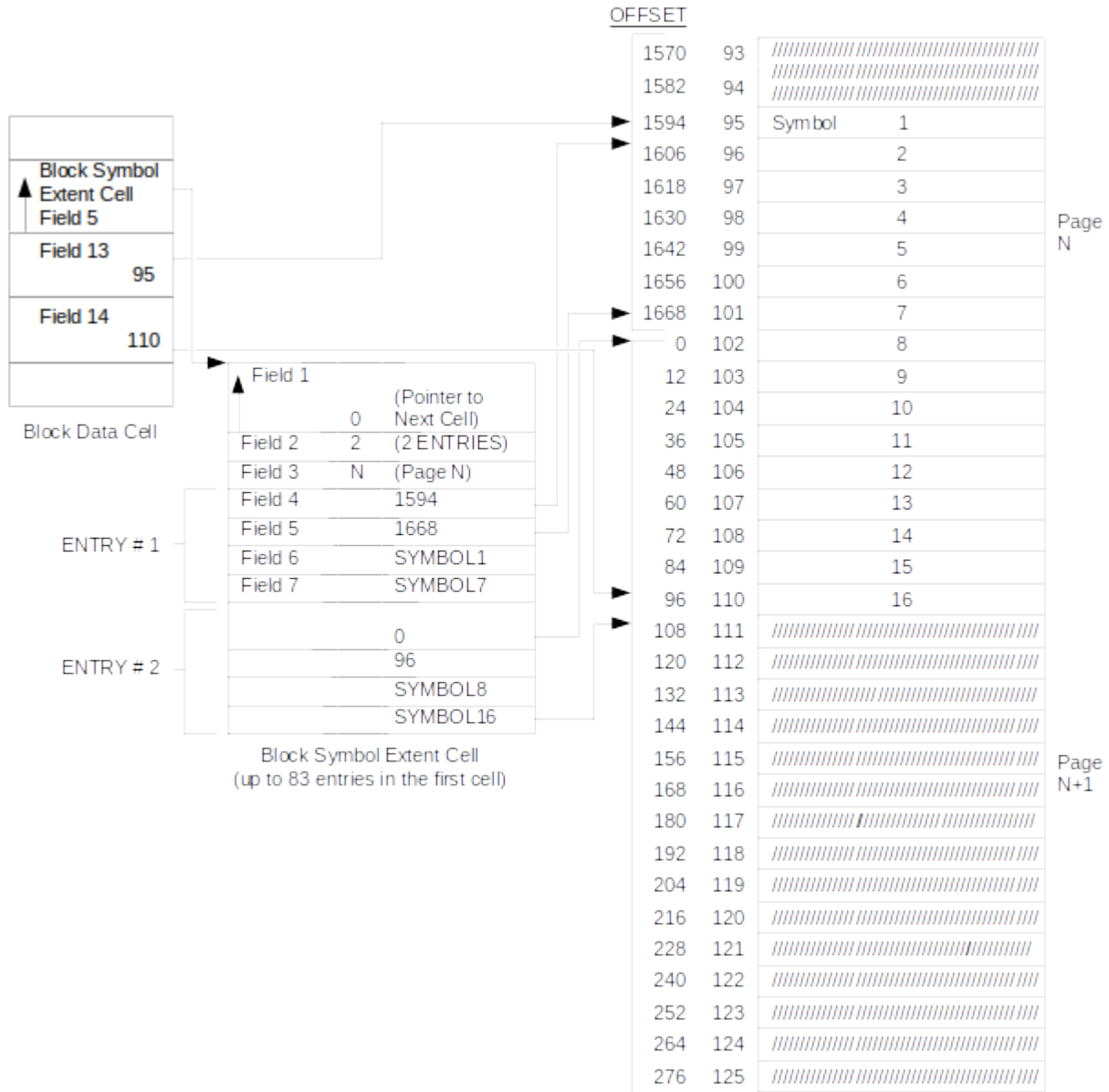


Figure 2-24 Example of Block Symbol Extent Cell

Fullword Offset	Halfword Offset	Byte Offset Decimal Hex	Field Number			
0	0	0 (0)	1	↑	Pointer to next higher member in HAL/S Block Data Cell Tree	4
1	2	4 (4)	2	↑	Pointer to next lower member in HAL/S Block Data Cell Tree	4
2	4	8 (8)	3	↑	Pointer to 1st nested block within the scope of this block	4
3	6	12 (C)	4	↑	Pointer to next block at same level or enclosing block	4
4	8	16 (10)	5	↑	Pointer to Block Symbol Extent Cell for block*	4
5	10	20 (14)	6a		Symbol Index Number of Block Name	2
-	11	22 (16)	6b		Unused	2
6	12	24 (18)	7		0 1 2 3 4 5 6 7	1
-	-	25 (19)	8		Version No. of Block Template	1
-	13	26 (1A)	9		Block Index Number in the Block Index Table	2
7	14	28 (1C)	10		Block ID	2
-	15	30 (1E)	11		Block Category	1
-	-	31 (1F)	12		Function Type	1
8	16	32 (20)	13		Index to first symbol of block in the Symbol Index Table	2
-	17	34 (22)	14		Index to last symbol of block in the Symbol Index Table	2
9	18	36 (24)	15		First Stmt. No. of HAL/S Block	2
-	19	38 (26)	16		Last Stmt. No. of HAL/S Block	2
10	20	40 (28)	17a		ISN of 1st executable stmt. after initial DECLAREs	2
-	21	42 (2A)	17b		Head of Stack Frame Variables (in address order)	2
11	22	44 (2C)	18		Length of Block Name	1
-	-	45 (2D)	19		Block Name	1-32

* Note: This value is set to 0 if a Block Symbol Extent Cell entry does not exist (i.e., all symbol references for this HAL/S block lie upon a single physical record of the Symbol Index Table.)

Figure 2-25 Block Data Cell

The HAL/S Block Data Cell is described in Figure 2-25 on page 39. The meanings of its fields are as follows:

<u>Field No.</u>	<u>Description</u>
1	Pointer to the next HAL/S Block Data Cell whose name is alphabetically higher. This field is 0 if no references exist. This field, in association with Field 2, define the Symbol Quantity and Alphabetic Tree. This Tree Structure contains Block Data Cells for both internal and external blocks. See Figure 2-26 on page 41 for more information.
2	Pointer to the next member of the HAL/S Block Data Cell whose name is alphabetically lower. This pointer is zero if no reference exists.
3	Pointer to the first nested block within the scope of this block. This pointer is zero if no nested block exists. This field and Field 4 define the Hierarchical Block Tree. This tree contains only internal blocks. See Figure 2-27 on page 42 for more information.
4	Pointer to a block which is at the same level as this block (e. g., in Figure 2-27 on page 42 for the MERV, this field would point to the block HENRY; Field 3 would point to JOHN). If no other block exists at the same level, this pointer is negative (two's complement) and points to the HAL/S Block Data Cell which has enclosing scope (e.g., for block HENRY in Figure 2-27 on page 42, this field points back to TOM).

Block Name	Number Symbols	Block Type
Fred	420	COMPOOL
Tom	116	PROCEDURE
Mary	92	COMPOOL
Alice	85	PROCEDURE
Henry	73	PROCEDURE
Merv	60	PROCEDURE
Ben	42	PROCEDURE
John	35	PROCEDURE
Ellen	30	PROCEDURE

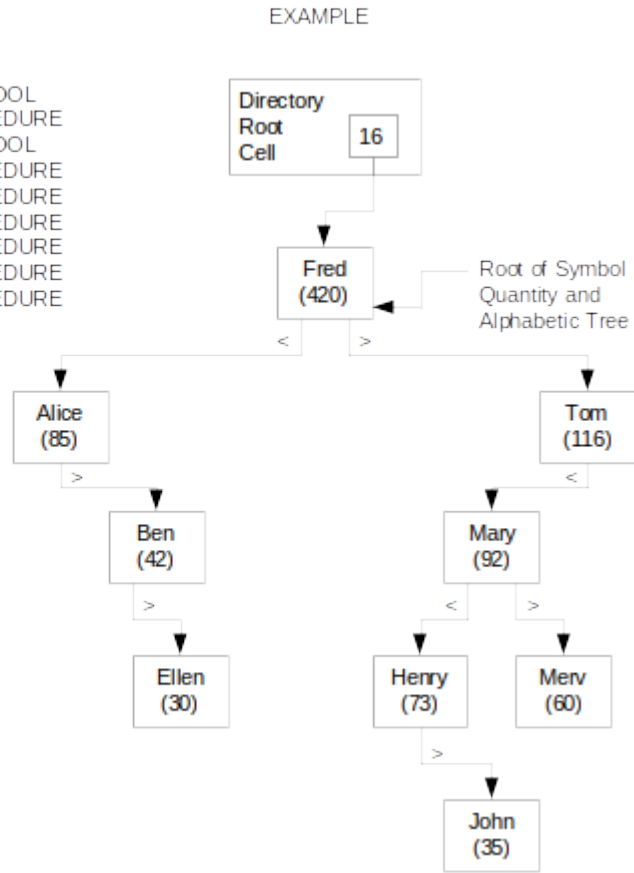


Figure 2-26 Alphabetic Name Tree

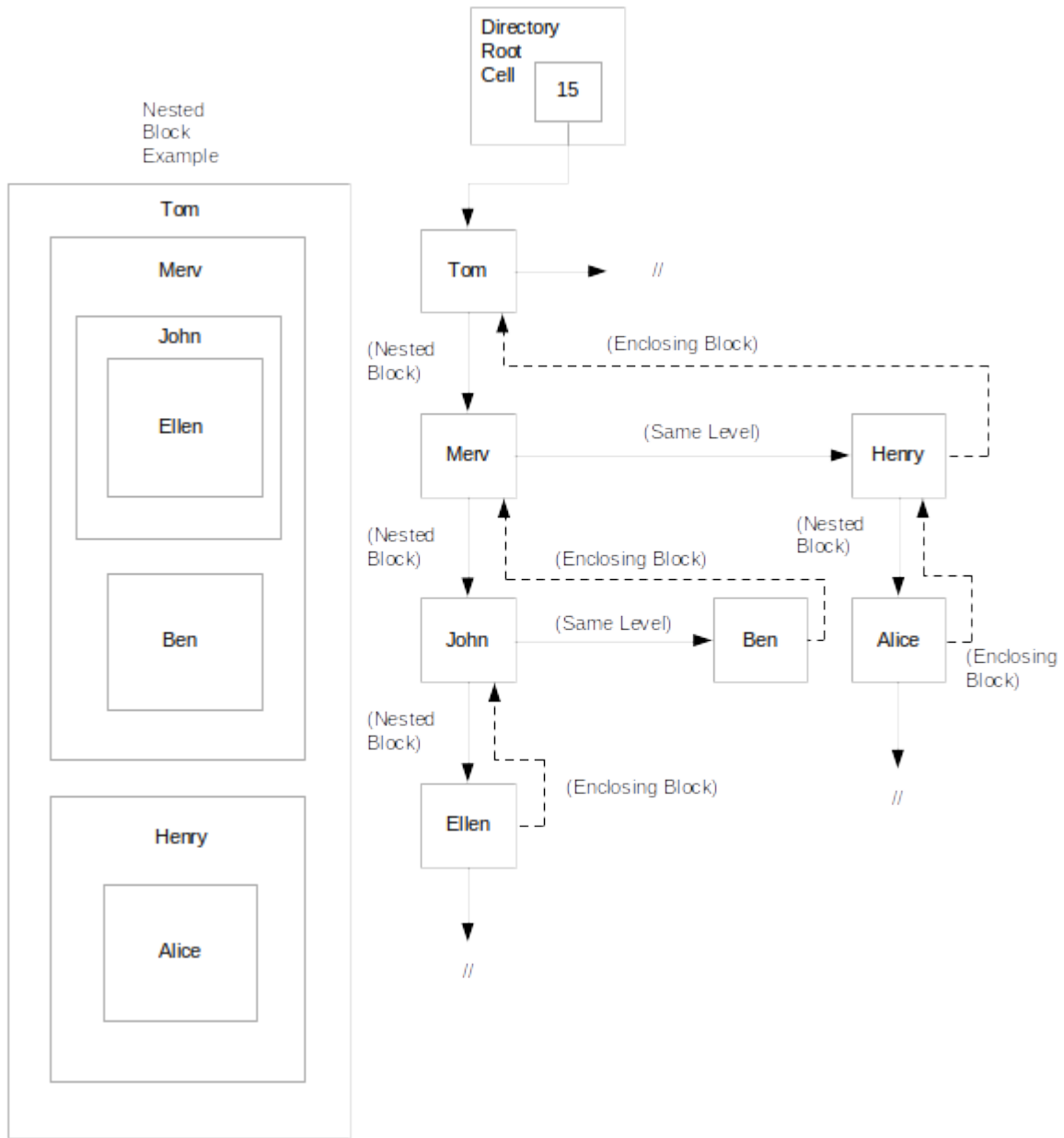


Figure 2-27 Hierarchical Block Tree

<u>Field No.</u>	<u>Description</u>
5	Pointer to the Block Symbol Extent Cell, if one exists. If all symbols belonging to this block lie entirely within a single page of the Symbol Index Table, then no Extent entry exists and this pointer will be zero.
6a	Index in the Symbol Index Table for this block's name (symbol number).

Field No. Description

- 6b Unused
- 7 The flag bits identify block characteristics, such as REENTRANT, EXCLUSIVE, and RIGID. The bit assignments are:
- | <u>Bit No.</u> | |
|----------------|----------------|
| 0 | REENTRANT Flag |
| 1 | EXCLUSIVE Flag |
| 2 | ACCESS Flag |
| 3 | RIGID Flag |
| 4 | EXTERNAL Flag |
| 5 | NONHAL Flag |
| 6 | Unused |
| 7 | Unused |
- 8 The version number of the template for this block. This field is only defined for blocks that are EXTERNAL and for the Compilation Unit Block.
- 9 The entry number (index) of the block in the Block Index Table.
- 10 The Block ID is a unique number assigned to the block. It occupies the rightmost 7 bits of the field. When executing the code for this block, this same number is found in the low order 7 bits of the Block ID in the Local Block Data area. The Block ID with its Compilation Unit (COMPUNIT) Number can be used along with an offset to locate a variable in another "active" stack space. A stack variable may only be used if it is "active" (i.e., belongs to the same block, or an encompassing block, in which the action is to be taken).
- 11 The category of the HAL/S block (i.e., COMPOOL, PROGRAM, TASK, PROCEDURE, FUNCTION, and UPDATE). The codes for each of these block categories are as follows:
- | <u>No.</u> | |
|------------|-----------|
| 1 | PROGRAM |
| 2 | PROCEDURE |
| 3 | FUNCTION |
| 4 | COMPOOL |
| 5 | TASK |
| 6 | UPDATE |

<u>Field No.</u>	<u>Description</u>																																																
12	The type of FUNCTION. This field contains a non-zero value only if Field No.11 contains a 3.The code for each of the FUNCTION types is the same as the Function Types listed in the Symbol Data Cell. The types are as follows: <table border="1"> <thead> <tr> <th colspan="3"><u>CODE</u></th> </tr> <tr> <th>Decimal</th> <th>Hex</th> <th></th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1</td> <td>BIT (16-bits)</td> </tr> <tr> <td>2</td> <td>2</td> <td>CHARACTER</td> </tr> <tr> <td>3</td> <td>3</td> <td>MATRIX (SP)</td> </tr> <tr> <td>4</td> <td>4</td> <td>VECTOR (SP)</td> </tr> <tr> <td>5</td> <td>5</td> <td>SCALAR (SP)</td> </tr> <tr> <td>6</td> <td>6</td> <td>INTEGER (SP)</td> </tr> <tr> <td>7-8</td> <td>7-8</td> <td>Not Used</td> </tr> <tr> <td>9</td> <td>9</td> <td>BIT (32-bit)</td> </tr> <tr> <td>10</td> <td>A</td> <td>Not Used</td> </tr> <tr> <td>11</td> <td>B</td> <td>MATRIX (DP)</td> </tr> <tr> <td>12</td> <td>C</td> <td>VECTOR (DP)</td> </tr> <tr> <td>13</td> <td>D</td> <td>SCALAR (DP)</td> </tr> <tr> <td>14</td> <td>E</td> <td>INTEGER (DP)</td> </tr> <tr> <td>16</td> <td>10</td> <td>STRUCTURE</td> </tr> </tbody> </table>	<u>CODE</u>			Decimal	Hex		1	1	BIT (16-bits)	2	2	CHARACTER	3	3	MATRIX (SP)	4	4	VECTOR (SP)	5	5	SCALAR (SP)	6	6	INTEGER (SP)	7-8	7-8	Not Used	9	9	BIT (32-bit)	10	A	Not Used	11	B	MATRIX (DP)	12	C	VECTOR (DP)	13	D	SCALAR (DP)	14	E	INTEGER (DP)	16	10	STRUCTURE
<u>CODE</u>																																																	
Decimal	Hex																																																
1	1	BIT (16-bits)																																															
2	2	CHARACTER																																															
3	3	MATRIX (SP)																																															
4	4	VECTOR (SP)																																															
5	5	SCALAR (SP)																																															
6	6	INTEGER (SP)																																															
7-8	7-8	Not Used																																															
9	9	BIT (32-bit)																																															
10	A	Not Used																																															
11	B	MATRIX (DP)																																															
12	C	VECTOR (DP)																																															
13	D	SCALAR (DP)																																															
14	E	INTEGER (DP)																																															
16	10	STRUCTURE																																															
13-14	Indexes to the first and last entries in the Symbol Index Table for the block.																																																
15-16	First and last internal statement numbers (ISN) of the blocks. These two fields are zero for a COMPOOL compilation.																																																
17a	The ISN of the first executable statement of the block following the initial DECLAREs in the block.																																																
17b	The index in the Symbol Index Table of the initial symbol of the set of address-ordered symbols referring to stack space variables.																																																
18	The number of characters in the block name. This is a number from one to 32.																																																
19	The name of the block. This field is variable in length and contains up to 32 characters.																																																

2.2.2.2.3.3 Block Symbol Extent Cell

The Block Symbol Extent Cell (Figure 2-29 on page 46) identifies the first and last symbols for each of the physical records of a HAL/S block in the Symbol Index Table. The

Extent Cell identifies the first physical record of the Symbol Index Table pertinent to the HAL/S Block. The cell then supplies the first eight characters of the names and the offsets of the first and last symbols occurring in this record. In turn, the names and offsets of the first and last symbols for each of the remaining physical records of the Symbol Index Table are supplied. It should be noted that a one-to-one correspondence exists between the position of a reference in the Extent Cell relative to the initial reference and the physical record number to which the reference applies in relation to the first physical record of the Symbol Index Table (see Figure 2-31 on page 48). This relationship applies since all of the symbols for a block are grouped together. Since the symbols for a HAL/S block are organized alphabetically in the Symbol Index Table and their physical records are contiguous, the Extent Cell can be used to isolate quickly the pertinent physical record of the Symbol Index Table in which a symbol lies.

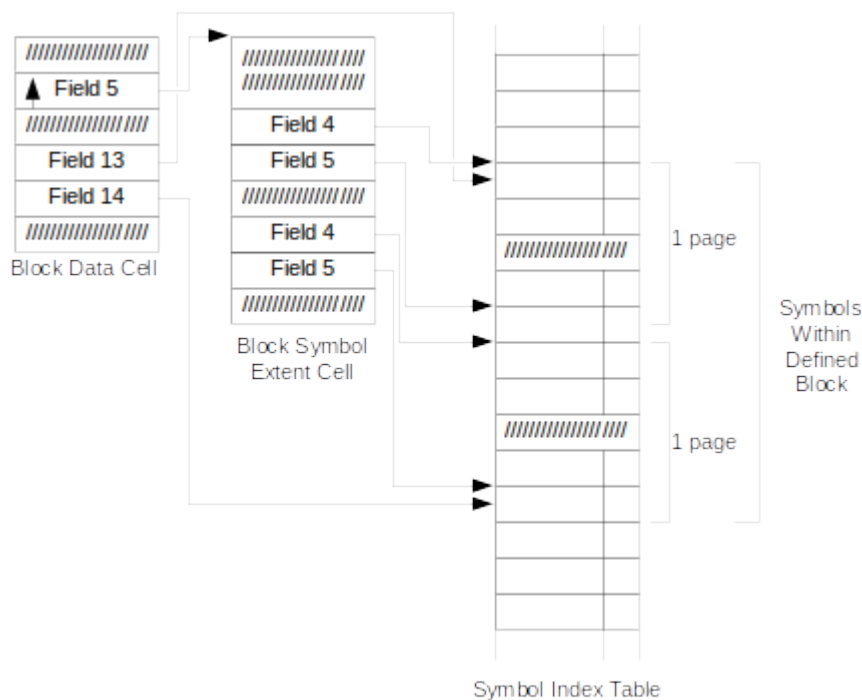


Figure 2-28 Relationship of Block Data Cells, Block Symbol Extent Cells, and Symbol Index Table

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset Decimal (Hex)</u>	<u>Field Number</u>			
0	0	0 (0)	1	Successor	4	
1	2	4 (4)	2	Number of Extent Entries	2	
-	3	6 (6)	3	Page Number of 1st Physical Record of Symbol Index Table	2	
2	4	8 (8)	4	First Offset	2	1st Physical Record Corresponding to Field 3
-	5	10 (A)	5	Last Offset	2	
3	6	12 (C)	6	First Symbol on Block	8	
5	10	20 (14)	7	Last Symbol on Block	8	
7	14	28 (1C)	4	First Offset	2	2nd Physical Record
-	15	30 (1E)	5	Last Offset	2	
8	16	32 (20)	6	First Symbol on Block	8	
10	20	40 (28)	7	Last Symbol on Block	8	
12	24	48 (30)	4	First Offset	2	3rd Physical Record
-	25	50 (32)	5	Last Offset	2	
13	26	52 (34)	6	First Symbol on Block	8	
15	30	60 (3C)	7	Last Symbol on Block	8	
17	34	68 (44)	4	First Offset	2	4th Physical Record
-	35	70 (46)	5	Last Offset	2	
18	36	72 (48)	6	First Symbol on Block	8	
20	40	80 (50)	7	Last Symbol on Block	8	
						• • •

Figure 2-29 Block Symbol Extent Cell

2.2.2.2.4 Symbol Data Structures

The Symbol Data Structures consist of the Symbol Index Table, Symbol Data Cell, Constant Value Cells, Replace Text Cells, Procedure/Function Formal Parameter Cells, and Name Terminal Initialization Cells. These data structures provide information about symbol types, attributes, memory locations, and initialization values. The Symbol Data Structures also provide information about the relative position of a symbol within a structure, as well as define the statements in which a symbol is declared, modified, used as a subscript, or referenced.

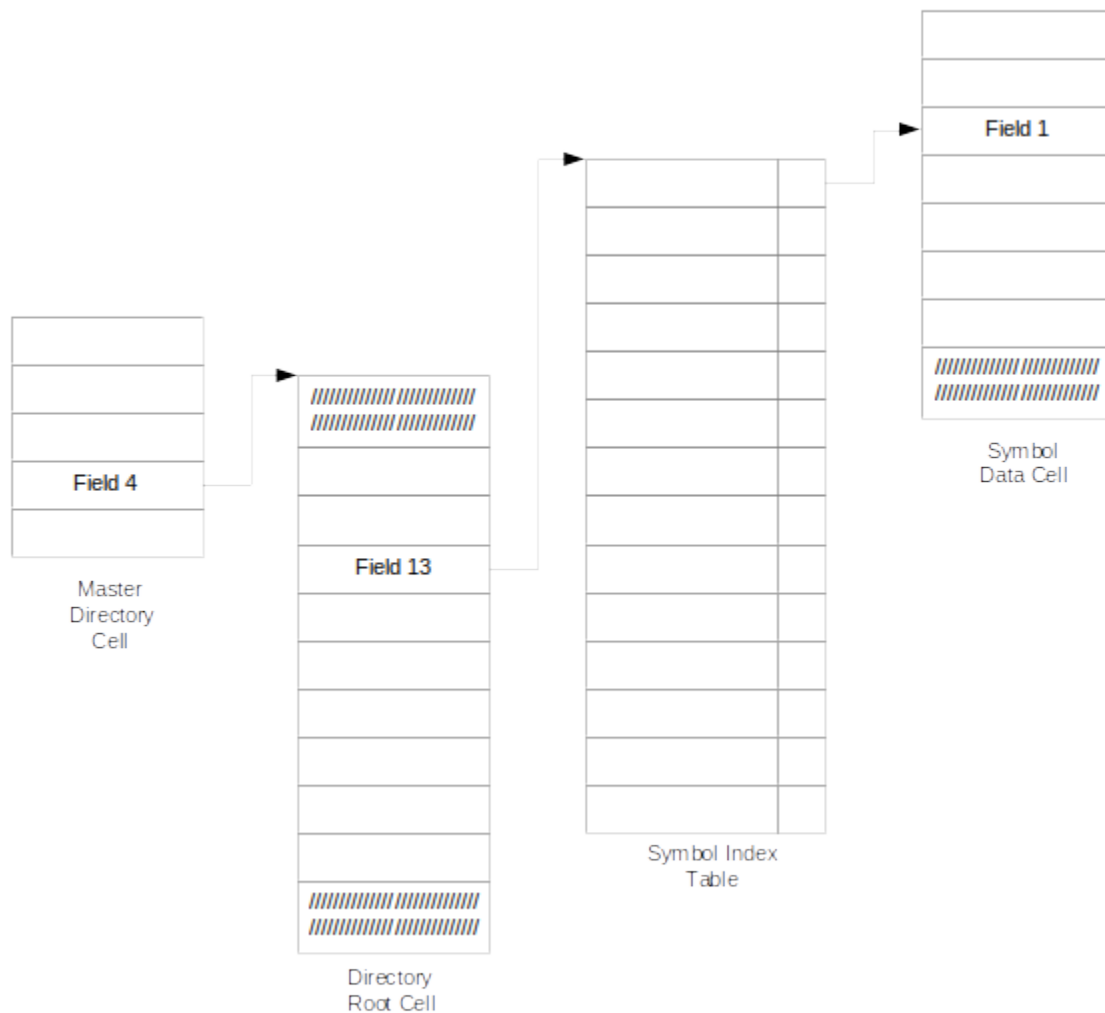


Figure 2-30 Symbol Data Structures Overview

2.2.2.2.4.1 Symbol Index Table

The Symbol Index Table (Figure 2-31 on page 48) provides the means by which a simple binary search via a pointer can be established to a symbol's data. The table is organized

into physical records which are obtained through application of the Symbol Directory Tables. Each physical record contains 1680 bytes of information or 140 entries of 12 bytes each. Every entry consists of an 8 byte field containing a maximum of eight characters of the symbol name and a 4 byte pointer to the Symbol Data Cell. The symbol names in the physical records for a HAL/S Block, not the table itself, are organized alphabetically. An entry exists for each label and variable declared in the compilation, except for those variables of the INCLUDED COMPOOLS which are not used in the compiler code (i.e., not referenced or assigned).

As the table carries only the first eight characters of a name, any excess characters are found in the Symbol Data Cell (see Section 2.2.2.4.2, "Symbol Data Cell" on page 49) at the entry pointed to by the 4 byte pointer. As the first eight characters may not be unique, it is necessary whenever more than eight characters are represented to check any excess to insure a proper reference. If a match is not found, it may be necessary to check both forwards and backwards from this point, especially if this point was realized from a binary search. Also, it may sometimes be necessary to secure the next physical record to continue the search for a unique reference.



Figure 2-31 Symbol Index Table

2.2.2.2.4.2 Symbol Data Cell

The Symbol Data Cell is referenced by pointers in the Symbol Index Table and provides all of the information that is known about a symbol (except the first eight or less characters of the name). The cell and its symbol data are variable in length. A cell (Figure 2-32 on page 50) contains information about the symbol type, attributes, relative memory location, number of bytes of memory occupied, and the block in which it is defined. A cell may also contain the symbol name continuation, number and range of dimensionality, structure template linkages, etc.

The Symbol Data Cell may also include a list of the statements in which the symbol is referenced, assigned, or declared. The statement references are in the form of indexes (ISNs) to the Statement Index Table and contain flag bits defining whether the statement DECLAREs, References, Assigns, or uses the variable as a subscript (combinations are possible). If all of the statement references cannot be contained on the physical record for the Symbol Data Cell, the data is extended to another physical record by means of a pointer. Whether or not the list is extended is indicated by either a halfword or fullword of hex'F's immediately preceding the pointer.

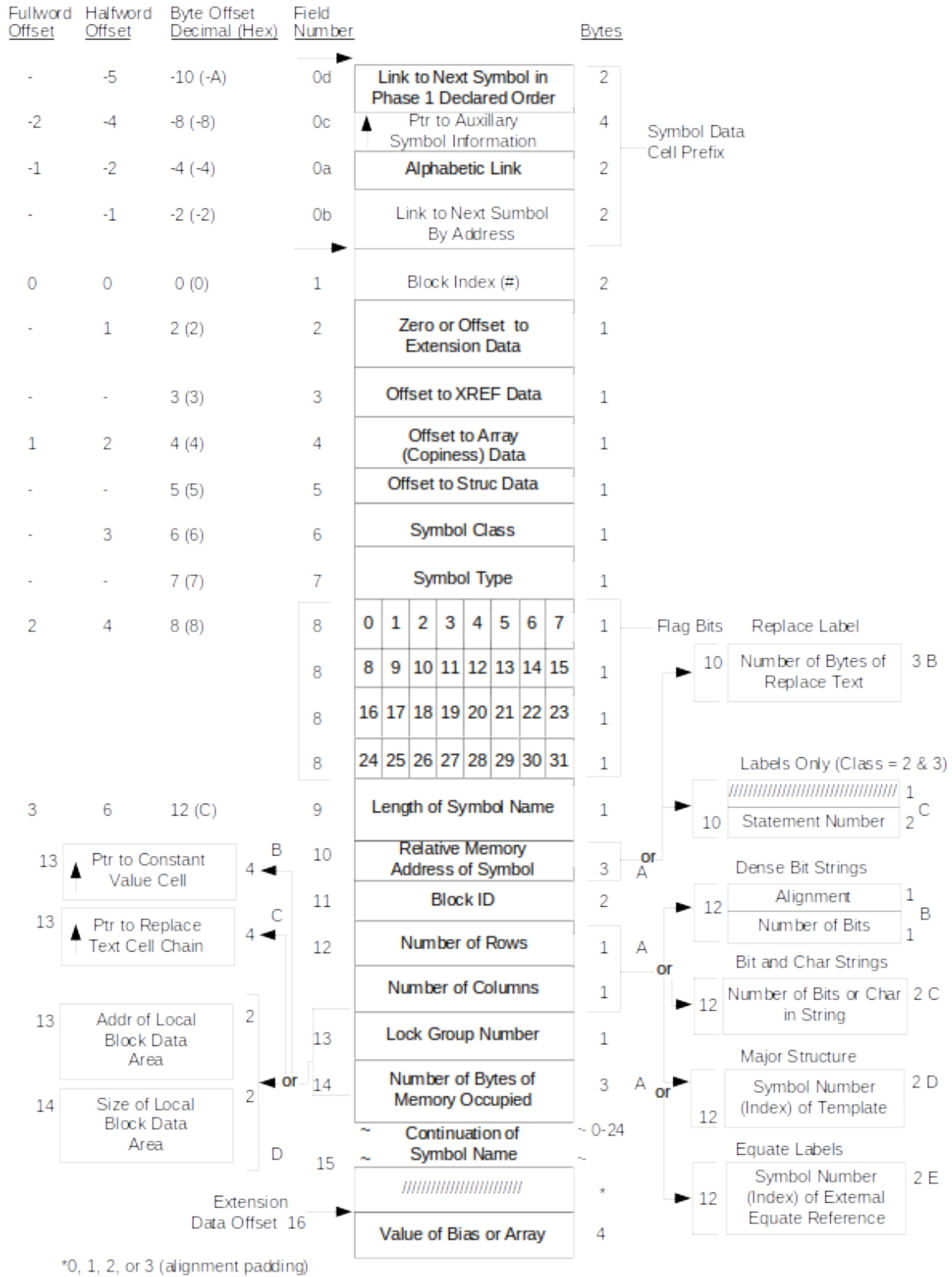


Figure 2-32 Symbol Data Cell (Part 1 of 3)

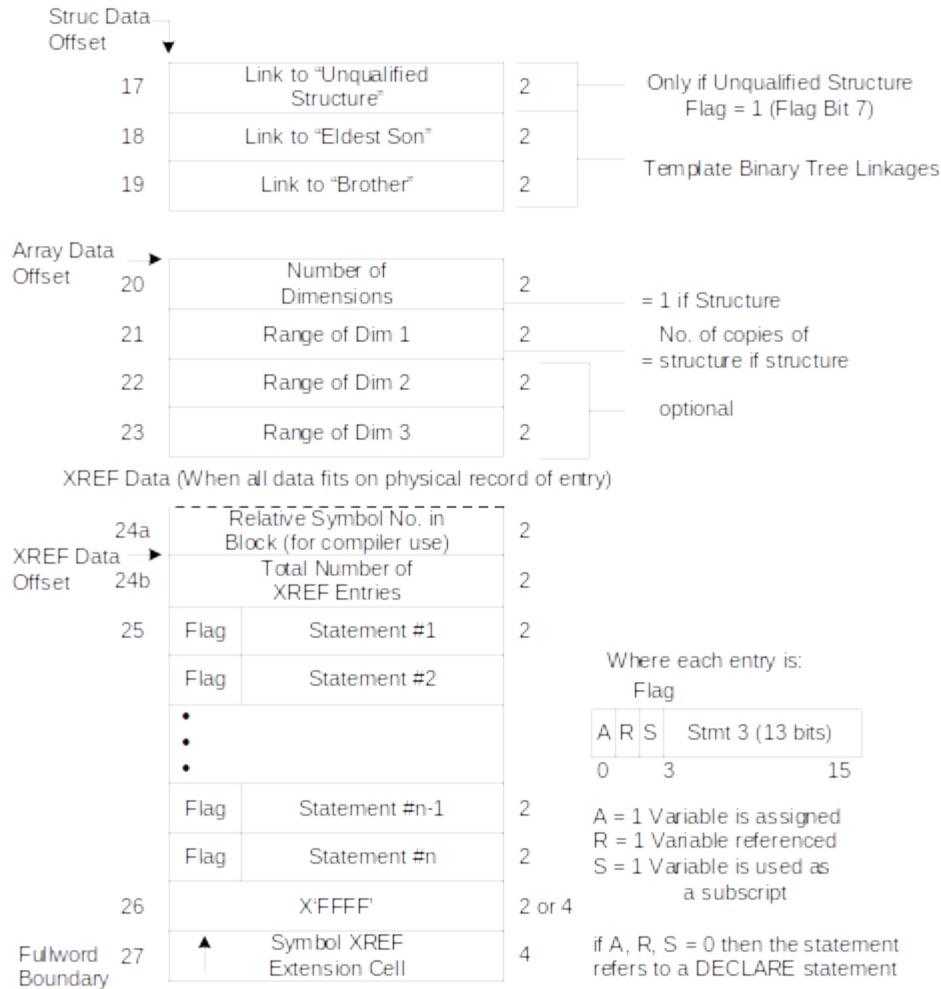


Figure 2-32 Symbol Data Cell (Part 2 of 3)

Symbol XREF Extension Cell (when data overflows physical record)



Figure 2-32 Symbol Data Cell (Part 3 of 3)

The cell provides linkages to structure templates or their members if the symbol is part of a HAL/S structure organization. In case of a symbol in a structure declaration (e.g., DECLARE X Y -STRUCTURE;), the symbol (i.e., "X") is defined as a structure type (SYMBOL TYPE = X'10'), is, classified as a qualified or unqualified structure name

(FLAG BIT 7 = 1 if unqualified and zero if qualified), and will have an index in Field 12 to the pertinent structure template.

In case of the template itself (i.e., "Y" in the above example), the entry will identify itself as a template by having FLAG BIT 6 set on. If an unqualified structure has been declared using the template, FLAG BIT 7 = 1 and Field 17, Link to Unqualified Structure, is set to point to the template's unqualified structure. The template contains an offset (Field 5) to the structure data and the structure data provides the initial link to the variables defined within the template (Field 18, Link to Eldest Son). Additionally, Field 0b of the Symbol Data Cell for the structure template is used as a list head for a linked list chaining all symbols belonging to the template in order of increasing template-relative address (Field 0b in these other template cells then identifies the next symbol in address order).

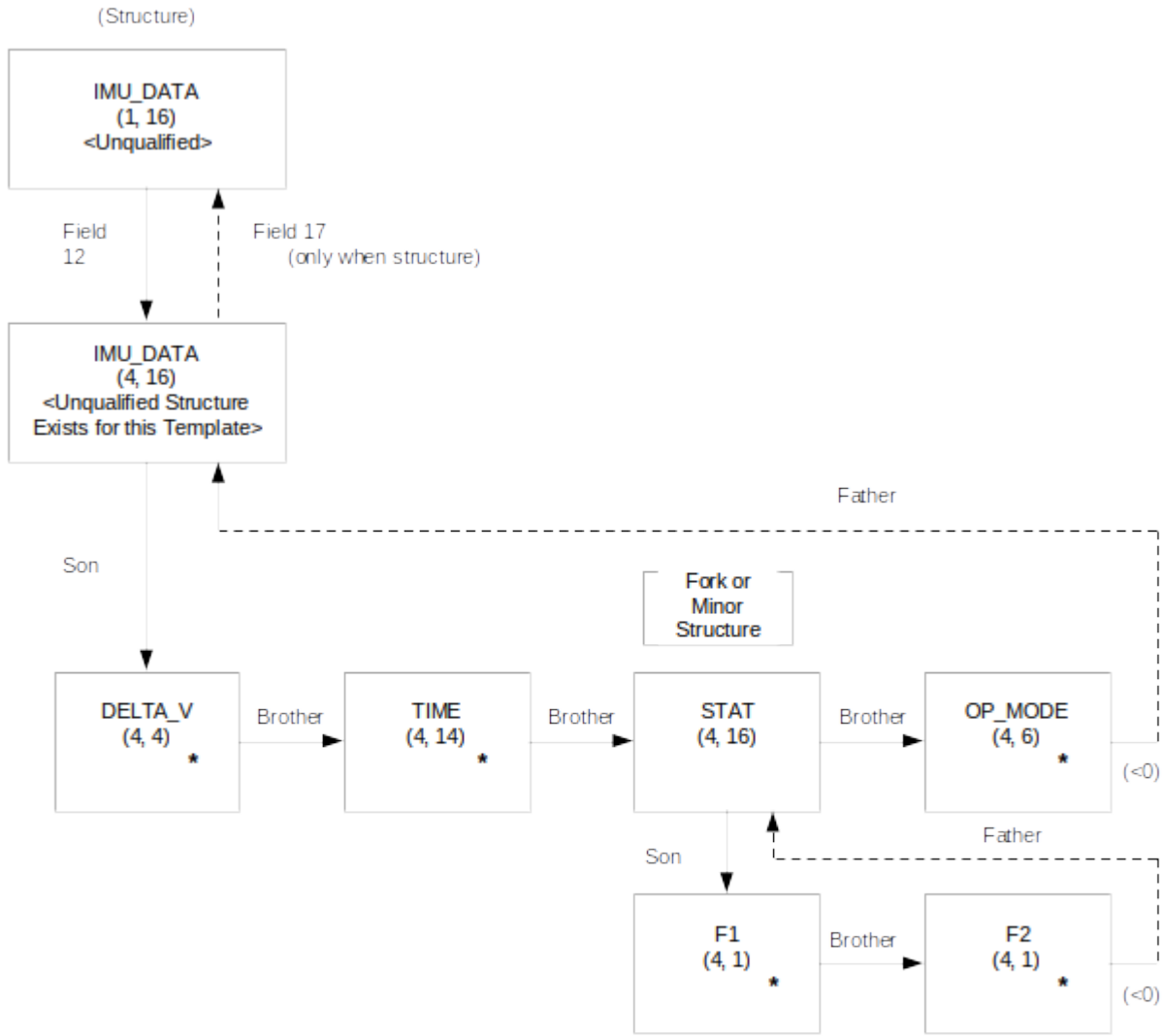
The variables within a template which are identified as belonging to a template by SYMBOL CLASS = 4 (Field 6), may be classified as qualified or unqualified, and refer to other templates or variables. If the reference is to another template SYMBOL TYPE = X"10", the Link to Eldest Son (Field 18) is zero, and Field 12 contains an index to the referenced template. Field 19 (link to Brother) may or may not contain a reference depending on the organization of the structure.

The variables in the template are organized in the form of a tree. The association of one variable with another can be determined by following the links supplied in Field 18 (Link to Eldest Son) and Field 19 (Link to Brother). Field 5 (Offset to Structure Data) will always contain an offset to the structure data (i.e., to the structure linkages). If the information is present in Field 18 a link exists to a lower level structure variable. If information is present in Field 19, a link exists to a following variable at the same structure level. However, if no following variables exist at the same level, a negative link (2's complement) back to the nearest parent appears in Field 19. If the variable belongs to an unqualified structure, Field 17 (Link to Unqualified Structure) contains an index to the structure itself.

See Figure 2-33 on page 53 for an example of the brother/son linkages contained in a single HAL/S Template and Figure 2-33 on page 55 for an example of nested HAL/S Templates.

<u>EXAMPLE</u>	<u>SYMBOL</u>	<u>CLASS</u>	<u>TYPE</u>	<u>FLAGS</u>
STRUCTURE IMU_DATA:	IMU_DATA	4	16	TEMPLATE, UNQUALIFIED
1 DELTA_V VECTOR,	DELTA_V	4	4	
1 TIME INTEGER DOUBLE,	TIME	4	14	
1 STAT,	STAT	4	16	
2 F1 BOOLEAN,	F1	4	1	
2 F2 BOOLEAN	F2	4	1	
1 OP_MODE INTEGER;	OP_MODE	4	6	
DECLARE IMU_DATA	IMU_DATA	1	16	UNQUALIFIED
IMU_DATA=STRUCTURE;				

Figure 2-33 Structures and Templates for a Single Structure (Part 1 of 2)

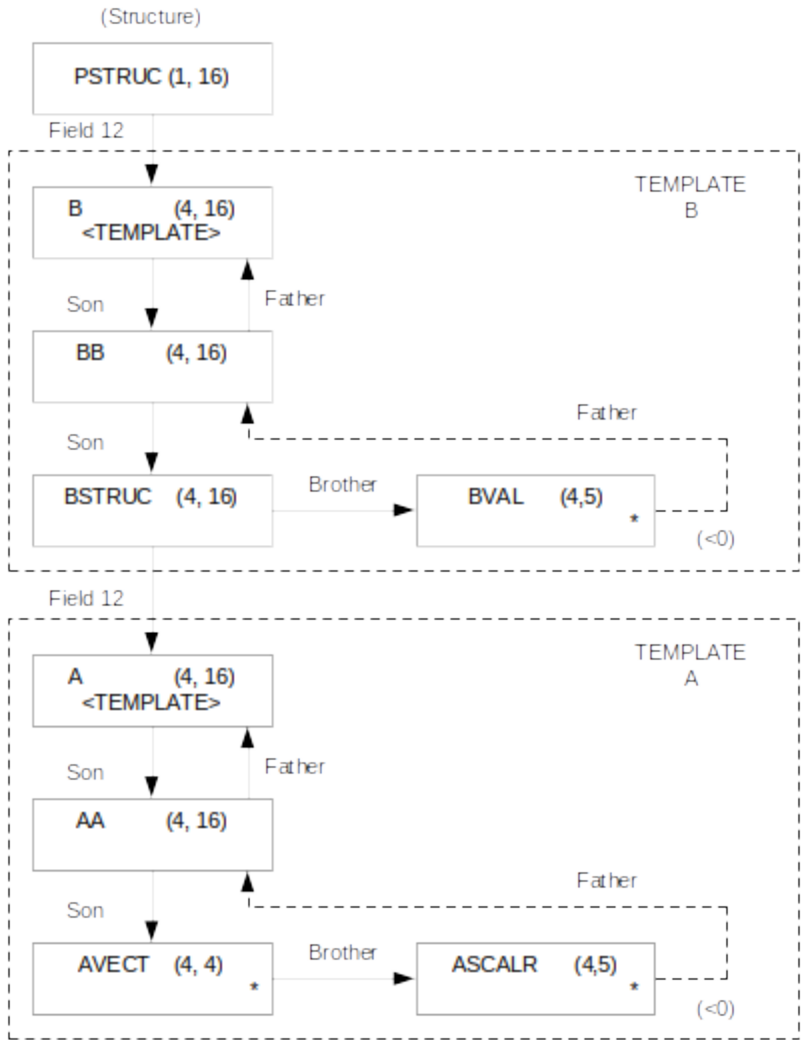


- Notes:
1. If a "Son" or "Brother" link is not shown, then it is 0
 2. Structure Terminals are denoted by an asterisk(*)
 3. Parenthesized numbers denote symbol class and type
 4. < > denotes symbol attribute Flags

Figure 2-33 Structures and Templates for a Single Structure (Part 2 of 2)

	<u>SYMBOL</u>	<u>CLASS</u>	<u>TYPE</u>	<u>FLAGS</u>
STRUCTURE A:	A	4	16	TEMPLATE
1 AA,	AA	4	16	
2 AVECT VECTOR,	AVECT	4	4	
2 ASCALR SCALAR;	ASCALR	4	5	
STRUCTURE B:	B	4	16	TEMPLATE
1 BB,	BB	4	16	
2 BSTRUC A-STRUCTURE,	BSTRUC	4	16	
2 BVAL SCALAR;	BVAL	4	5	
DECLARE PSTRUC B-STRUCTURE;	PSTRUC	1	16	

Figure 2-34 Structures and Templates for Nested Structures (Part 1 of 2)



- Notes:
1. If a "Son" or "Brother" link is not shown, then it is 0
 2. Structure Terminals are denoted by an asterisk(*)
 3. Parenthesized numbers denote symbol class and type
 4. <> denotes symbol attribute Flags

Figure 2-34 Structures and Templates for Nested Structures (Part 2 of 2)

The statement cross-reference information for structure templates and terminals contains the references for all structures using that template. In addition, only those nodes/terminals explicitly specified in the HAL/S source code actually contain the cross-reference for that statement (see Figure 2-35 on page 57 for more information). It should also be noted that any structure information is not propagated to other levels (e.g., symbol flag information is not propagated from the structure to the template terminals).

<u>ISN</u>		<u>XREF</u>
100	STRUCTURE A:	A (0, 100) (2, 101) (2, 102)
100	1 AA,	AA (0, 100)
100	2 AVECT VECTOR,	AVECT (0, 100) (4, 103) (4, 104)
100	2 ASCALR SCALAR;	ASCALR (0, 100)
101	DECLARE A1 A-STRUCTURE;	A1 (0, 101) (4, 103)
102	DECLARE A2 A-STRUCTURE;	A2 (0, 102) (4, 104)
103	A1. AVECT = 0;	
104	A2. AVECT = 0;	

XREF: (n,ISN) where n is a combination of the following:

0- DECLARE	2- REFERENCE
1- SUBSCRIPT	4- ASSIGN

Figure 2-35 Structure Symbol Cross-Reference Information (Part 1 of 2)

<u>ISN</u>		<u>XREF</u>
100	STRUCTURE A:	A (0, 100) (2, 101) (2, 102)
100	1 AA,	AA (0, 100) (2, 104) (4, 104) (4, 106)
100	2 BB,	BB (0, 100)
100	3 BVECT VECTOR,	BVECT (0, 100) (4, 105) (4, 106)
100	2 BVAL SCALAR;	BVAL (0, 100) (4, 103)
101	DECLARE A A-STRUCTURE;	A (0, 101) (4, 103) (4, 104) (4, 105) (4, 106)
102	DECLARE AB A-STRUCTURE;	AB (0, 102) (2, 104)
103	BVAL = 0;	
104	A.AA = AB.AA;	
105*	BVECT = 0;	
106*	A.AA.BB.BVECT = 0;	

*Equivalent assignment which generates different cross-reference information

Figure 2-35 Structure Symbol Cross-Reference Information (Part 2 of 2)

Figure 2-36 on page 58 illustrates the different linked lists involving the Symbol Data Cells. The field numbers that make up the linked list are also provided. Note that the #R linked list cannot exist unless the SDL Flag (Bit 12 of the flags in the Directory Root Cell) is a zero. #R Remote data is prohibited if the SDL Flag is a one.

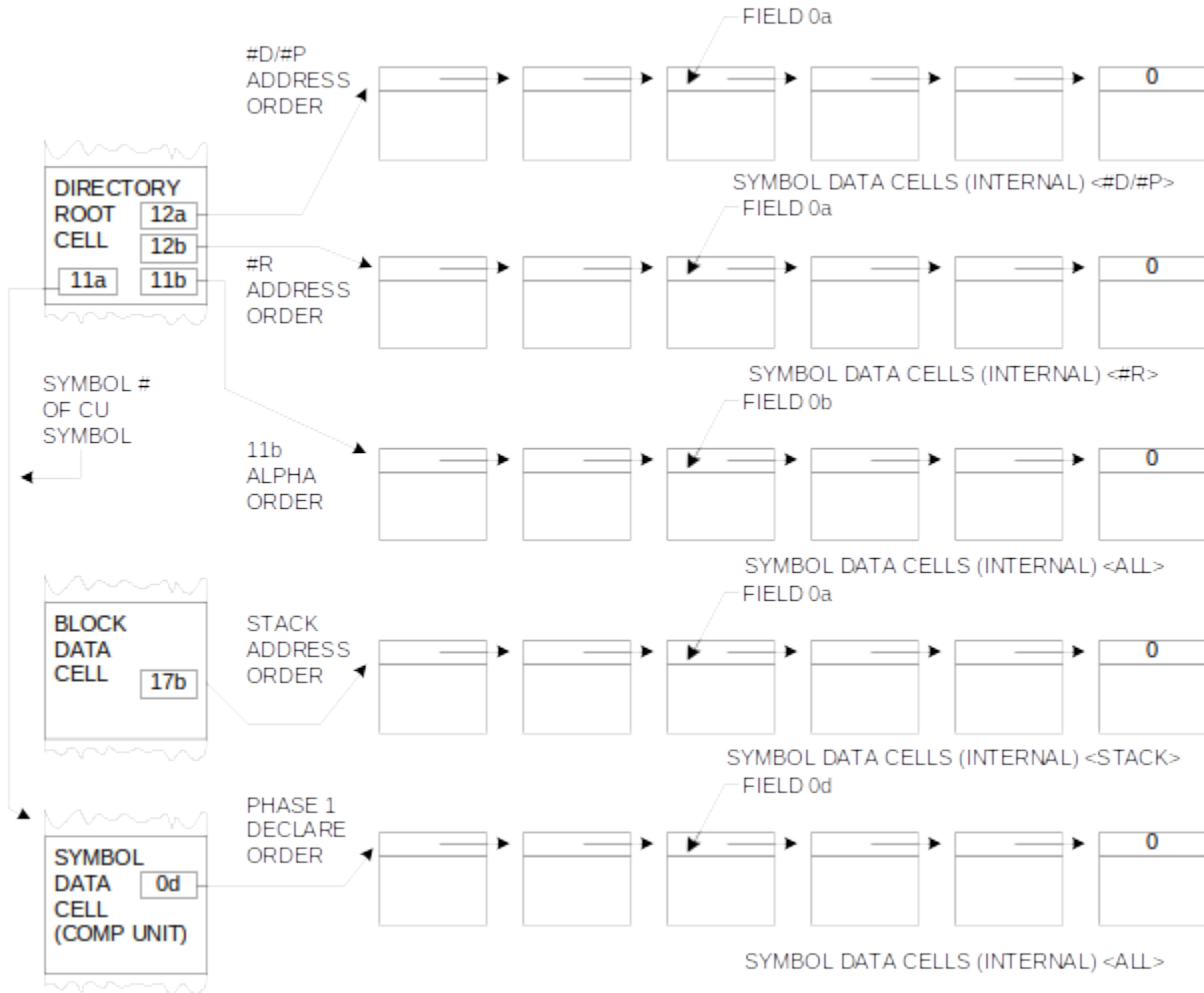


Figure 2-36 Symbol Data Cell Linked Lists

The meaning of the fields of the Symbol Data Cell are as follows:

Field No. Description

- 0a Index into the Symbol Index Table of next alphabetic symbol. Only internal symbols (no COMPOOL or EXTERNAL Procedure/Function symbols) are included in this chain of symbols. The chain terminates with a zero. The initial symbol of the chain is defined by Field 11b of the Directory Root Cell.
- 0b Within the following categories all internal symbols are linked by address:

#D data #P data	}	The List Header is defined by Field 12a of the Directory Root Cell. The last entry is zero.
--------------------	---	---

#R data	}	The List Head is defined by Field 12b of the Directory Root Cell. The last entry is zero. #R data is prohibited if the SDL flag is a one.
Stack data	}	The List Head is defined by Field 17b of the HAL/S Block Data Cell.

All internal and external symbols defined in a structure template are address linked. For these categories the field contains an index in the Symbol Index Table of the next symbol in the address chain.

- 0c The pointer to the Auxiliary Symbol Information is always present. Bit 29 of the Flag Bits (Field 8) is set to indicate the presence of data in Field 0c.

For Equate External labels, the Auxiliary Symbol Information Pointer (ASIP) refers to a Variable Reference Cell (see Section 2.2.2.2.7, “Variable Reference Cell” on page 105) describing the variable being equated to. This cell will be present only when Field 12 of the Symbol Data Cell is inadequate to describe the HAL/S variable (i.e., when the variable is in a qualified structure or is subscripted).

For procedure and function names, the ASIP points to a Procedure/Function Formal Parameter Cell (see Section 2.2.2.2.4.5, “Procedure/Function Formal Parameter Cell” on page 76.)

Last, the ASIP provides information about the initialization of NAME variables. For simple NAME variables, the ASIP points to a Variable Reference Cell (see Section 2.2.2.2.7, “Variable Reference Cell” on page 105) describing the variable initially pointed to by the NAME variable. For NAME variables which are structure terminals, the ASIP in the Symbol Data Cell for the major structure name points to a linked list of NAME Terminal Initialization Cells (see Section 2.2.2.2.4.6, “Name Terminal Initialization Cell” on page 78) with one cell for each NAME variable in the structure template. If the simple NAME variable or the structure is not initialized, the ASIP is absent.

Field No. Description

- 0d This field defines a linked list of internal symbols in the order in which they were entered into the symbol table (in HAL/S Compiler Phase 1 order). The root of this chain is the symbol number of the compilation unit.
- 1 Index Number of the HAL/S Block in the Block Index Table. This index serves two primary purposes: 1) to identify the block in which the symbol is defined, 2) to provide a reference to the CSECT name of a COMPOOL symbol in the Block Index Table so that, in conjunction with the relative memory address of the symbol (Field 10), an actual address can be

determined.

- 2 Offset within the cell to the Extension Data (at Field 16). If Extension Data does not exist, this field is zero. Otherwise, this cell extension contains Bias or Array data.
- 3 Offset within the cell to the statement cross-reference data, array dimension
- 4 data, and structure data, respectively. If the corresponding data is not
- 5 present, then the offset is zero.
- 6 Symbol Class and Symbol Type identify the classes of symbols and their
- 7 attributes. The assigned codes are as follows:

6,7
(Cont'd)

<u>Class</u>	<u>Type</u>			
	Decimal	Hex		
1 Variable	1	1	BIT (16-bit) (halfword)	
	2	2	CHARACTER	
	3	3	MATRIX (SP)	
	4	4	VECTOR (SP)	
	5	5	SCALAR (SP)	
	6	6	INTEGER (SP) (halfword)	
	9	9	BIT (32-BIT) (fullword)	
	10	A	Unused	
	11	B	MATIRX (DP)	
	12	C	VECTOR (DP)	
	13	D	SCALAR (DP)	
	14	E	INTEGER (DP) (fullword)	
	16	10	STRUCTURE	
	17	11	EVENT Variable (1 bit right-justified in a halfword)	
	2 Label	1	1	PROGRAM
		2	2	PROCEDURE

Field No. Description

6,7
(Cont'd)

<u>Class</u>	<u>Type</u>		
	Decimal	Hex	
2 Label (Cont'd)	3	3	FUNCTION (see class 3)
	4	4	COMPOOL
	5	5	TASK
	6	6	UPDATE
	7	7	Statement
	8	8	EQUATE

	9	9	REPLACE Label
3 Function	1	1	BIT (16) (halfword)
	2	2	CHARACTER
	3	3	MATRIX (SP)
	4	4	VECTOR (SP)
	5	5	SCALAR (SP)
	6	6	INTEGER (SP) (halfword)
	9	9	BIT (32-bit) (fullword)
	10	A	Unused
	11	B	MATRIX (DP)
	12	C	VECTOR (DP)
	13	D	SCALAR (DP)
	14	E	INTEGER (DP) (fullword)
	16	10	STRUCTURE
4 Template	1	1	BIT (16-bit) (halfword)
	2	2	CHARACTER
	3	3	MATRIX (SP)
	4	4	VECTOR (SP)
	5	5	SCALAR (SP)
	6	6	INTEGER (SP) (halfword)
	9	9	BIT (32-bit) (fullword)
	10	A	Unused
	11	B	MATRIX (DP)
	12	C	VECTOR (DP)
	13	D	SCALAR (DP)
	14	E	INTEGER (DP) (fullword)

Field No. Description

6,7 (Cont'd)	<u>Class</u>	<u>Type</u>	
	4 Template (Cont'd)	Decimal 16	Hex 10
			Template Flag = 1 Template Template Flag = 0 Minor Structure a) If Field 18 ≠ 0, it points to first terminal. b) If Field 18 = 0, Field 12 points to next template.
		17	1
			EVENT Variable (1 bit right-justified in a halfword)
	5 Template Label	1	1
		2	2
		3	3
		4	4
		5	5
			PROGRAM _____ _____ _____ TASK

The format of the character string (Type 2 in classes 1, 3, and 4) is as follows:

Byte 0 1 2

Max Ct	Char Ct	Characters
-----------	------------	------------

where:

Max Ct is the maximum number of characters in the string, and

Char Ct is the current number of characters in the string.

The characters are blocked into halfwords such that if the number of characters is odd, the last halfword will contain a blank pad character.

- 8 The Flag Bits define the characteristics of the symbol. In addition, they identify Stack and NAME variables and supply information about the use of the variable in a structure. The Flag Bits are as follows:

<u>Bit</u>	<u>Meaning When Set</u>
0	COMPOOL Flag

<u>Field No.</u>	<u>Description</u>	<u>Meaning When Set</u>
8 (Cont'd)	<u>Bit</u>	
	1	Input Parameter
	2	Assign Parameter
	3	TEMPORARY
	4	AUTOMATIC
	5	NAME Variable (the NULL NAME variable pointer is a halfword of zeros)
	6	Template Flag (if on, represents template)
	7	Unqualified Structure Flag
	8	REENTRANT Flag (for the block and its variables)
	9	DENSE Flag
	10	CONSTANT Flag
	11	ACCESS Flag
	12	Indirect Flag
		The Indirect Flag is ON only for parameters which refer to:
		<ul style="list-style-type: none"> • Aggregates • Assign parameters • Process event variables, which also have the Latched Flag (bit 13) ON.
	13	LATCHED Flag (for EVENT variables)
	14	LOCKED Flag

} Stack Variables

<u>Field No.</u>	<u>Description</u>
8 (Cont'd)	<p><u>Bit</u> <u>Meaning When Set</u></p> <p>15 REMOTE Flag. This flag, in conjunction with the fields listed below, has the following meanings:</p> <p>(1) If bit 5 (NAME Variable) is ON, then the variable is a 32-bit pointer.</p> <p>(2) If bit 24 (INCLUDED REMOTE) is ON and Field 6 (symbol class) = 2 (label) and Field 7 (symbol type) = 4 (COMPOOL), then the label is a COMPOOL that is included remote.</p> <p>(3) If bit 24 (INCLUDED REMOTE) is ON and Field 6 (symbol class) = 2 (label), then the symbol lives in a remote #P (COMPOOL) CSECT.</p> <p>(4) If bit 5 and bit 24 are both OFF, then the variable lives in a #R CSECT. #R data is prohibited if the SDL flag is a one.</p> <p>16 Non-zero Bias Flag</p> <p>17 INITIAL Flag (for variable being initialized)</p> <p>18 RIGID</p> <p>19 Literal. Variable is in literal pool and Field 10 is zero. Field 14 is a pointer to a Constant Value Cell. The variable is consequently inaccessible to diagnostics. The CONSTANT Flag is set ON for literals.</p> <p>20 EXTERNAL. Variable belongs to an EXTERNAL block (e.g., an included COMPOOL).</p> <p>21 Stack Variable (variable is in a stack). A variable is a stack variable if it is an input parameter, an assigned parameter, a TEMPORARY variable, or an AUTOMATIC variable in a REENTRANT PROCEDURE.</p> <p>22 Local Block Data. This flag is set ON if the block contains a reference to Local Block Data. When this flag is ON, Fields 13 and 14 of the Symbol Data Cell contain address and size information.</p>

<u>Field No.</u>	<u>Description</u>	
8 (Cont'd)	<u>Bit</u>	<u>Meaning When Set</u>
	23	EQUATE. When this flag is ON, the symbol is referenced in an EQUATE statement.
	24	INCLUDED REMOTE Flag. Indicates that the variable lives in a remote #P (COMPOOL) CSECT.
	25	EXCLUSIVE. This flag identifies the symbol as the name of a block that is EXCLUSIVE.
	26	Unused
	27	Misc. Name Flag. Indicates that the symbol is the template of a structure with NAME terminals, or that the symbol is a variable pointed to by a NAME variable.
	28	Macro Arg. Flag. Indicates that the symbol is the name of a REPLACE Macro that has arguments.
	29	ASIP Flag. Indicates the presence of Filed 0c, the Auxiliary Symbol Information Pointer.
	30	Unused
	31	This symbol is a BIT variable which is assigned from a multi-instruction masking operation.
9	Total number of characters in the symbol name. If this value is 8 or less, then all symbol characters are contained within the Symbol Index Table and thus Field 15 is non-existent. If Field 15 exists at all, then its length, in bytes, is equal to this value minus 8.	
10	A	Relative memory address (in terms of halfwords) of the symbol within a data CSECT or stack space.
	B	If the symbol is a REPLACE Label, this field contains the number of bytes in the SDF representation of the REPLACE Text.
	C	If the symbol is a Label (as specified by Field 6), then the address is replaced by the Internal Statement Number corresponding to that label.

<u>Field No.</u>	<u>Description</u>
11	The Block ID is a compiler generated internal code. If the symbol's data is stored on the stack, the Block ID identifies the stack in which the symbol's data exists during execution. If a specified stack symbol's data is not in the current stack space, one can retrieve the data for the symbol at another level by threading backwards through the stack so long as the symbol has proper scope.
12	<p>A For matrices and vectors, the first half of this field contains:</p> <ul style="list-style-type: none"> a) The number of rows if the symbol is a matrix. b) The value one if the symbol is a vector. <p>The second half of this field contains:</p> <ul style="list-style-type: none"> a) The number of columns if the symbol is a matrix. b) The number of components if the symbol is a vector. <p>B For DENSE bit strings this field contains:</p> <ul style="list-style-type: none"> a) In the first byte an alignment factor as follows: <ul style="list-style-type: none"> 0 - right aligned and has leading zeroes. n - where n is greater than zero and less than X"FF", n indicates the number of bit positions the string must be shifted to right align the string: leading bits must be masked. X"FF" - the string is right aligned; however, the leading bits must be masked. b) The second byte specifies the number of bits contained in the string. <p>C For bit and character strings the field contains:</p> <ul style="list-style-type: none"> a) The number of bits if the symbol is a bit string. b) The number of characters if the symbol is a character string. If negative, the field then indicates that the number of characters is unknown. If this field is negative and Field 16 is zero (the case where Field 16 is non-zero is discussed in the notes for Field 16), the character string will be a "*" character string. Information about the character string will be found in the stack space at the relative address specified in Field 10 as follows:

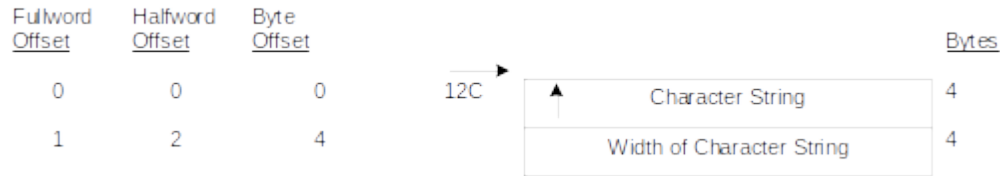


Figure 2-37 Stack Variable Character String Format

Field No.	Description
12 (Cont'd)	D For a structure, the field contains the symbol number (index to an entry in the Symbol Index Table) of the structure template (SYMBOL TYPE = X'10').
	E For EQUATE labels, the field contains the index into the Symbol Index Table of the symbol being equated to.
13	A Lock Group Number. A hex "FF" signifies LOCK (*).
	A The number of halfwords occupied by the symbol. In case of STRUCTURE, the size includes all copies. In case of a NAME variable, the size of the symbol pointed to.
14	
13	B When the LITERAL and CONSTANT flags are set (Field 8, bits 10 and 19), the symbol is a CONSTANT variable and this field is a 4-byte pointer to a Constant Value Cell which is described in Figure 2-41 on page 72 and Figure 2-42 on page 73.
13	C When the symbol is a REPLACE label, this field is a 4-byte pointer to the Head of a Chain of Replace Text Cells which are described in Section 2.2.2.2.4.4, "Replace Text Cells" on page 73.
13	D The relative address of the Local Data Area within the #D CSECT (halfwords). Fields 13 and 14 refer to Local Block Data only if Bit 22 of Field 8 is ON.
	D The length of the Local Block Data area (i.e., 2 or 5 halfwords).
15	Remainder of symbol name. This can be from 0 to 24 characters.

<u>Field No.</u>	<u>Description</u>
16	<p>Except for an array of "*" character strings, a data item may not be pointed to directly whenever the HAL/S Compiler performs variable indexing or references data using an indirect pointer (e.g., Parameter Passing or Name Variable manipulation). The HAL/S Compiler often points to an address somewhere before the beginning of the actual data. The difference between the address ahead of the data and the address at the beginning of the actual data is known as the Bias or Offset. This method is used since it is more efficient to set the database register to point to a fictitious 0th item of an aggregate (matrix, vector, multi-copy structure, or array).</p>

For an array of "*" character strings, this field contains an Arrayness Value. This situation is indicated by a negative character count in Field 12 and a value contained in this field. In this case, additional information about the character strings (see Figure 2-38 on page 69) can be found in the Stack Space specified in Field 10. The true Bias is calculated by multiplying the Arrayness Value and the width specified in the Stack space. The resulting Bias will be in halfwords.

Figure 2-39 on page 70 shows the algorithm used to calculate the Bias Factor.



Notes:

1. This pointer can be a pointer to another indirect pointer
2. Includes header information

Figure 2-38 Array of Character Strings

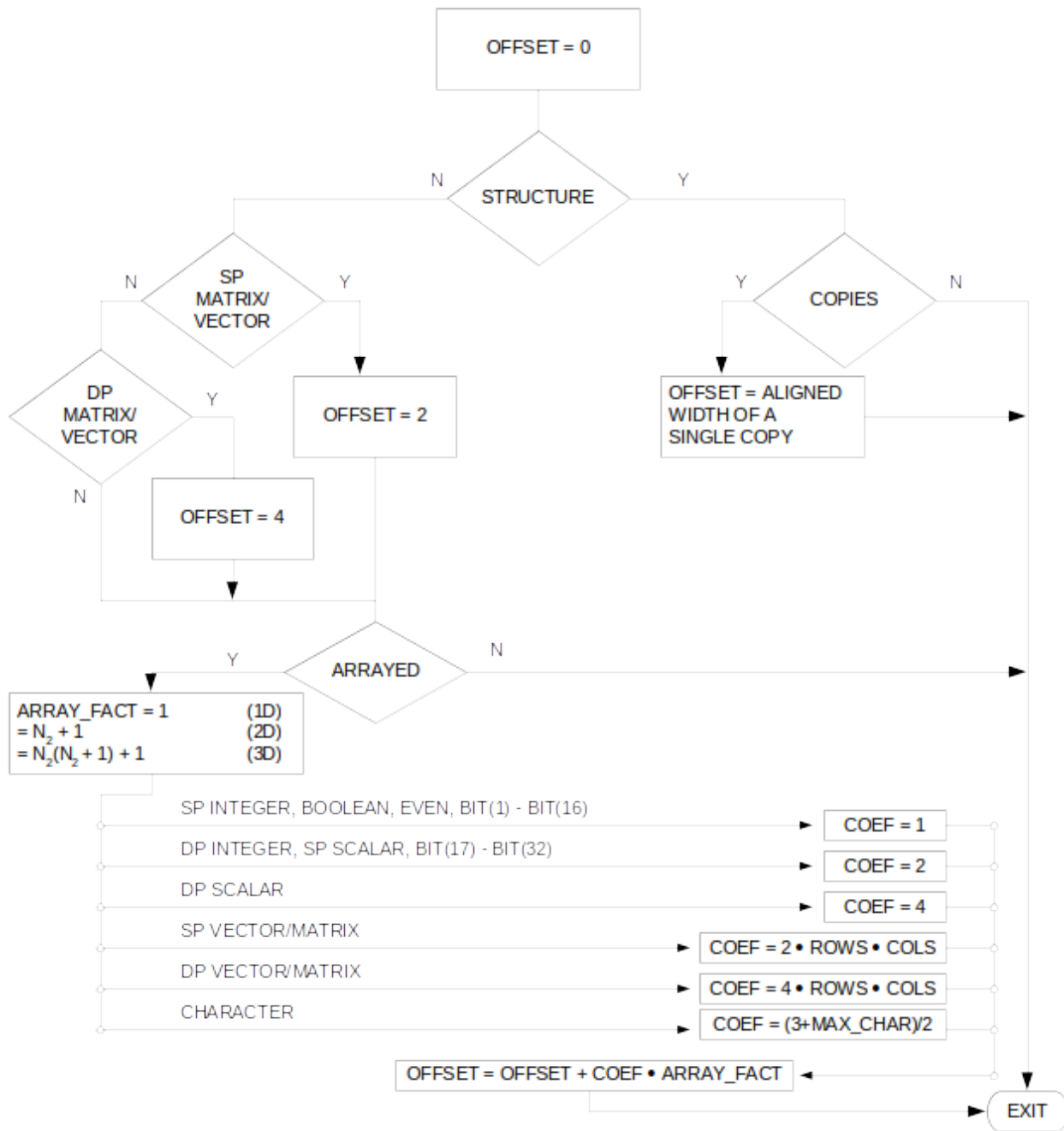


Figure 2-39 Algorithm for Calculating the Bias Factor

<u>Field No.</u>	<u>Description</u>
17 18 19	Structure Links (see preceding text for explanation).
20	If the symbol is an array, this field contains the number of dimensions. If the symbol is a structure, this field contains a one.
21 22 23	Fields 21-23 are the ranges of each of the dimensions of the array. If the symbol is a structure, Field 21 contains the number of copies of the structure and fields 22 and 23 do not exist.
24a	Relative symbol number in block (compiler only).
24b	The total number of statement cross reference entries.
25	Indexes to the Statement Index Table. They identify the statements in which the symbol is modified, referenced, used as a subscript, or declared.
26	This field is present only when all of the symbol's cross-reference data does not fit in the same physical SDF record as the beginning part of the Symbol Data Cell. This field varies in length (so that Field 27 may start on a full word boundary) and contains either 2 or 4 bytes of hex "FF".
27	Like Field 26 above, this field is present only when all of the cross-reference data will not fit in the same record. The field contains an SDF pointer to the Symbol XREF Extension Cell.

2.2.2.2.4.3 Constant Value Cells

The Constant Value Cells contain data that was specified using the CONSTANT attribute in the symbol declaration to set an initial value. There are two types of Constant Value Cells: character strings and scalars/integers. The two types are discussed in the following sections.

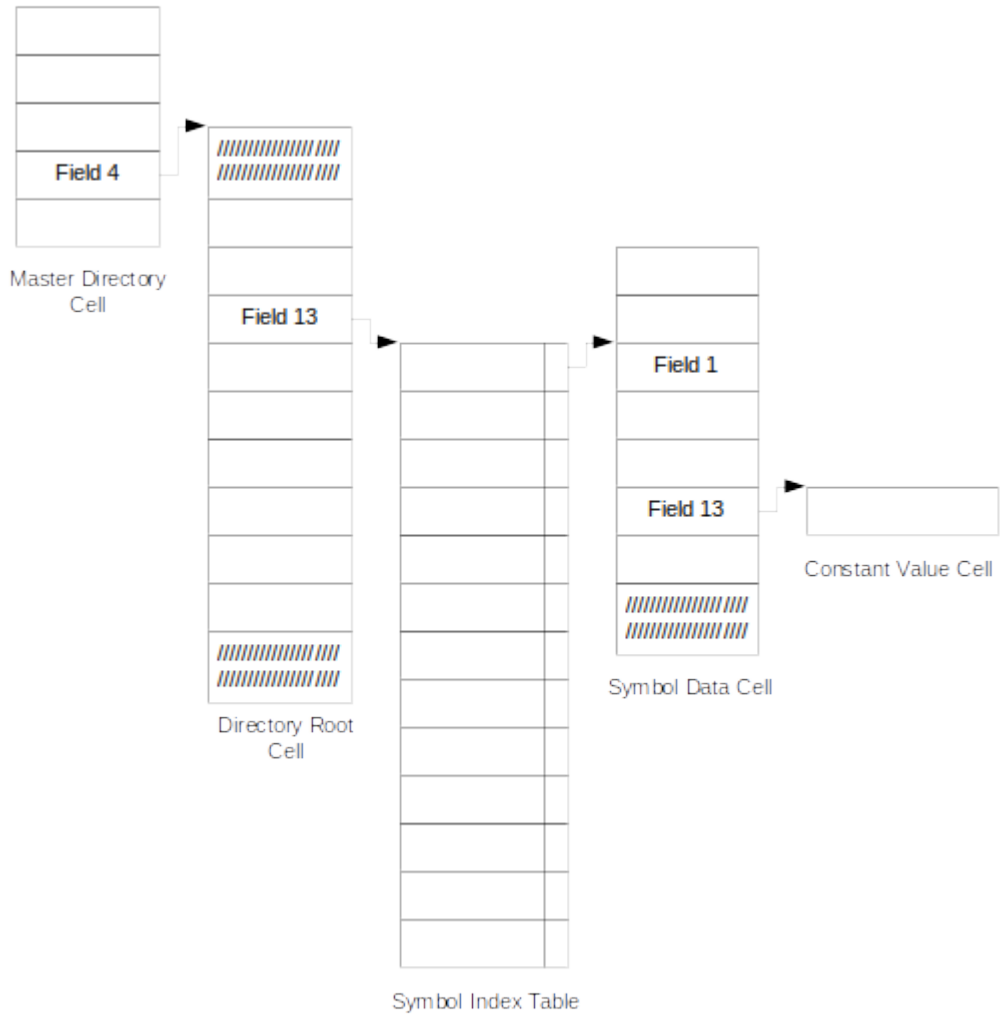


Figure 2-40 Constant Value Cell Overview

2.2.2.2.4.3.1 String Constant Value Cells

The String Constant Cell shown in Figure 2-41 on page 72 contains initialization data for character constants and literals. The format of this cell is described below:

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> No. of Bytes of Text - 1 </div>	1
-	-	1	2		<div style="border: 1px solid black; padding: 5px; display: inline-block;"> EBCDIC Text </div>

Figure 2-41 String Constant Value Cell

Field No. Description

- 1 This field contains the number of characters in the string minus 1.
- 2 This field contains the character data for the string.

2.2.2.2.4.3.2 Scalar/Integer Constant Value Cells

The Scalar/Integer Constant Value Cell shown in Figure 2-42 on page 73 contains initialization data for numeric symbols declared using the CONSTANT attribute. The format of this cell is described below.

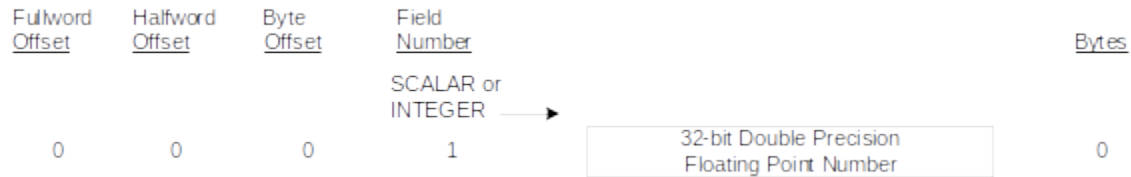


Figure 2-42 Scalar/Integer Constant Value Cell

<u>Field No.</u>	<u>Description</u>
1	This field contains the value of the constant stored as a double precision 64-bit floating point number.

2.2.2.2.4.4 Replace Text Cells

As shown in Figure 2-43 on page 74, the Replace Text Parameter Cell (see Figure 2-45 on page 75) is pointed to by Field 13 of the Symbol Data Cell; Field 1 of the Replace Text Parameter Cell, in turn, points to the Replace Text Macro Cell. If all of the Replace Text will not fit into one Replace Text Macro Cell, Field 1 then points to a list of subsequent Replace Text Macro Cells which will contain the remaining Replace Text. Figure 2-44 on page 74 not only shows the different types of Replace Text, but also shows the information generated for the different Replace Text Cells.

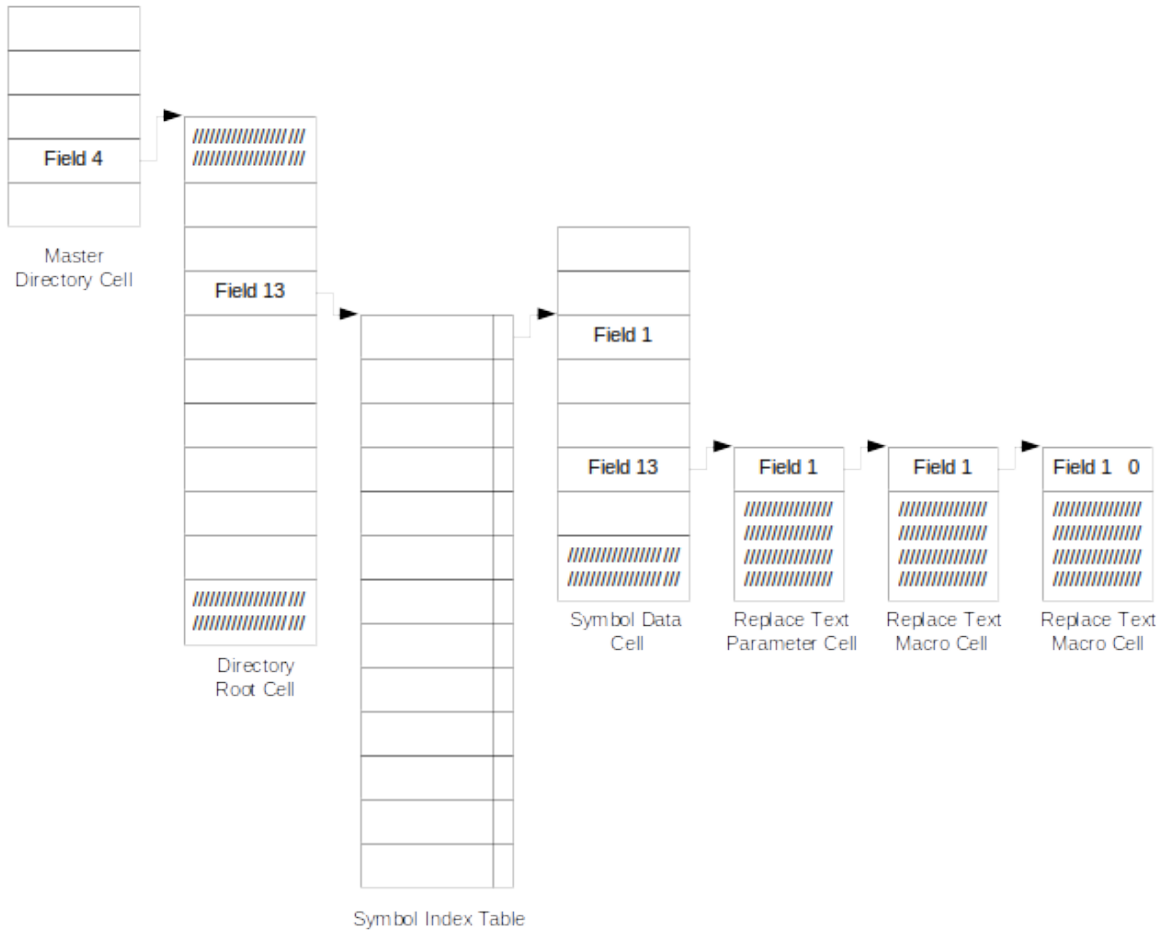


Figure 2-43 Replace Text Overview

<u>Type</u>	<u>Example</u>	<u>Result</u>	<u>Parameter Cell</u>	<u>Macro Text</u>
Simple Replace:	REPLACE N BY "4"; DECLARE V1 VECTOR(N);	DECLARE V1 VECTOR(4);	NONE	4
Parametric Replace:	REPLACE A(X,Y) BY "READ (X)Y";	--	"X","Y"	READ (X)Y
Replace Macros:	REPLACE TEST (A,B,C) BY "IF A THEN B ELSE C" TEST (P=0, S=1;, S=2;)	IF P=0 THEN S=1; ELSE S=2;	"A","B","C"	IF A THEN B ELSE C

Figure 2-44 Replace Text Examples

The Replace Text Parameter Cell is shown in Figure 2-45 on page 75.

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	↑ Pointer to the Next Replace Text Cell	4
1	2	4	2	- (#ARGS + 1)	2
-	3	6	3	No. of Blank Bytes	2
2	4	8	4	Pseudo-Descriptor	4
				⋮	
			4	Pseudo-Descriptor	4
			5	EBCDIC TEXT	As many as needed to represent the arguments

Figure 2-45 Replace Text Parameter Cell

The meanings of the fields are as follows:

Field No. Description

- 1 A link to the Replace Text Macro Cell (Figure 2-47 on page 76) in the chain. Subsequent cells contain the text of the macro.
- 2 This field indicates the number of Replace Macro Arguments. For a macro with no arguments, this field has the value X'FFFF'.
- 3 Argument text is stored in the SDF in a compressed format where a string of consecutive blanks is represented by two bytes: the first having the value X'EE', and the second byte containing the number of blanks minus one. The number of blank bytes is the difference between the number of bytes of compressed text in the SDF and the number of bytes of REPLACE text in the HAL/S source.
- 4 There is one pseudo-descriptor corresponding to each Replace Macro argument and their order corresponds to the order of the arguments in the macro invocation. Adding the address of the beginning of the cell to the pseudo-descriptor generates an XPL string descriptor which points to the Replace Macro argument's name in the text field (see Figure 2-46 on page 75).

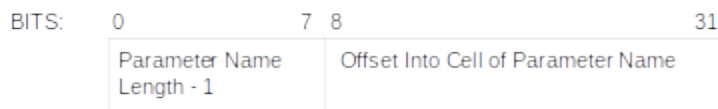


Figure 2-46 Replace Text Parameter Cell Pseudo Descriptor

Field No. Description

5 EBCDIC text containing the Replace Macro argument names.

The subsequent Replace Text Macro Cells have the following format:

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	↑	4
1	2	4	2		2
-	3	6	3		Up to 1000 Bytes

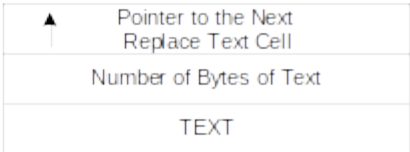


Figure 2-47 Replace Text Macro Cell

The meanings of the fields are as follows:

Field No. Description

- 1 A link to the next Replace Text Macro Cell in the chain. This field has the value X'00000000' for the last cell in the chain.
- 2 The number of bytes of text that follow.
- 3 Up to 1000 bytes of Replace Macro Text.

2.2.2.2.4.5 Procedure/Function Formal Parameter Cell

As shown in Figure 2-48 on page 77, the Formal Parameter Cell is referenced by the Auxiliary Symbol Information Pointer (field 0c) of the Symbol Data Cell (see Section , “Symbol Data Cell” on page 49) that corresponds to the name of the procedure or function. The cell indicates the formal parameters (i.e., those defined in the procedure or function header) associated with the procedure or function.

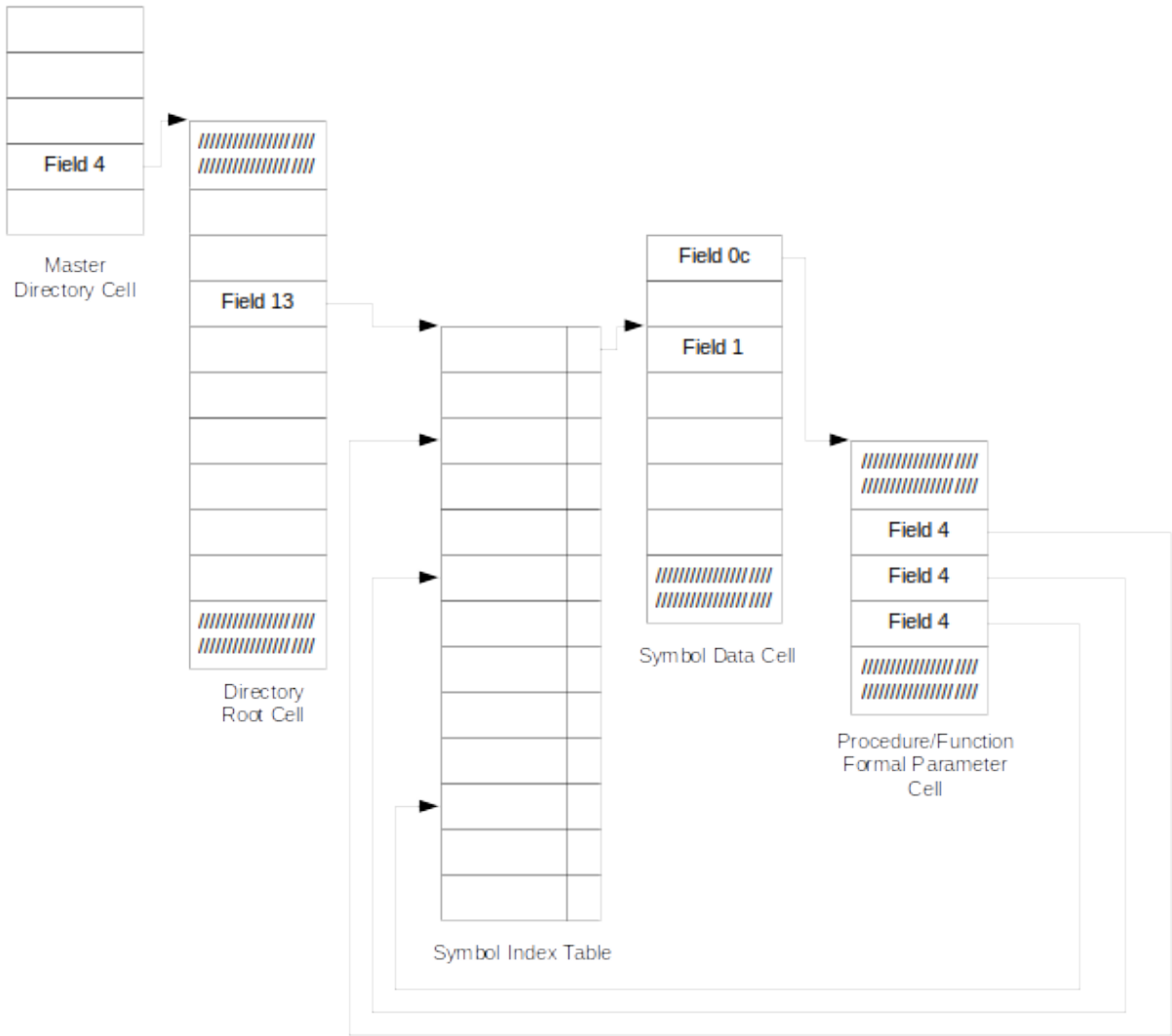


Figure 2-48 Procedure/Function Formal Parameter Cell Override

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	# Bytes in Cell	2
-	1	2	2	# of Parameters	1
-	-	3	3	# of Input Params	1
1	2	4	4	Parm # *	2
			4	Parm # *	2
			4	Parm # *	2
			4	• • •	
			4	Parm # *	2

* Index in Symbol Index Table

Figure 2-49 Procedure/Function Formal Parameter Cell

The meaning of the fields of the Formal Parameter Cell are as follows:

<u>Field No.</u>	<u>Description</u>
------------------	--------------------

- | | |
|---|--|
| 1 | The number of bytes in the cell. |
| 2 | The number of formal parameters defined in the procedure or function. |
| 3 | The number of formal parameters which are input parameters. |
| 4 | Indexes to corresponding Symbol Data Cells for each parameter. The parameters are listed in the order of their occurrence in the block (i.e., procedure or function) header. |

2.2.2.2.4.6 Name Terminal Initialization Cell

The Name Terminal Initialization Cell describes the initial pointer value or values of a name structure terminal. The cell contains a complete reference to the terminal name including the structure qualifiers for any nested structures. Unless the NAME initialization points to a simple variable (i.e., a symbol that is neither subscripted nor part of a structure), this cell is followed by a list of pointers to Variable Reference Cells (see Section 2.2.2.2.7, "Variable Reference Cell" on page 105) which describe a variable pointed to by one or more copies of the name terminal. All of the Name Terminal Initialization Cells associated with a particular structure are grouped into a linked list. As shown in Figure 2-50 on page 79, the head of the list is pointed to by the Auxiliary Symbol Information Pointer (field 0c) of the Symbol Data Cell (see Section , "Symbol Data Cell" on page 49) that corresponds to the name of the major structure.

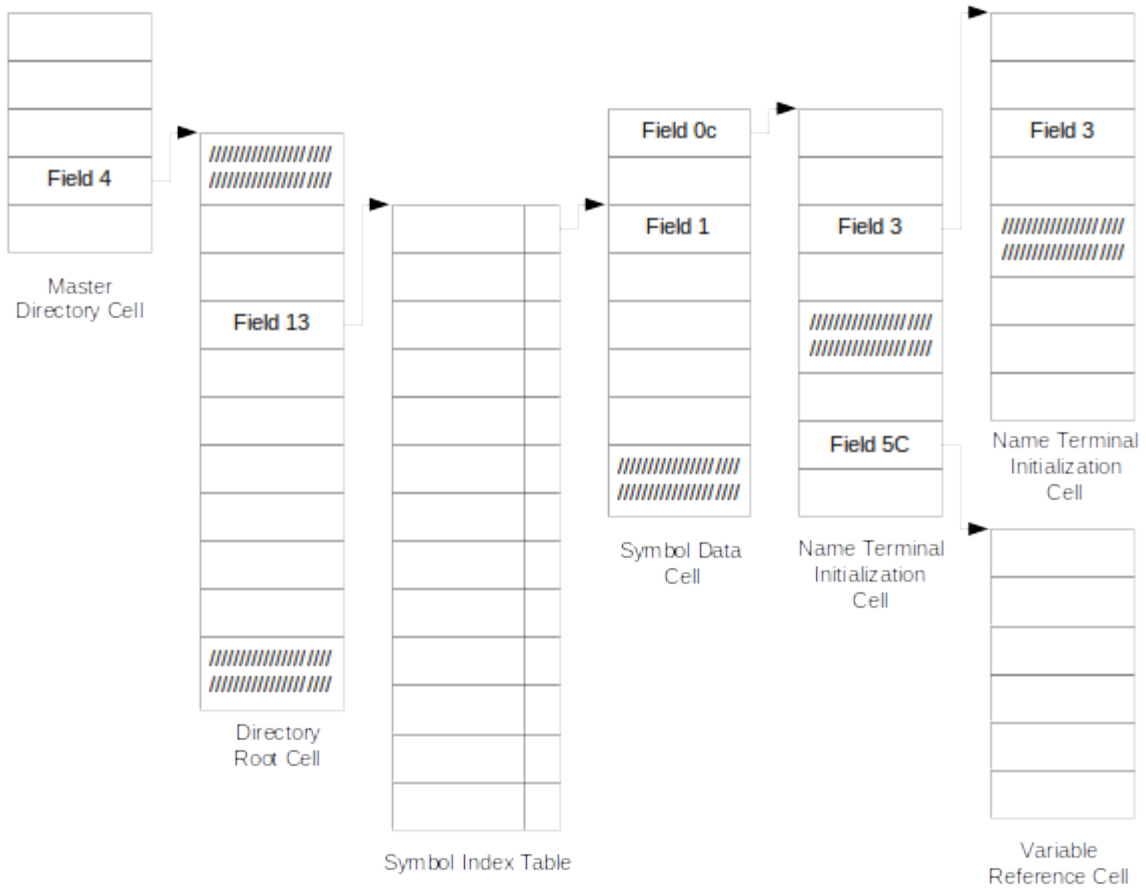


Figure 2-50 Name Terminal Initialization Cell Overview

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1		2
-	1	2	2		2
1	2	4	3	▲ Ptr to next Name Terminal Initialization Cell	4
2	4	8	4	Symbol #*	2
			4	Symbol #*	2
				• • •	
			4	Symbol #*	2
				//////////////////////////////////// //////////////////////////////////// ////////////////////////////////////	0 or 2 or bytes of padding so that the initial list is fullword aligned
			5	Initial List Word	4
			5	Initial List Word	4
				• • •	
			5	Initial List Word	4

* Indexes in Symbol Index Table

Figure 2-51 Name Terminal Initialization Cell

The meaning of the fields of the Name Terminal Initialization Cell are as follows:

<u>Field No.</u>	<u>Description</u>
1	Number of bytes in the cell.
2	Number of symbol indexes (field 4).
3	Pointer to the Name Terminal Initialization Cell for the next name terminal in the template.
4	Indexes into the Symbol Index Table (see the explanation of the Variable Reference Cell, field 4 on page 109).
5	The Initial List Words describe the initial pointer values of the various copies of the NAME terminal. The Initial List Words are grouped into fixed-length operators which contain either one or two words. The value of the first halfword of each operator determines its type. The formats of the operator types are as follows:

INITIAL POINTER VALUE OPERATOR

The Initial Pointer Value Operator contains a pointer to the Variable Reference Cell that describes the initial (first) NAME Pointer Value. If this operator occurs within Loop Operators (see below), then several copies of the NAME pointer may be initialized to this value.

<u>Field No.</u>		<u>Bytes</u>		
5A	X '00'	2		
5B	Copy No.	2		
5C	<table border="1"><tr><td>F l a g</td><td>↑ Symbol Index or Pointer to Variable Reference Cell</td></tr></table>	F l a g	↑ Symbol Index or Pointer to Variable Reference Cell	4
F l a g	↑ Symbol Index or Pointer to Variable Reference Cell			

Figure 2-52 Initial Pointer Value Operator

The sub-fields for the Initial Pointer Value Operator are listed below:

<u>Field No.</u>	<u>Description</u>
5A	This field identifies the Operation Operator Type as being an Initial Pointer Value Operator.
5B	This field contains the number of the first copy of the NAME terminal that is initialized to this value.
5C	When the 1-Bit Flag is Off: Field 5C points to a Variable Reference Cell which describes the Structure variable or subscripted variable referenced in a NAME Initialization (i.e., the symbol referenced in the initialization part of a Declaration).

When the 1-Bit Flag is On:

The last 16 bits of field 5C contain the Symbol Index of the Symbol being referenced by the NAME pointer. A Symbol referenced in this manner is a simple variable (i.e., a variable that is neither subscripted nor part of a structure).

INITIALIZATION LOOP START OPERATOR

The Initialization Loop Start Operator, along with its corresponding Initialization Loop End Operator, defines a list of initial NAME pointer values that are repeated.

<u>Field No.</u>		<u>Bytes</u>
5D	X'01'	2
5E	Nest Level	2
5F	Repetition Factor	2
5G	Loop Increment	2

Figure 2-53 Initialization Loop Start Operator

The sub-fields for the Initialization Loop Start Operator are listed below:

<u>Field No.</u>	<u>Description</u>
------------------	--------------------

- | | |
|----|---|
| 5D | This field identifies the Operation Operator Type as being an Initialization Loop Start Operator. |
| 5E | The Nest Level indicates the depth to which this loop is nested within other such loops and matches the Nest Level in the corresponding Initialization Loop End Operator. |
| 5F | The Repetition Factor indicates the number of times the enclosed operator(s) is to be repeated. |
| 5G | This field contains the Loop Increment. The Loop increment is added to the Copy number associated with each Variable Reference Cell every time the loop is repeated in order to generate all of the copies with that initial value. |

If an Initial Pointer Value Operator with a Copy number of n is in an Initialization Loop Start Operator with a Repetition Factor of x and a Loop Increment of y , then that Initial Pointer Value Operator applies to copies $n, n+y, n+2y, \dots, n+(x-1)y$.

INITIALIZATION LOOP END OPERATOR

The Initialization Loop End Operator marks the end of a repeated list of Initial Pointer Values that was begun by the Initialization Loop Start Operator.

<u>Field No.</u>		<u>Bytes</u>
5H	X '02'	2
5I	Nest Level	2

Figure 2-54 Initialization Loop End Operator

The sub-fields for the Initialization Loop End Operator are listed below:

<u>Field No.</u>	<u>Description</u>
------------------	--------------------

- | | |
|----|---|
| 5H | This field identifies the Operation Operator Type as being an Initialization Loop End Operator. |
| 5I | The Nest Level matches the Nest Level in the corresponding Initialization Loop Start Operator. |

END OF INITIALIZATION OPERATOR

This is the last operator in every cell. If the Extension Flag (Field 5K) is zero, then this operator also marks the end of the initialization data. If there is more initialization data than will fit in a SDF page, the Extension Flag (Field 5K) is set to one and the remaining part of the initialization list is contained in an Extension Cell located by the pointer (Field 5L) found in the second word of the operator. The initial list may be divided between any two operators; no single operator will be split across two cells. The Extension Cell (Figure 2-55 on page 83) has the same general format as the Name Terminal Initialization Cell except that fields 2, 3, and 4 do not appear.

<u>Field No.</u>		<u>Bytes</u>
5J	X '03'	2
5K	Extension Flag	2
5L	 Extension Cell Pointer	4

Figure 2-55 End of Initialization (Cell) Operator

The sub-fields for the End of Initialization Operator are listed below:

Field No. Description

- 5J This field identifies the Operation Operator Type as being an End of Initialization Operator.
- 5K In the event more initialization data exists than will fit in an SDF page, this flag is set ON and will be followed immediately by field 5L.
- 5L When field 5K is ON, this field will exist. The field contains an SDF pointer to the Name Terminal Initialization Extension Cell (Figure 2-56 on page 84).

NAME TERMINAL INITIALIZATION EXTENSION CELL

This Cell exists only when the Initialization data will not fit within a single SDF page. The field numbers correspond to the fields described for the regular Name Terminal Initialization Cell (Figure 2-51 on page 80).

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	# Bytes in Cell	2
-	1	2	Filler	////////////////////////////////////	2
1	2	4	5	Initial List Word	4
			5	Initial List Word	4
				• • •	
			5	Initial List Word	4

Figure 2-56 Name Terminal Initialization Extension Cell

2.2.2.2.5 Statement Data Structures

The Statement Data Structures consist of the Statement Index Table, Executable Statement Data Cells, Declare Statement Data Cells, Expression Variables Cells, and Procedure/Function Invocation Cells. These cells provide information about the statements and the means by which this data can be addressed.

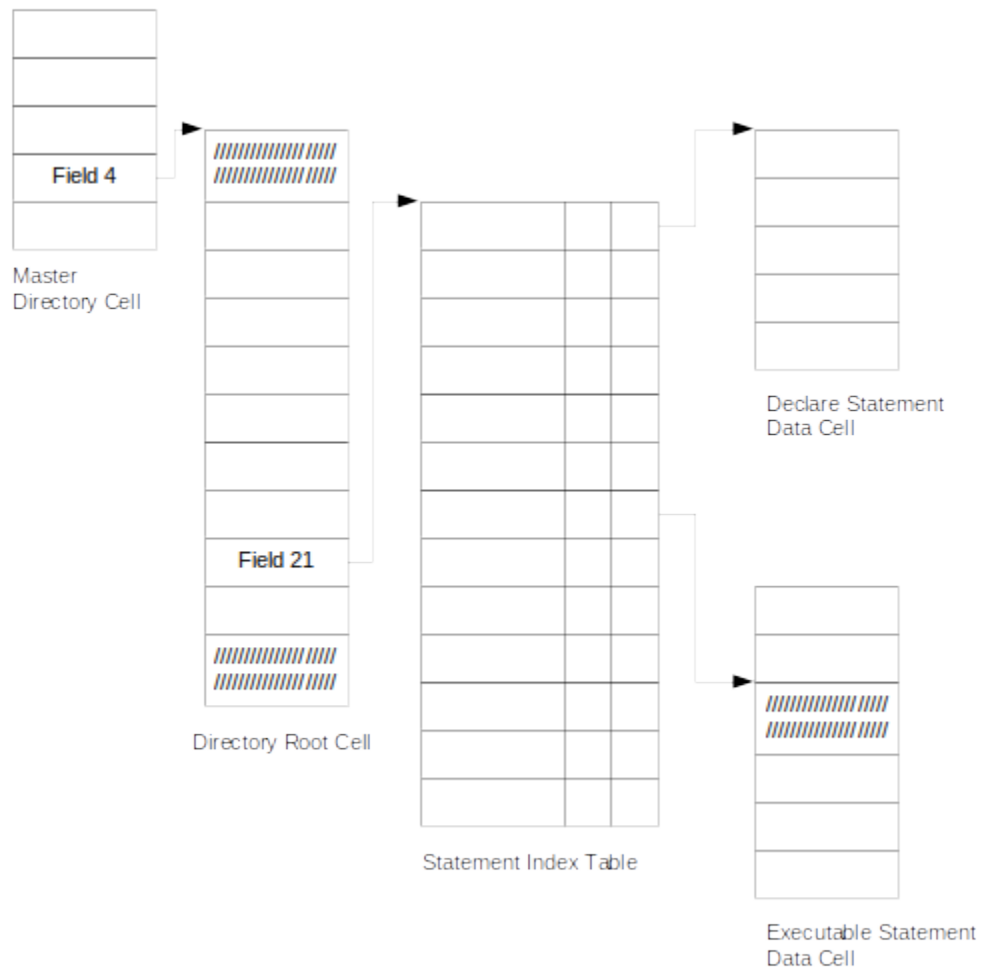


Figure 2-57 Statement Data Structures Overview

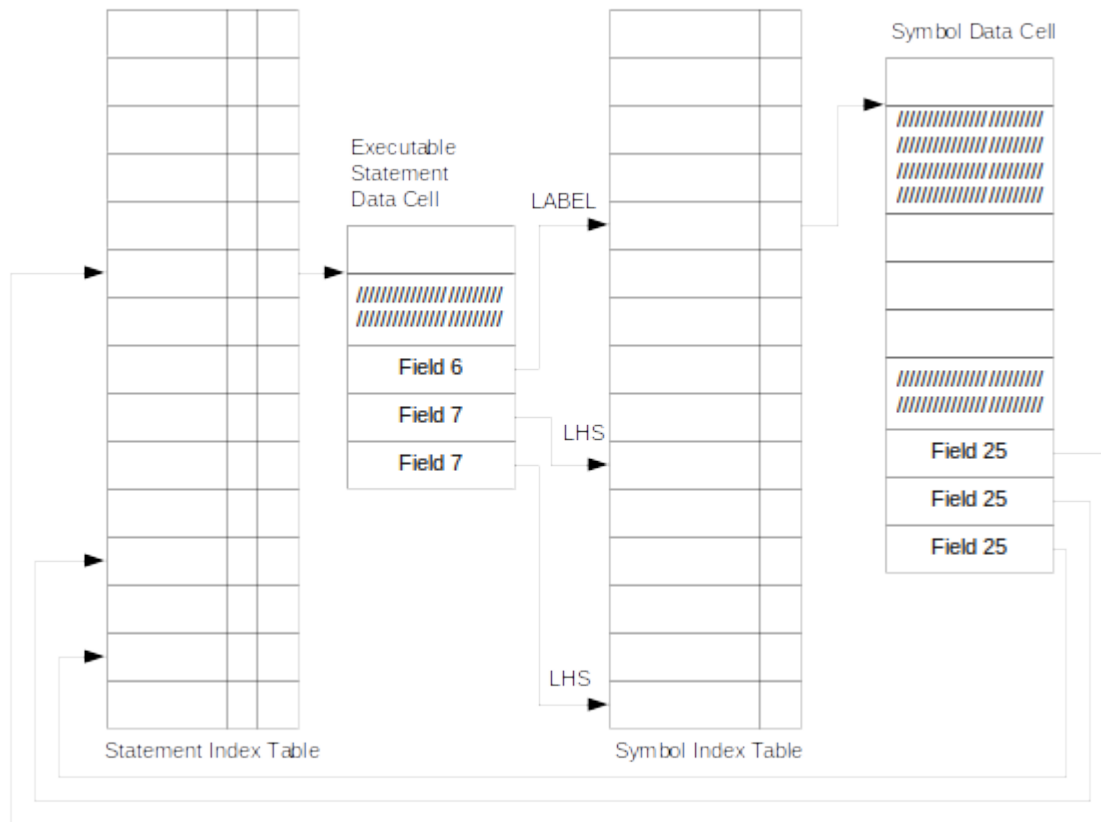


Figure 2-58 Statement/Symbol Relationship Overview

2.2.2.2.5.1 Statement Index Table

The Statement Index Table (Figure 2-59 on page 87) is pointed to by the Directory Root Cell and consists of 1680-byte physical records which are mapped and pointed to by the Block Statement Extent Cell. The Statement Index Table provides the means by which access can be made to the attributes of a statement. Entry to an element of the table can be through a binary search on statement reference numbers (SRNs) or by direct use of the internal statement number (ISN) generated by the compiler.

If the SRN_FLAG in the Directory Root Cell is on, each entry in the table is 12 bytes in length and consists of a six byte SRN field, a two byte INCLUDE count and a four byte pointer (page number and offset) field whose contents point to the location of the Statement Data Cell. If the SRN_FLAG is OFF, only the Statement Data Cell Pointer exists in the table; the SRN and Include Count Fields do not exist. An entry exists for each statement beyond the INCLUDEs of external modules. The entries are ordered in accordance with the internal statement numbers generated by the compiler. The SRNs, which are supplied by the user, are in ascending order in the table unless the NON_MONOTONIC_SRN_FLAG is ON in the Directory Root cell.

However, INCLUDED statements which appear immediately after an INCLUDE statement are represented in the INCLUDE count field by a 16-bit positive integer which is X'0001' for the first included statement, X'0002' for the second, etc. DECLARE

statements are treated in the same manner as executable statements. The pointer field in the table for a DECLARE statement is a negative pointer (-PTR) and points to a DECLARE Statement Data Cell. All other pointer fields of non-executable statements (other than DECLAREs) are zero.

6 Bytes	2 Bytes	4 Bytes
Statement Reference Number (SRN) +	INCLUDE count* +	▲ Pointer to Statement Data
⋮		⋮

* 16-bit positive integer
 + These fields appear only when SRN_FLAG is on (see text).

Figure 2-59 Statement Index Table

It should be noted that:

1. SRNs are not necessarily unique within a Compilation Unit (see Figure 2-60 on page 88 for an example). In this case the NON_UNIQUE_SRN_FLAG will be set to “on” in the Directory Root Cell.
2. The segment of the Statement Index Table containing the ISNs/SRNs for a particular block may contain embedded statements belonging to nested blocks (see Figure 2-61 on page 88 for an example).

M A = 1; B = 2; 000306
 ISN = 204 ISN = 205

Figure 2-60 Example of Non-unique SRNs

ISN		BLOCK NAME
204	ALPHA: PROCEDURE;	ALPHA
205	DECLARE A SCALAR;	ALPHA
206	A = 1;	ALPHA
207	BETA: PROCEDURE;	BETA
208	DECLARE B SCALAR;	BETA
209	B = 1;	BETA
210	CALL GAMMA (B);	BETA
211	RETURN;	BETA
212	CLOSE BETA;	BETA
213	CVAR = A;	ALPHA
220	CLOSE ALPHA;	ALPHA

Figure 2-61 Block Statement Nesting

2.2.2.2.5.2 Statement Data Cells

The Statement Data Cells consist of two types: Executable Statement Data Cell and the Declare Statement Data Cell. These Cells, which are referenced by SDF pointers in the Statement Index Table, provide information about the statements in a Compilation Unit.

Note: Declare Statement Data Cells do not contain address information; this information is readily available in the Symbol Data Cell (see Figure 2-32 on page 50).

2.2.2.2.5.2.1 Executable Statement Data Cell

The Statement Data Cell for an executable statement is indicated by a positive SDF pointer in the Statement Index Table. In addition, the Statement Type Field of both the Executable and the Declare Statement Data Cells may be used to determine whether the Cell is a Declare or Executable Cell since they occupy the same location in both cells. This Cell provides information about the Statement Type (see Figure 2-63 on page 93), offsets of the first and last machine instructions generated for the statement, and the indexes into the Symbol Index Table for any labels or assigned variables appearing in the statement.

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset Decimal (Hex)</u>	<u>Field Number</u>		<u>Bytes</u>	
-4	-8	-16 (-10)	0a	↑ LHS Statement Variables	4	Statement Data Cell Prefix
-3	-6	-12 (-C)	0b	↑ RHS Statement Variables	4	
-2	-4	-8 (-8)	0c	Flag Field	2	
-	-3	-6 (-6)	0d	Unused	2	
-1	-2	-4 (-4)	0e	↑ HALMAT Cell Pointer	4	
0	0	0 (0)	1	→ HAL/S Block Index	2	
-	1	2 (2)	2	Statement Category	1	
-	-	3 (3)	3	Statement Type	1	
1	2	4 (4)	4	Number of Label Indexes	1	
-	-	5 (5)	5	Number of LHS Halfwords	1	
			6	LABEL * 1 *	2	
			6	LABEL * 2 *	2	
				...		
			6	LABEL * m *	2	
			7	LHS * 1 *	2	
			7	LHS * 2 *	2	
				...		
			7	LHS * n *	2	
			8	Memory Address #1 (Relative)	3	
			9	Memory Address #2 (Relative)	3	
			10	Original SRN	6	

Figure 2-62 Executable Statement Data Cell

The Cell is of variable length and is of the following format:

Field No. Description

- 0a A pointer to an Expression Variables Cell (see Section 2.2.2.2.6, “Expression Variables Cell” on page 102). For real-time statements (UPDATE PRIORITY, SCHEDULE, CANCEL, and TERMINATE), the cell lists all processes whose status is changed. For other statement types, the cell describes the statement variables occurring in a left-hand-side context (i.e., whose values may be changed by the statement). The pointer is present only when the LHS information (Field 6) is incomplete, that is when some of the LHS
- 0a

<u>Field No.</u>	<u>Description</u>								
(Cont'd)	<p>variables are subscripted. When present, the LHS Expression Variables Cell contains complete information about the LHS variables, thus duplicating the field 6 data.</p> <p>The presence of Field 0a is indicated by setting the LHS bit (Field 2B).</p>								
0b	<p>A pointer to an Expression Variables Cell (see Section 2.2.2.2.6, "Expression Variables Cell" on page 102) describing the statement variables, including control variables, and any procedures or functions that occur in a right-hand-side context.</p> <p>The presence of Field 0b is indicated by setting the RHS bit (Field 2C).</p>								
0c	<table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left;"><u>Bit No.</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>0-5</td> <td>Unused</td> </tr> <tr> <td>6</td> <td> <p>1 – Statement contains an occurrence of a multi-instruction bit masking operation</p> <p>0 – Statement contains no bit masking operation</p> </td> </tr> <tr> <td>7-15</td> <td>Unused</td> </tr> </tbody> </table>	<u>Bit No.</u>	<u>Description</u>	0-5	Unused	6	<p>1 – Statement contains an occurrence of a multi-instruction bit masking operation</p> <p>0 – Statement contains no bit masking operation</p>	7-15	Unused
<u>Bit No.</u>	<u>Description</u>								
0-5	Unused								
6	<p>1 – Statement contains an occurrence of a multi-instruction bit masking operation</p> <p>0 – Statement contains no bit masking operation</p>								
7-15	Unused								
0d	Unused								
0e	<p>Pointer to a HALMAT Cell (see Section 2.2.2.2.9.1, "HALMAT Cells" on page 118). If there is no HALMAT for the statement (e.g., uninitialized DECLARE statement) the pointer is –1. If HALMAT is not included in the SDF (HALMAT_FLAG in the Directory Root Cell is OFF), this field is zero.</p>								
1	<p>Index to the HAL/S Block Index Table. This index into the Block Index Table makes it possible to obtain the CSECT name of the block within which the statement lies.</p>								

Field No. Description

2 The Statement Category Field indicates the presence of Fields 10, 0a, and 0b, as well as the Statement sub-type and Statement Context. This field, in conjunction with Field 3 below, is also used to resolve certain ambiguous statement types as defined by Field 3 (see Figure 2-63 on page 93). The following table shows the correspondence between the field and the data contained within it:

Number of Bits:	1	1	1	2	3
Sub-Fields:	A	B	C	D	E
	↑	↑	↑	↑	↑
	Original SRN	LHS	RHS	Sub- type	Statement Context Information

<u>Sub-Field</u>	<u>Description</u>
A	This indicates the presence of the Original SRN (Field 10).
B	This indicates the presence of the pointer to the LHS Expression Variables Cell (Field 0a).
C	This indicates the presence of the pointer to the RHS Expression Variables Cell (Field 0b).
D	This is used to resolve Statement Types 1, 2, 3, 4 and 10 into distinct HAL/S constructs.
E	This is used to specify the following Statement Contexts:
	<u>Statement Context Information</u>
	0 Null
	1 ELSE Statement
	2 THEN Statement
	4 ON ERROR Statement Reference

3 The Statement Type Field is used to indicate the type of statement this cell represents. This field is in the same position as the Statement Type Field in the Declare Statement Data Cell; therefore, this field can be used to determine the format of the Statement Data Cell (i.e., Declare or

Field No. Description

Executable).

Types:

Decimal Hex

0	0	Null
1	1	EXIT, REPEAT, GO TO
2	2	CALL
3	3	READ, READALL, WRITE
4	4	ASSIGNMENT
5	5	IF Condition
6	6	CLOSE
7	7	RETURN
8	8	END
9	9	SCHEDULE
10	A	CANCEL, TERMINATE
11	B	WAIT
12	C	UPDATE PRIORITY
13	D	SET, SIGNAL, RESET
14	E	SEND ERROR
15	F	ON ERROR
16	10	FILE
17	11	DO
18	12	DO WHILE, DO UNTIL
19	13	DO FOR
20	14	DO CASE

Decimal Hex

3 (Cont'd)	21	15	DECLARE (Used by Declare Statement Data Cell)
	22	16	BLOCK HEADER
	23	17	EQUATE (Used by Declare Statement Data Cell)
	24	18	TEMPORARY (Used by Declare Statement Data Cell)
	25-30	19-1E	Not Used

<u>Field No.</u>	<u>Description</u>
31	1F %NAMEBIAS
32	20 %SVC
33	21 %NAMECOPY
34	22 %COPY
35	23 %SVC I
36	24 %NAMEADD

Statement Type (Hex):				
	1	3	4	10
Statement Subtype:				
0	EXIT	READ	Assignment	FILE Input
1	REPEAT	READALL	NAME Assignment	FILE Output
2	GO TO	WRITE		

Figure 2-63 Statement Type

<u>Field No.</u>	<u>Description</u>
4	Number of label indexes. One such index is provided for each label attached to the statement. The index identifies the Symbol Index Table entry corresponding to the label.
5	Number of left-hand-side (LHS) halfwords.
6	Label indexes to corresponding symbol data.
7	LHS indexes to corresponding symbol data and/or sets of indexes which are keyed by a leading two byte negative value that identifies the number of following structure qualifiers and symbol indexes in a set. An index exists for each HAL/S variable that is "modified" by the statement. Thus, the variable can either be on the left-hand side of an assignment statement or can be the assigned variable in a READ or CALL statement. See Figure 2-64 on page 94 for an example of LHS Indexes.

```
DECLARE X SCALAR;  
STRUCTURE Q:  
  1 A,  
  2 B SCALAR;  
  2 C SCALAR;  
DECLARE Z Q-STRUCTURE;
```

Statement Example:

X, Z.A.B = 0;

5 LHS HALFWORDS

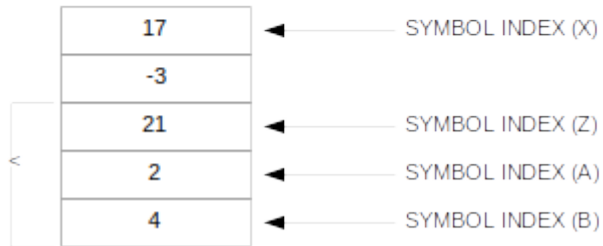


Figure 2-64 Left Hand Side (LHS) Indexes

<u>Field No.</u>	<u>Description</u>
8-9	The relative memory address of the first and last emitted lines of code for this statement (see description for Field 1). These two fields exist only if the ADDRS_FLAG in the Directory Root Cell is set.
10	This field is only present when the statement is part of INCLUDED HAL/S source text. The field contains 6 EBCDIC characters which are the original SRN of the statement in the INCLUDE file.

2.2.2.2.5.2.2 DECLARE Statement Data Cell

The Statement Data Cell for a DECLARE statement is indicated by a negative (two's complement) SDF pointer in the Statement Index Table. In addition, the Statement Type Field of both the Executable and the Declare Statement Data Cells may be used to determine whether the Cell is a Declare or Executable Cell since they occupy the same location in both cells. The Cell is of variable length and is of the following format:

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1		2
-	1	2	2		1
-	-	3	3		1
			4	↑ Expression Variables Cell Ptr	4
			5	↑ HALMAT Cell Ptr	4
			6	Original SRN	6

Figure 2-65 DECLARE Statement Data Cell

<u>Field No.</u>	<u>Description</u>
1	Index to the HAL/S Block Index Table of the HAL/S Block in which the declare statement appeared.
2	The Flag Field indicates the presence of Fields 4, 5, and 6. The following table shows the correspondence between the flags and the fields.

Number of Bits: 1 1 1 5

Sub-Fields:

A	B	C	Unused
---	---	---	--------

Field No. Description

2 (Cont'd)	<u>Sub-fields</u>	<u>Description</u>
	A	Indicates the presence of Field 4 (Expression Variables Cell Pointer)
	B	Indicates the presence of Field 5 (HALMAT Cell Pointer)
	C	Indicates the presence of Field 6 (Original SRN)

3 The Statement Type Field is used to indicate the type of statement this cell represents. This field is in the same position as the Statement Type Field in the Executable Statement Data Cell; therefore, this field can be used to determine the format of the Statement Data Cell (i.e., Declare or Executable).

Types:

<u>Decimal</u>	<u>Hex</u>	
21	15	Declare Statement
23	17	Statement contains Equate
24	18	Temporary variable Declaration
25	19	Replace Statement
26	1A	Structure definition

4 Points to an Expression Variables Cell listing NAME variables which are initialized in this statement.

5 Points to a HALMAT Cell if any HALMAT was generated for the statement and if HALMAT is included in the SDF.

6 The original SRN. Present only if the statement was INCLUDED.

2.2.2.2.5.3 Statement Extent Cell

The Statement Extent Cell (Figure 2-68 on page 99) allows the rapid location of the physical records containing statement information from a data set. The Statement Extent Cell contains six characters for the first and last SRNs followed by the two byte include counts contained in each of the physical records of the Statement Index Table. As the statement numbers increase in value from one reference and physical record to the next, a table look-up can be performed to determine the appropriate Statement Index Table Record. This table is present for all compilation units (see Figure 2-67 on page 98).

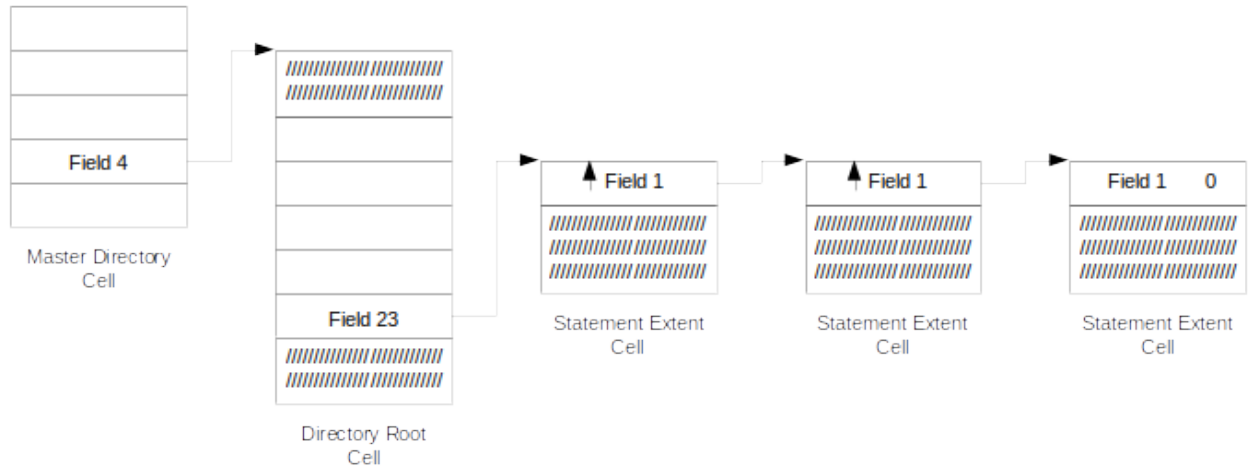


Figure 2-66 Statement Extent Cell Overview

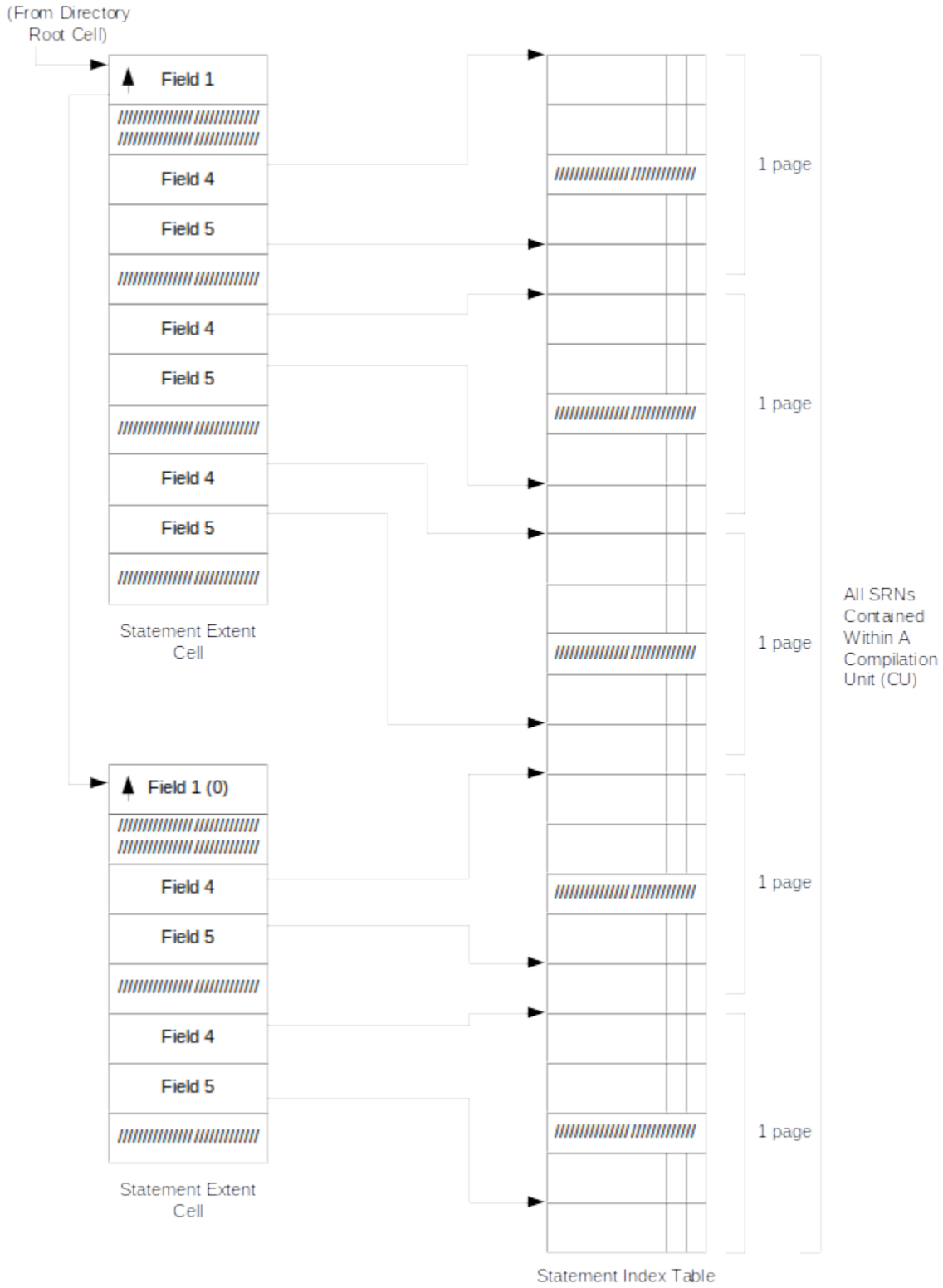


Figure 2-67 Relationship of Statement Extent Cells and Statement Index Table

Fullword Offset	Halfword Offset	Byte Offset Decimal (Hex)	Field Number			
0	0	0 (0)	1	↑	Successor	4
1	2	4 (4)	2		Number of Extent Entries	2
-	3	6 (6)	3		Page Number of 1st Physical Record of Statement Index Table	2
2	4	8 (8)	4		First Offset	2
-	5	10 (A)	5		Last Offset	2
3	6	12 (C)	6a		First SRN on Block	6
-	9	18 (12)	6b		First SRN Include Count	2
5	10	20 (14)	7a		Last SRN on Block	6
-	13	26 (1A)	7b		Last SRN Include Count	2
7	14	28 (1C)	4		First Offset	2
-	15	30 (1E)	5		Last Offset	2
8	16	32 (20)	6a		First SRN on Block	6
-	19	38 (26)	6b		First SRN Include Count	2
10	20	40 (28)	7a		Last SRN on Block	6
-	23	46 (2E)	7b		Last SRN Include Count	2
12	24	48 (30)	4		First Offset	2
-	25	50 (32)	5		Last Offset	2
13	26	52 (34)	6a		First SRN on Block	6
-	29	58 (3A)	6b		First SRN Include Count	2
15	30	60 (3C)	7a		Last SRN on Block	6
-	33	66 (42)	7b		Last SRN Include Count	2
17	34	68 (44)	4		First Offset	2
-	35	70 (46)	5		Last Offset	2
18	36	72 (48)	6a		First SRN on Block	6
-	39	78 (4E)	6b		First SRN Include Count	2
20	40	80 (50)	7a		Last SRN on Block	6
-	43	86 (56)	7b		Last SRN Include Count	2
					⋮	

1st Physical Record Corresponding to Field 3

2nd Physical Record

3rd Physical Record

4th Physical Record

Figure 2-68 Statement Extent Cell

2.2.2.2.5.4 Procedure/Function Invocation Cell

As shown in Figure 2-69 on page 100, the Procedure/Function Invocation Cell is referenced by a pointer in an Expression Variables Cell (see Section 2.2.2.2.6, “Expression Variables Cell” on page 102) or in another Procedure/Function Invocation Cell. The cell associates each formal parameter with a list of the variables, procedures, and functions involved in the expression which corresponds to the actual parameter of this particular invocation.

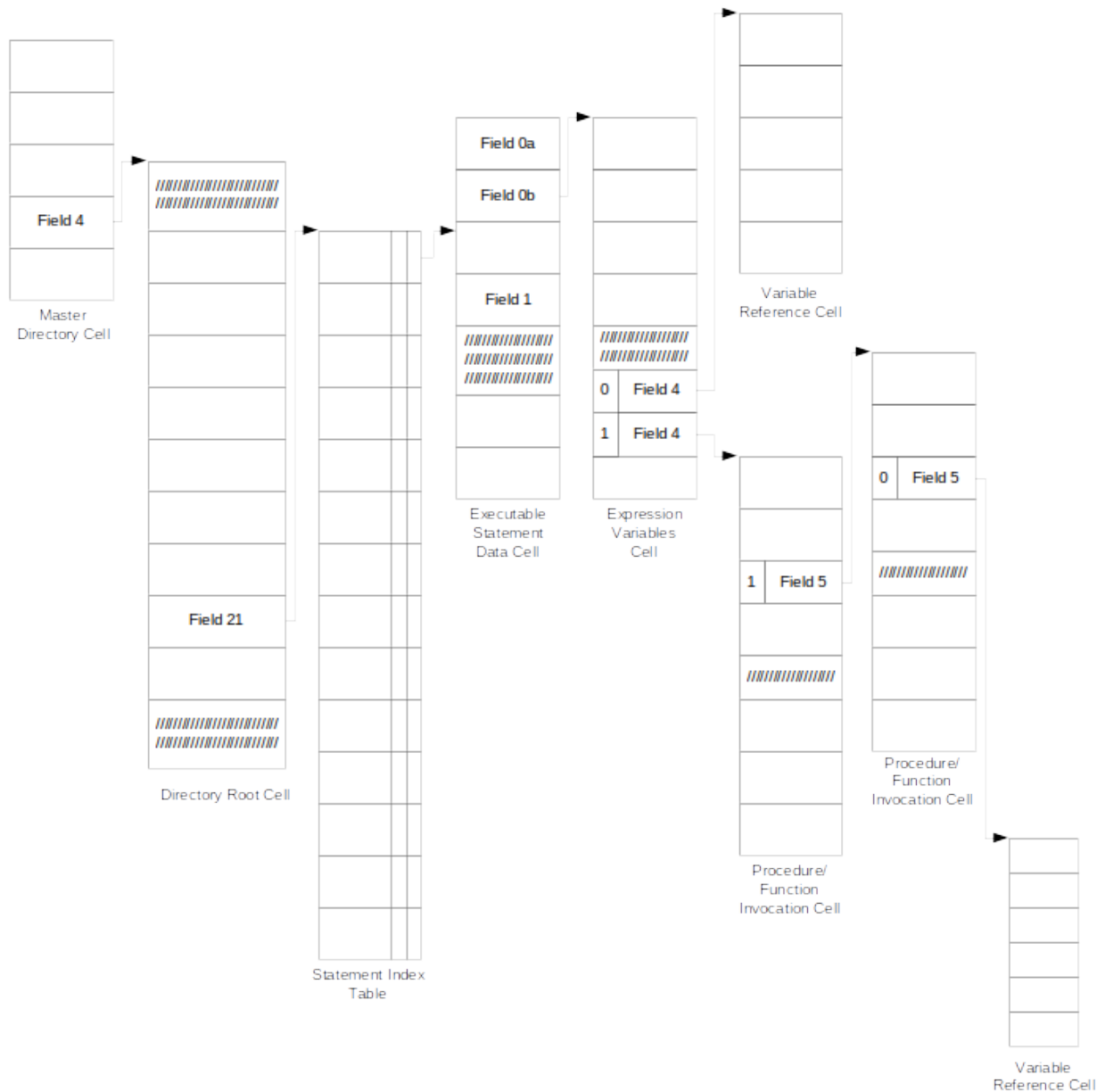


Figure 2-69 Procedure/Function Invocation Cell Overview

Fullword Offset	Halfword Offset	Byte Offset	Field Number		Bytes				
0	0	0	1	# Bytes in Cell	2				
-	1	2	2	# of Parameters	2				
1	2	4	3	# of Input Parameters	2				
-	3	6	4	Block Name Symbol # *	2				
2	4	8	5	<table border="1"> <tr> <td>T</td> <td rowspan="3">Actual Parameter Ptr</td> </tr> <tr> <td>A</td> </tr> <tr> <td>G</td> </tr> </table>	T	Actual Parameter Ptr	A	G	4
T	Actual Parameter Ptr								
A									
G									
			5	<table border="1"> <tr> <td>T</td> <td rowspan="3">Actual Parameter Ptr</td> </tr> <tr> <td>A</td> </tr> <tr> <td>G</td> </tr> </table>	T	Actual Parameter Ptr	A	G	4
T	Actual Parameter Ptr								
A									
G									
				.					
				.					
			5	<table border="1"> <tr> <td>T</td> <td rowspan="3">Actual Parameter Ptr</td> </tr> <tr> <td>A</td> </tr> <tr> <td>G</td> </tr> </table>	T	Actual Parameter Ptr	A	G	4
T	Actual Parameter Ptr								
A									
G									
			6	Formal Parameter Symbol #*	2				
			6	Formal Parameter Symbol #*	2				
				.					
				.					
			6	Formal Parameter Symbol #*	2				

Internal Blocks Only

* Index in Symbol Index Table

Figure 2-70 Procedure/Function Invocation Cell

The meanings of the fields of the Procedure/Function Invocation Cell are as follows:

Field No.	Description
1	The number of bytes in the cell.
2	The number of formal parameters. Also the number of pointers.
3	The number of parameters that are input parameters.
4	Index into the Symbol Index Table for the procedure or function name.
5	A fullword zero indicates the actual parameter is a literal value. Otherwise, the interpretation of the Actual Parameter Pointer is determined by the value of the 2-bit Tag field as follows:

<u>Field No.</u>	<u>Description</u>
	<u>Tag</u>
5 (Cont'd)	<p>0 Pointer to a Variable Reference Cell (see Section 2.2.2.2.7, "Variable Reference Cell" on page 105.) The actual parameter is a single variable that is in a qualified structure or is subscripted.</p> <p>1 A pointer to a Procedure/Function Invocation Cell. The actual parameter is the result of a function invocation.</p> <p>2 A pointer to an Expression Variables Cell (see Section 2.2.2.2.6, "Expression Variables Cell" on page 102). The actual parameter is a complex expression.</p> <p>3 The actual parameter is a single simple variable. The pointer value is an index into the Symbol Index Table.</p>
6	Indexes into the Symbol Index Table for each formal parameter. The parameters occur in the order of their appearance in the block header of the procedure or function. The nth formal parameter corresponds to the nth Actual Parameter Pointer. This field is only present for calls to internal blocks.

2.2.2.2.6 Expression Variables Cell

The Expression Variables Cell contains a list of references to unsubscripted variables followed by a list of pointers which refer to Procedure/Function Invocation Cells (see Section 2.2.2.2.5.4, "Procedure/Function Invocation Cell" on page 100) and to Variable Reference Cells (see Section 2.2.2.2.7, "Variable Reference Cell" on page 105) describing subscripted variables. Each pointer has a 2-bit tag indicating what type of cell is being referenced. As shown in Figure 2-71 on page 103, the Expression Variables Cell is used in a variety of contexts. When referenced from a Procedure/Function Invocation Cell, it describes the expression passed as an actual parameter to the procedure or function. When referenced from a Variable Reference Cell, it describes the expressions in the subscript list of a subscripted variable. A Statement Data Cell may refer to two Expression Variables Cells (pointers in fields 0a, 0b). Field 0a points to an Expression Variables Cell which describes all variables which occur in a left-hand-side context, that is, whose values might be changed by the statement. All other variables, including control variables, and any procedure or function calls, are said to occur in a right-hand-side context and are described in an Expression Variables Cell pointed to by field 0b. When referred to from a Declare Statement Data Cell, an Expression Variables Cell contains the symbol index of each NAME variable or structure with NAME terminals that is initialized in the statement.

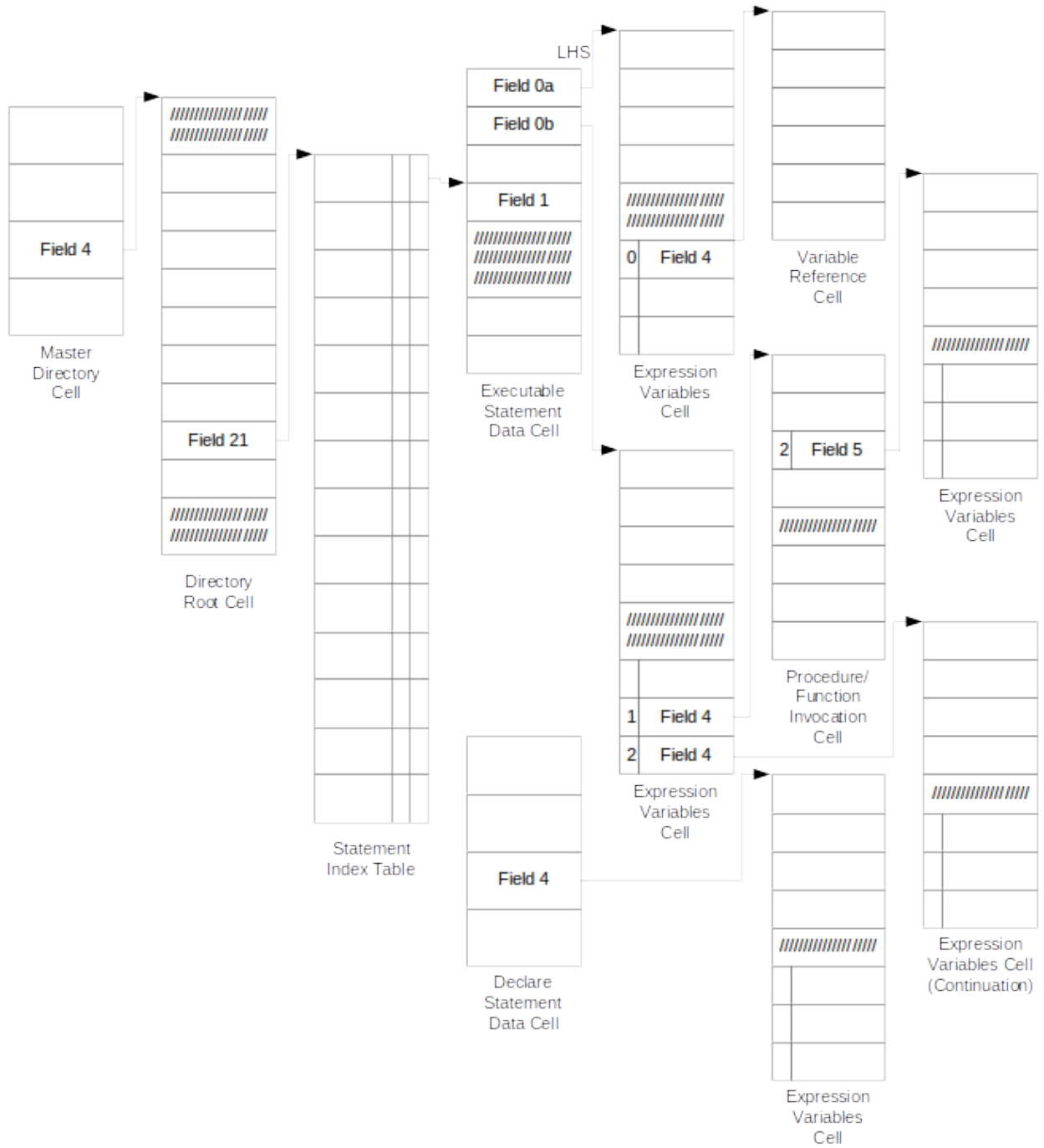


Figure 2-71 Expression Variables Cell Overview

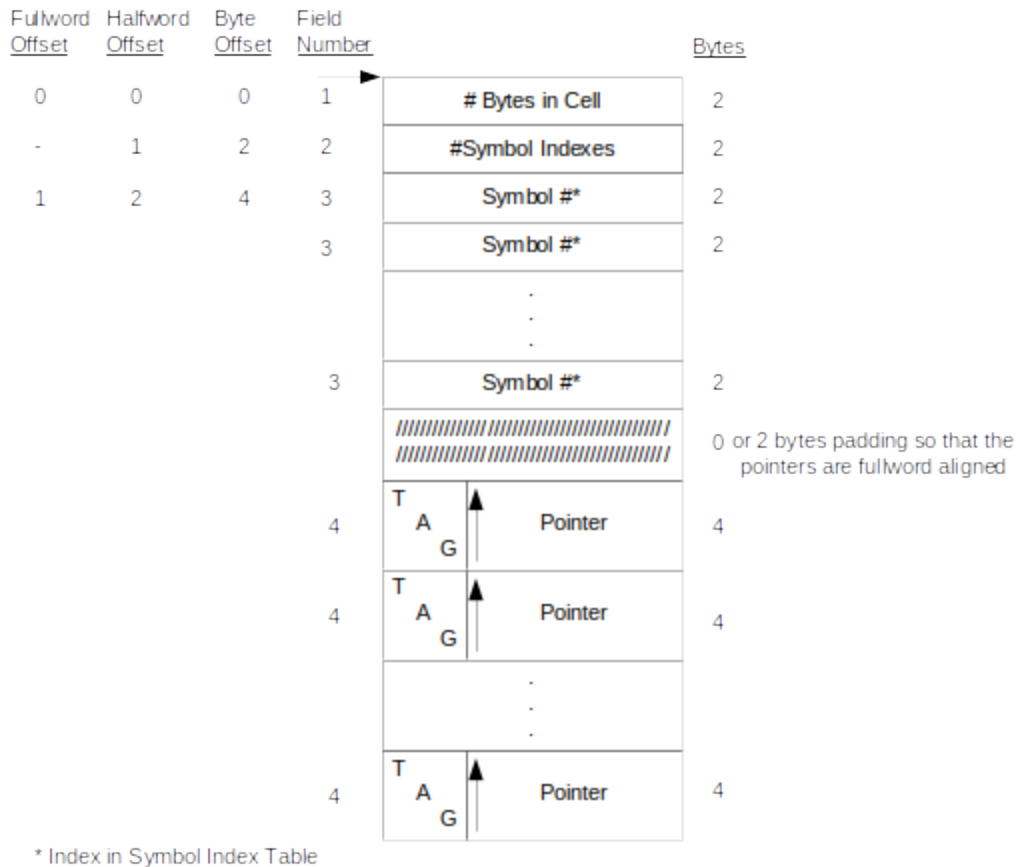


Figure 2-72 Expression Variables Cell

The meaning of the fields of the Expression Variables Cell are as follows:

Field No.	Description
1	The number of bytes in the cell.
2	The number of symbol index halfwords (field 3).
3	Indexes to corresponding symbol data and/or sets of indexes which are keyed by a leading two byte negative integer whose absolute value is the number of following structure qualifiers and symbol indexes in the set. See the explanation of the Variable Reference Cell, in field 4 of Section 2.2.2.2.7, "Variable Reference Cell" on page 105.

Field No. Description

- 4 The interpretation of the pointer is determined by the value of the 2-bit Tag Field as follows:

Tag

- 0 A pointer to a Variable Reference Cell describing a subscripted variable (see Section 2.2.2.2.7, "Variable Reference Cell" on page 105)
- 1 A pointer to a Procedure/Function Invocation Cell (see Section 2.2.2.2.5.4, "Procedure/Function Invocation Cell" on page 100).
- 2 In the highly unlikely event that an Expression Variables Cell did not fit on a single SDF page, it would be split into two cells and the last pointer of the first cell would have a Tag value of 2 and would point to the second cell.
- 3 A Tag value of 3 will only occur in the RHS Expression Variables Cell pointed to by field 0b of an Executable Statement Data Cell for a %NAMEADD or a %COPY statement. In this case, the pointer is not a pointer, it is instead the value of the third argument of the macro. When %COPY is called without specifying a halfword count, this field will be absent and the count is determined by the size of the source operand.

2.2.2.2.7 Variable Reference Cell

The Variable Reference Cell gives a complete description of a particular use of a variable. The cell occurs in a variety of contexts. It can be referenced by a pointer in an Expression Variables Cell (see Figure 2-73 on page 106) corresponding to an expression involving the variable. When it is referenced by a Name Terminal Initialization Cell (see Figure 2-74 on page 107), it describes the variable initially pointed to by one or more copies of the structure name terminal. For external Equate labels, the Auxiliary Symbol Information Pointer (Field 0c) of the Symbol Data Cell (see Figure 2-75 on page 108), for the label points to a Variable Reference Cell which describes the HAL/S variable which is equated to the external label. Finally, for non-structure NAME variables which are initialized, the Auxiliary Symbol Information Pointer of the Symbol Data Cell (see Figure 2-75 on page 108) for the NAME variable refers to a Variable Reference Cell describing the variable initially pointed to by the NAME variable.

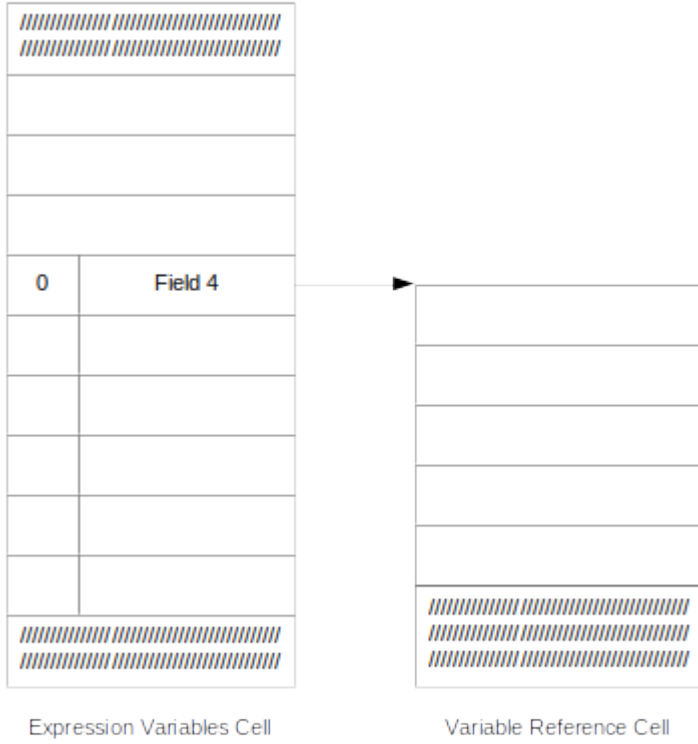


Figure 2-73 Variable Reference Cell Overview (Expression Variables Cell)

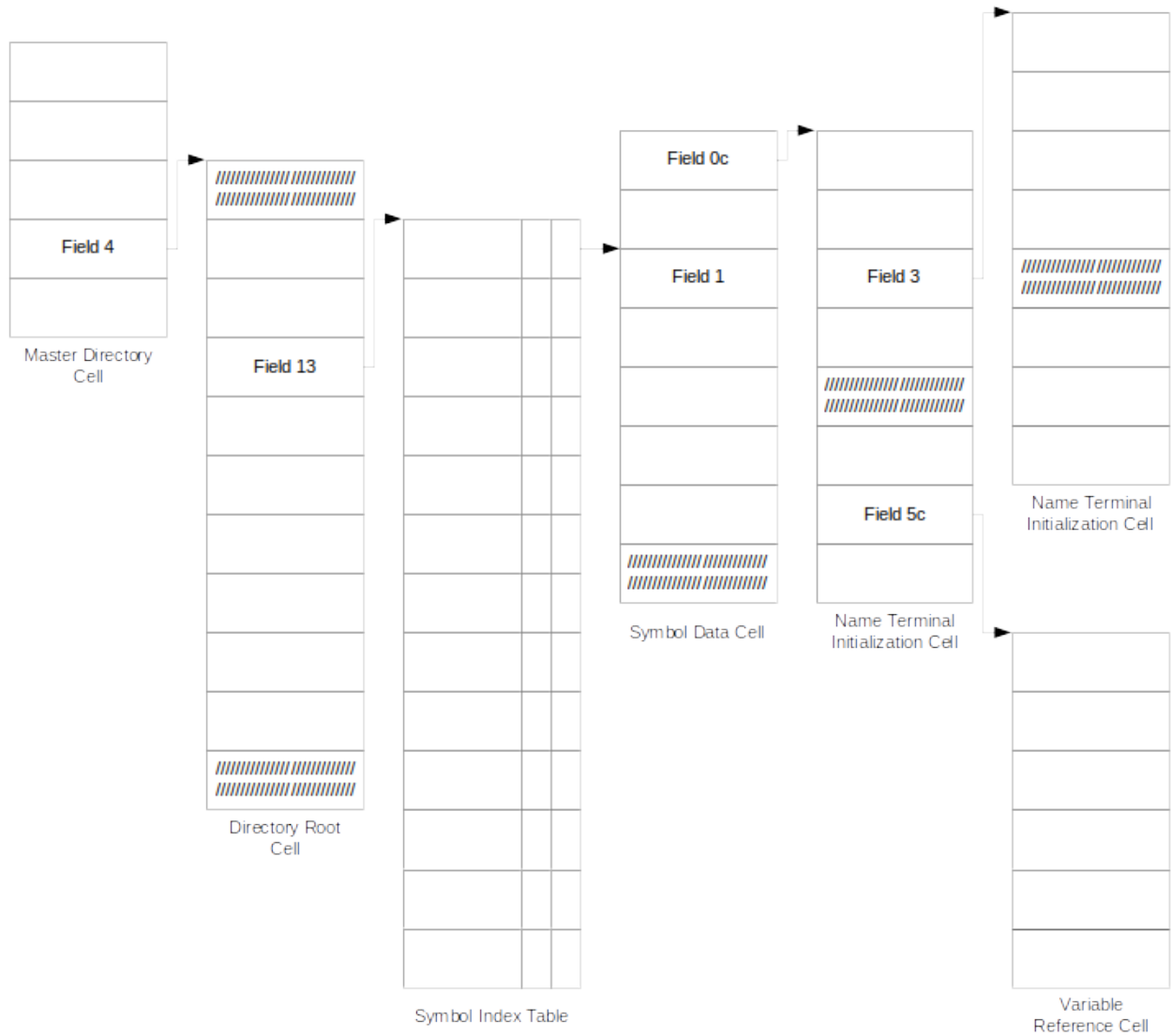
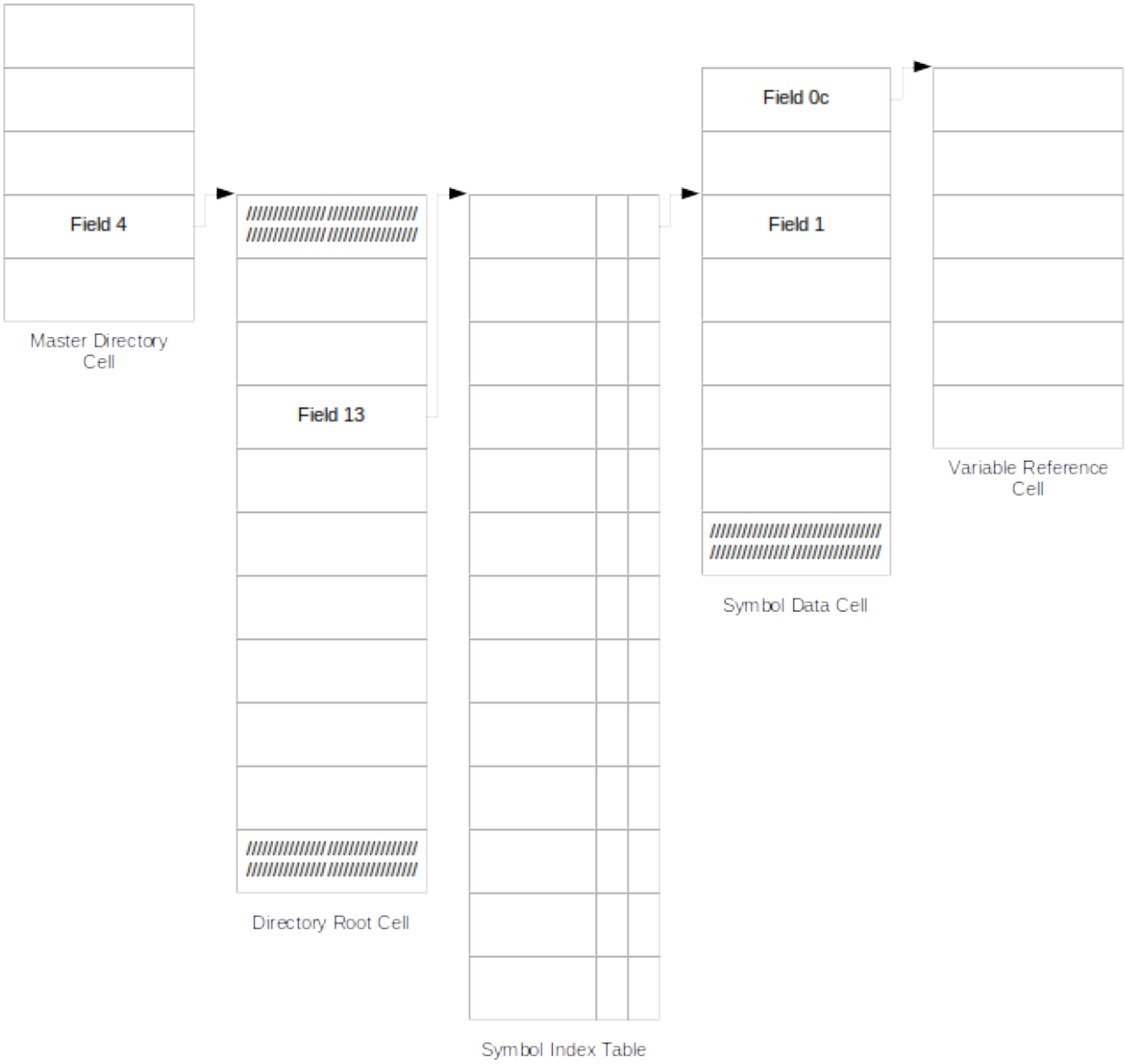


Figure 2-74 Variable Reference Cell Overview (Name Terminal Initialization Cell)



Symbol Data Cell points to Variable Reference Cell when:

1. HAL/S Symbol which points to a NONHAL Symbol is subscripted or is part of a structure
2. Non-Structure Name Variable points to subscripted or structure variable

Figure 2-75 Variable Reference Cell Overview (Symbol Data Cell)

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>		
0	0	0	1	# of Bytes in Cell	2		
-	1	2	2	<table border="1"> <tr> <td style="text-align: center;">F L A G</td> <td># of Symbol Indexes</td> </tr> </table>	F L A G	# of Symbol Indexes	2
F L A G	# of Symbol Indexes						
1	2	4	3	<table border="1"> <tr> <td style="text-align: center;">↑</td> <td>Ptr. to Expression Variables Cell</td> </tr> </table>	↑	Ptr. to Expression Variables Cell	4
↑	Ptr. to Expression Variables Cell						
2	4	8	4	Symbol #*	2		
			4	Symbol #*	2		
				. . .			
			4	Symbol #*	2		
			5	Subscript Descriptor	2		
			5	Subscript Descriptor	2		
				. . .			
			5	Subscript Descriptor	2		

* Index in Symbol Index Table

Figure 2-76 Variable Reference Cell

The meaning of the fields of the Variable Reference Cell are as follows (see Figure 2-76 on page 109):

Field No. Description

- 1 The number of bytes in the cell.
- 2 The number of symbol indexes (Field 4). The flag bit indicates the presence of subscript descriptors.
- 3 A pointer to an Expression Variables Cell (see Section 2.2.2.2.6, "Expression Variables Cell" on page 102) which lists all the variables, procedures, and functions involved in the subscript expressions. If all subscript values are known at compile time, this is indicated by a null pointer.
- 4 Indexes into the Symbol Index Table which contains the pointers to the Symbol Data Cells. In the case of a simple variable there would be a single Symbol Index. The case of a structure variable is best illustrated by an example.

Reference to a structure node X.B.C.E.F.Z:

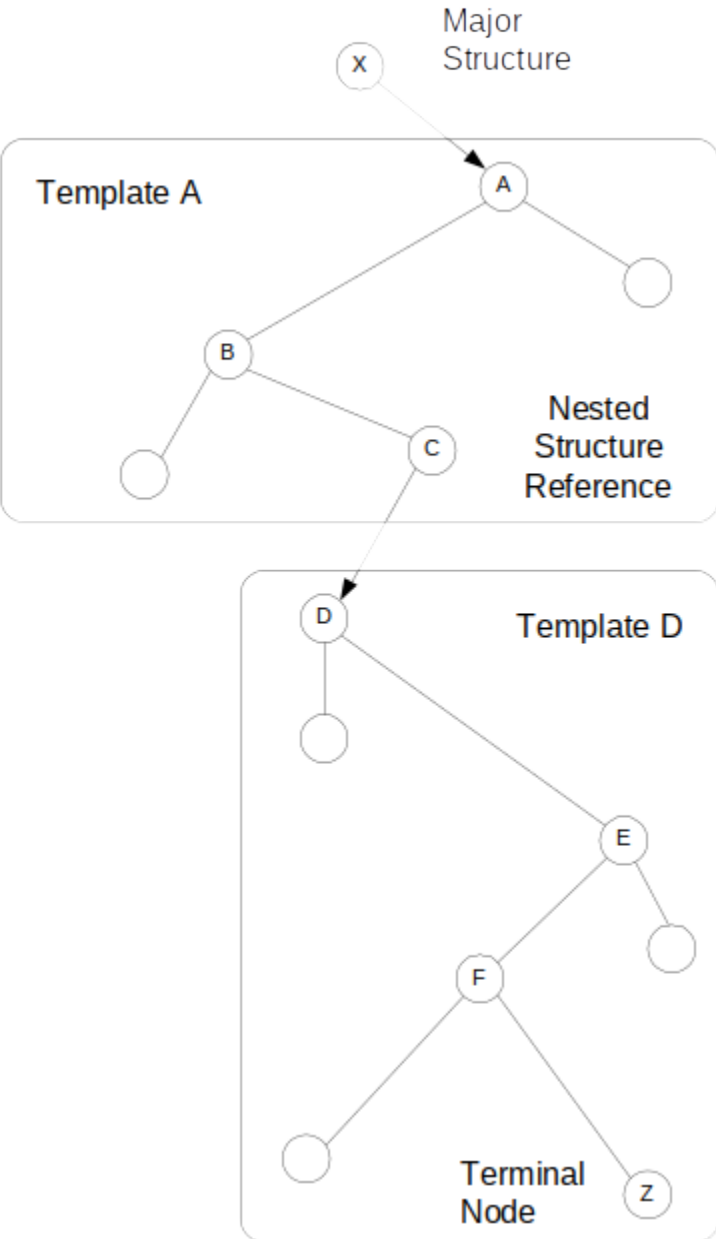


Figure 2-77 Structure Reference Diagram

This reference would generate three symbol indexes:

index for X
index for C
index for Z

Field No. Description

5 This field contains halfwords of subscript type information (Fields 5A, 5B, 5C, and 5D) and of the literal subscript value (Field 5E).

The first halfword is the subscript type information. It is discussed below:

Bits 0 5 6 7 8 11 12 15
 :

Element Type	Subscript Type (α)	Expression Type	Subscript Continuation Flag (β)
5A	5B	5C	5D

Sub-Field No. Description

5A The Element Type sub-field describes the type of item being subscripted. The valid types are:

- | | |
|------|---------------------------|
| Type | Description |
| 0 = | Component (Vector/Matrix) |
| 1 = | Array |
| 2 = | Structure |

5B The Subscript Type (α) describes the type of subscript operation being performed. The valid types are:

- | | |
|------|--------------------------|
| Type | Description |
| 0 = | * operation |
| 1 = | Index Value |
| 2 = | “TO” partition operation |
| 3 = | “AT” partition operation |

5C The Expression Type describes the data involved in the actual subscript. The different types are represented by combinations of the bits described

below:

<u>Sub-Field No.</u>	<u>Description</u>	Bits	8	9	10	11
5C (Cont'd)		:				
			Number Specified in Subscript	+ (plus) Subscript Expression	- (minus) Subscript Expression	Literal Value Specified

The valid expression type values are:

Type	Description
0 =	Variable Expression Specified
1 =	Literal Value Specified
2 =	Number + Variable Expression Specified
3 =	Number + Literal Specified
4 =	Number - Variable Expression Specified
5 =	Number - Literal Specified
6 =	invalid
7 =	invalid
8 =	Only Number specified in Subscript

When the Literal Value bit is set, the Subscript Type halfword is followed by an additional halfword which contains the Literal Value specified in the subscript (see Field 5E below).

5D The Subscript Continuation Flag (β) indicates whether the next two Subscript Descriptor halfwords are a continuation of the current set (i.e., the second part of a "TO"/"AT" subscript partition).

Type	Description
0 =	Subscript partition is not continued
1 =	Subscript partition is continued in the next two halfwords of the Subscript Descriptors.

5E Bits: 0 15

16 Bit Signed Literal Value

This field contains the signed 16-bit Literal subscript value specified in the HAL/S

program.

2.2.2.2.8 Function Tables

As shown in Figure 2-78 on page 114, the Function Index Table (see Figure 2-79 on page 115) is pointed to by Field 52 of the Directory Root Cell. The pointer field within the Function Index Table points to the Function XREF Data Cell (see Figure 2-80 on page 117). When necessary, Field 4 of the Function XREF Data Cell points to the Function XREF Extension Cell (see Figure 2-81 on page 117). The Function Tables are comprised of the Function Index Table and the Function XREF Data Cell. These cells provide a means of accessing cross-reference information for Built-In Functions and explicitly invoked HAL/S Shaping Functions.

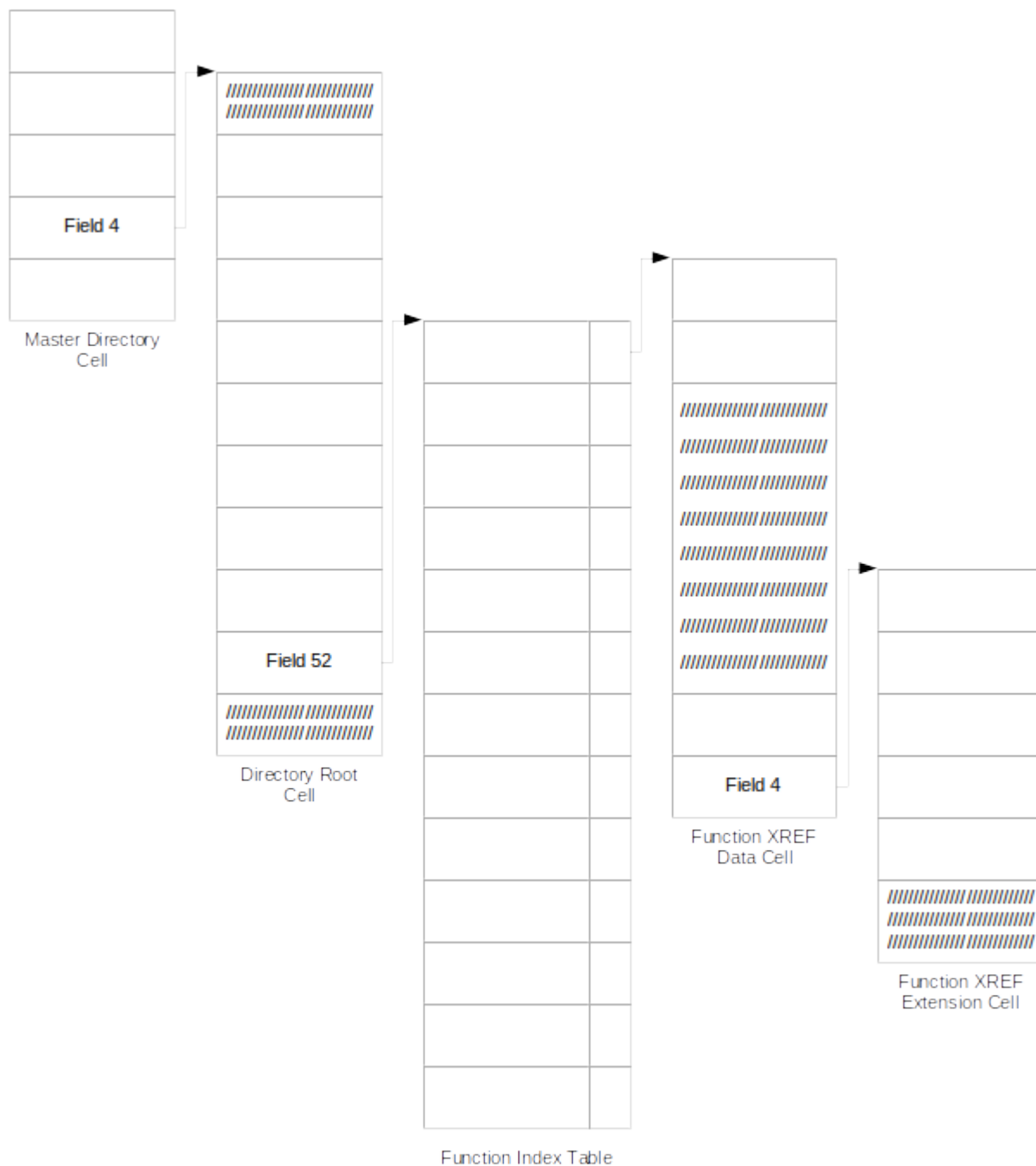


Figure 2-78 Function Data Overview

2.2.2.2.8.1 Function Index Tables

The Function Index Table (Figure 2-79 on page 115) consists of a four byte function-type field followed by an SDF pointer which points to the Function XREF Data Cell (see Figure 2-80 on page 117 and Figure 2-81 on page 2-117). This table contains an entry for every function used in the compilation unit.

4 Bytes Function Type	4 Bytes ↑ XREF DATA
.	.
.	.
.	.
.	.

Figure 2-79 Function Index Table

The Function types of the Built-in Functions are listed below:

Type	Function	Description
1	ABS	Absolute Value
2	COS	Cosine
3	DET	Determinant
4	DIV	Integer Division
5	EXP	e^x
6	LOG	Natural Log (Log base e)
7	MAX	Maximum Value in array
8	MIN	Minimum Value in array
9	MOD	Modulus
10	ODD	Odd Value (1, 3, 5, ...)
11	SHL	Bit Shift Left
12	SHR	Bit Shift Right
13	SIN	Sine
14	SUM	Sum of items in array
15	TAN	Tangent
16	XOR	Exclusive OR
17	COSH	Hyperbolic Cosine
18	DATE	Current Date
19	PRIO	Process Priority
20	PROD	Product of items in array
21	SIGN	Sign of Value (+1 for non-negative, -1 for negative)
22	SINH	Hyperbolic Sine
23	SIZE	Length of array or structure
24	SQRT	Square Root
25	TANH	Hyperbolic Tangent
26	TRIM	Remove Leading and Trailing Blanks in String
27	UNIT	Unit Vector with same direction
28	ABVAL	Length of a vector

29	FLOOR	Largest Integer $\leq X$
30	INDEX	Index in Character String
31	LJUST	Left Justify Character String
32	RJUST	Right Justify Character String
33	ROUND	Round to nearest Integer
34	TRACE	Sum of Matrix Diagonal
35	ARCCOS	Inverse Cosine
36	ARCSIN	Inverse Sine
37	ARCTAN	Inverse Tangent
38	ERRGRP	Group Number of Last Error
39	ERNUM	Number of Last Error
40	LENGTH	Length of Character String
41	MIDVAL	Middle Value
42	RANDOM	Random Number between zero and one
43	SIGNUM	Sign of Value (+1 for positive, 0 for zero, -1 for negative)
44	ARCCOSH	Inverse Hyperbolic Cosine
45	ARCSINH	Inverse Hyperbolic Sine
46	ARCTANH	Inverse Hyperbolic Tangent
47	ARCTAN2	Inverse Hyperbolic Tangent
48	CEILING	Smallest Integer $\geq X$
49	INVERSE	Matrix Inverse
50	NEXTIME	Task Scheduler
51	RANDOMG	Random Number between zero and one (Gaussian Distribution)
52	RUNTIME	Time since module began running
53	TRUNCATE	Truncate to Integer Value
54	CLOCKTIME	Elapsed Time Since Midnight
55	REMAINDER	Integer Division Remainder
56	TRANSPOSE	Transpose Matrix along Major Diagonal

For explicitly invoked Shaping Functions, the Function Types are as follows:

Type	Function
60	BIT
61	SUBBIT
62	INTEGER
63	SCALAR
64	VECTOR
65	MATRIX
66	CHARACTER

2.2.2.2.8.2 Function XREF Data Cell

Each Function XREF Data Cell contains the number of XREF entries and a list of the statements in which the function is invoked.

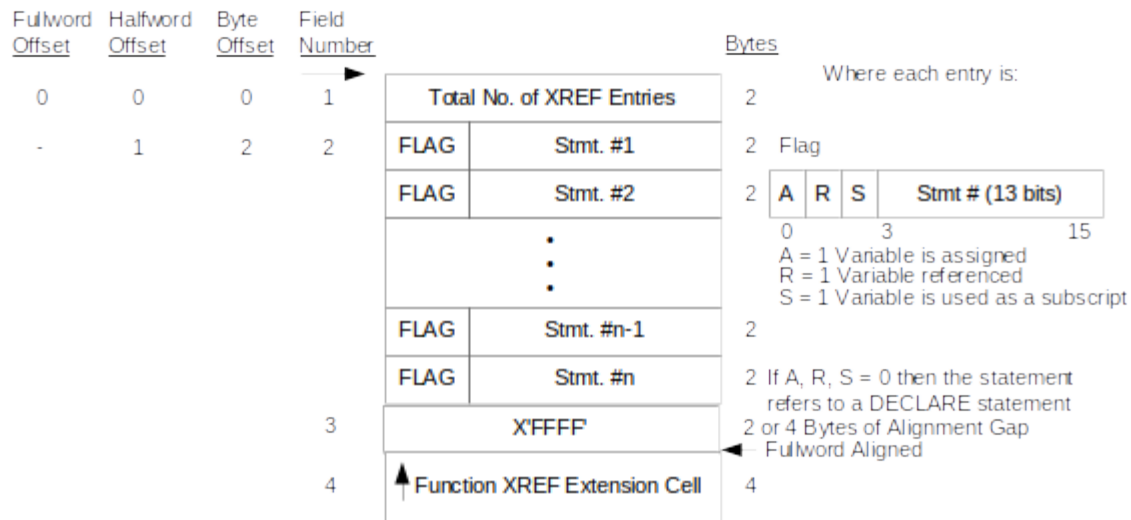


Figure 2-80 Function XREF Data Cell

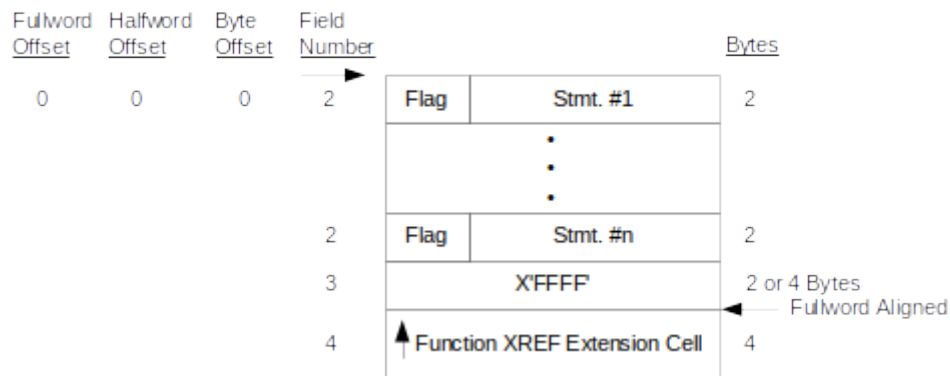


Figure 2-81 Function XREF Extension Cell

The Function XREF Data Cell fields are described below:

Field No. Description

- 1 This field contains the total number of cross-reference entries stored in the cell.
- 2 This field contains the statement cross-reference information in the lower 13 bits of the two-byte field. This information is in the form of indexes into the

Statement Index Table. The upper three flag bits denote the usage of the function and are the same as those contained in the Symbol Data Cell. For convenience, these flags are also provided in Figure 2-81 on page 117.

- 3 If all of the statement references cannot be contained in a single page, the data is extended to another page by means of an SDF pointer. When this is the case, this field will be either two or four bytes long (so that Field 4 can start on a full word boundary) and be filled with hex'FF'.
- 4 This field contains an SDF pointer which points to the Function XREF Extension Cell (see Figure 2-81Figure 2-80on page 117) which contains the remainder of the function cross-reference information.

2.2.2.2.9 HALMAT Data Structures

The HALMAT Data Structures consist of two basic classes of cells and tables: the HALMAT Cells and the Literal Tables which are used by the HALMAT Cells. Both classes of data structures are discussed in more detail in the following sections.

NOTE: The HAL/S-FC compiler feature that results in the creation of HALMAT Data Structures is not used and there are no plans for using it. This feature should be considered “unverified “ and should not be used in a production environment. The description of HALMAT Data Structures contained in the subsequent sections may not accurately reflect what the HAL/S-FC compiler will produce if this feature were to be used.

2.2.2.2.9.1 HALMAT Cells

The HALMAT Cells consist of the HALMAT Cell (see Figure 2-83 on page 120) and the HALMAT Extension Cell (see Figure 2-84 on page 120). These cells contain a modified version of the HALMAT Intermediate Language Data produced by Phase 1 of the HAL/S Compiler for a single HAL/S statement. In the SDF version of the HALMAT Cells, the Phase 1 Compiler symbol numbers have been changed to Indexes into the Symbol Index Table, the Virtual Accumulator (VAC) pointers to SDF offsets into the HALMAT Cells, and the indexes into the Phase 1 Compiler Literal Table to indexes into the SDF Literal Table (see Section 2.2.2.2.9.2, “Literal Data” on page 121). It should be noted that the HALMAT SMRK (HALMAT Statement Marker) Operators are removed before the HALMAT is inserted into the SDF.

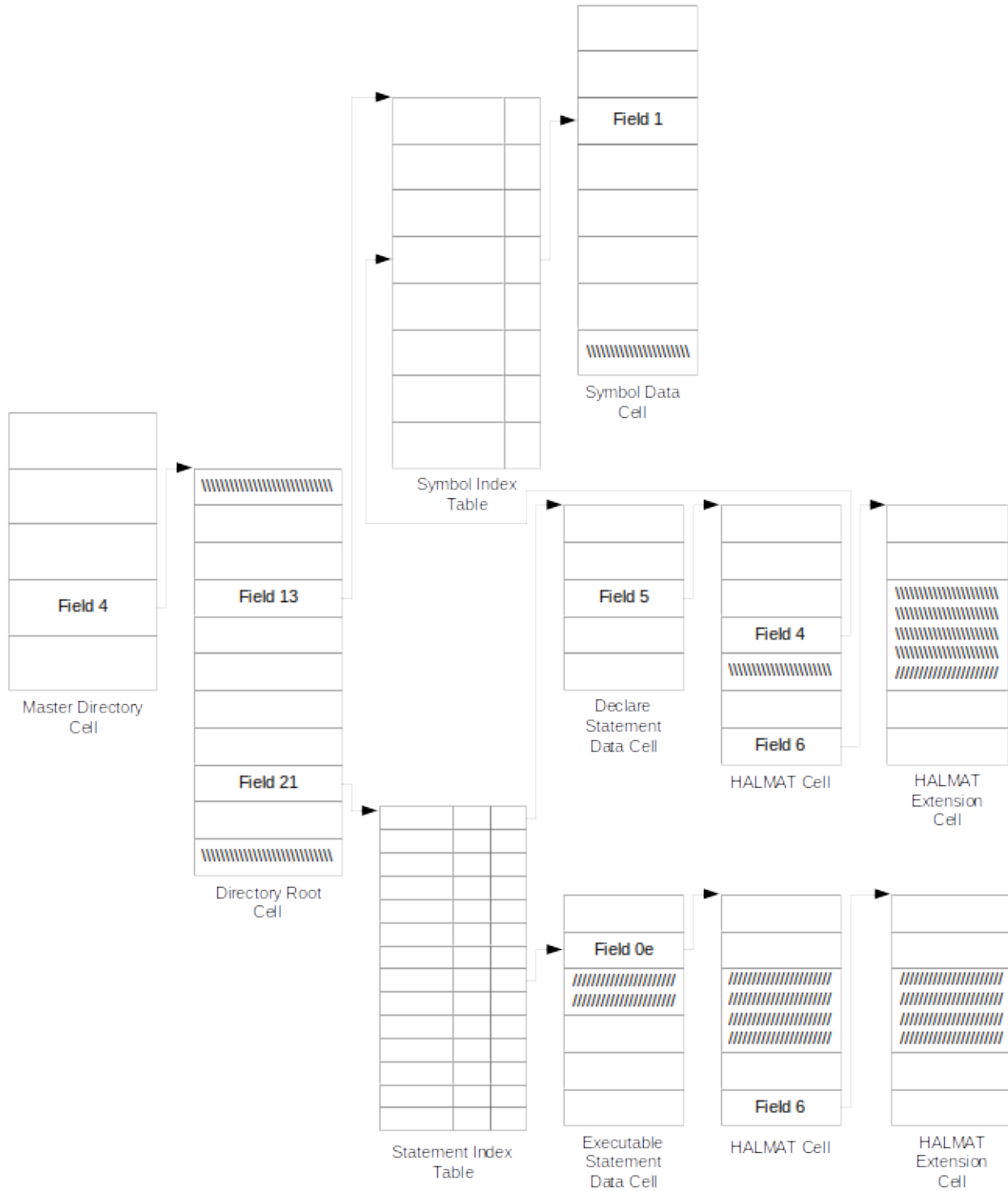


Figure 2-82 HALMAT Data Cells Overview

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	1	# of words of HALMAT for this statement	2
-	1	2	2	Word Offset to last operator	2
1	2	4	3	HALMAT Operator #1	4
			4	HALMAT Operand #1	4
			4	HALMAT Operand #2	4
				
			4	HALMAT Operand #n	4
			3	HALMAT Operator #2	4
				
			3	HALMAT Operator #n	4
			4	4
			5	FFF...FF	4
			6	↑ HALMAT Extension Cell	4

Present only if HALMAT is continued in extension cell

Figure 2-83 HALMAT Cell

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Bytes</u>
0	0	0	3	HALMAT Operator	4
			4	HALMAT Operand	4
				•	
				•	
				•	
				•	
				•	
				•	

Figure 2-84 HALMAT Extension Cell

The meaning of the fields in the HALMAT and HALMAT Extension Cells is as follows:

<u>Field No.</u>	<u>Description</u>
1	This field contains the number of words of HALMAT generated for the statement.
2	This field contains the fullword offset to the last HALMAT operator of the cell.
3	This field contains the HALMAT operator.
4	This field contains the HALMAT Operand.
5	This is a pad field of two or four bytes used to force fullword alignment. When this field contains all X'FF', it indicates the presence of the HALMAT Extension Cell Pointer (Field 6 below).
6	This field contains the Pointer to a HALMAT Extension Cell (see Figure 2-84 on page 120). It and Field 5 above are present only when all of the HALMAT information for a particular statement will not fit into the current SDF page.

2.2.2.2.9.2 Literal Data

The Phase 1 literal table is rearranged and included in the SDF. The SDF literal table differs from the compiler's table in that the three fields (i.e., LIT1, LIT2, and LIT3) are contiguous. As shown in Figure 2-85 on page 122, the Literal Extent Table (see Figure 2-86 on page 123) is pointed to by Field 50 of the Directory Root Cell; the only field of this table points to the Literal Table (see Figure 2-87 on page 124). In addition, as is indicated in the overview, Field 2B of the Character Literal Cell points to a string entry in the Literal Character Table; furthermore, Field 2 of the Template Subscript Literal Cell contains the Symbol Number of the Template Symbol.

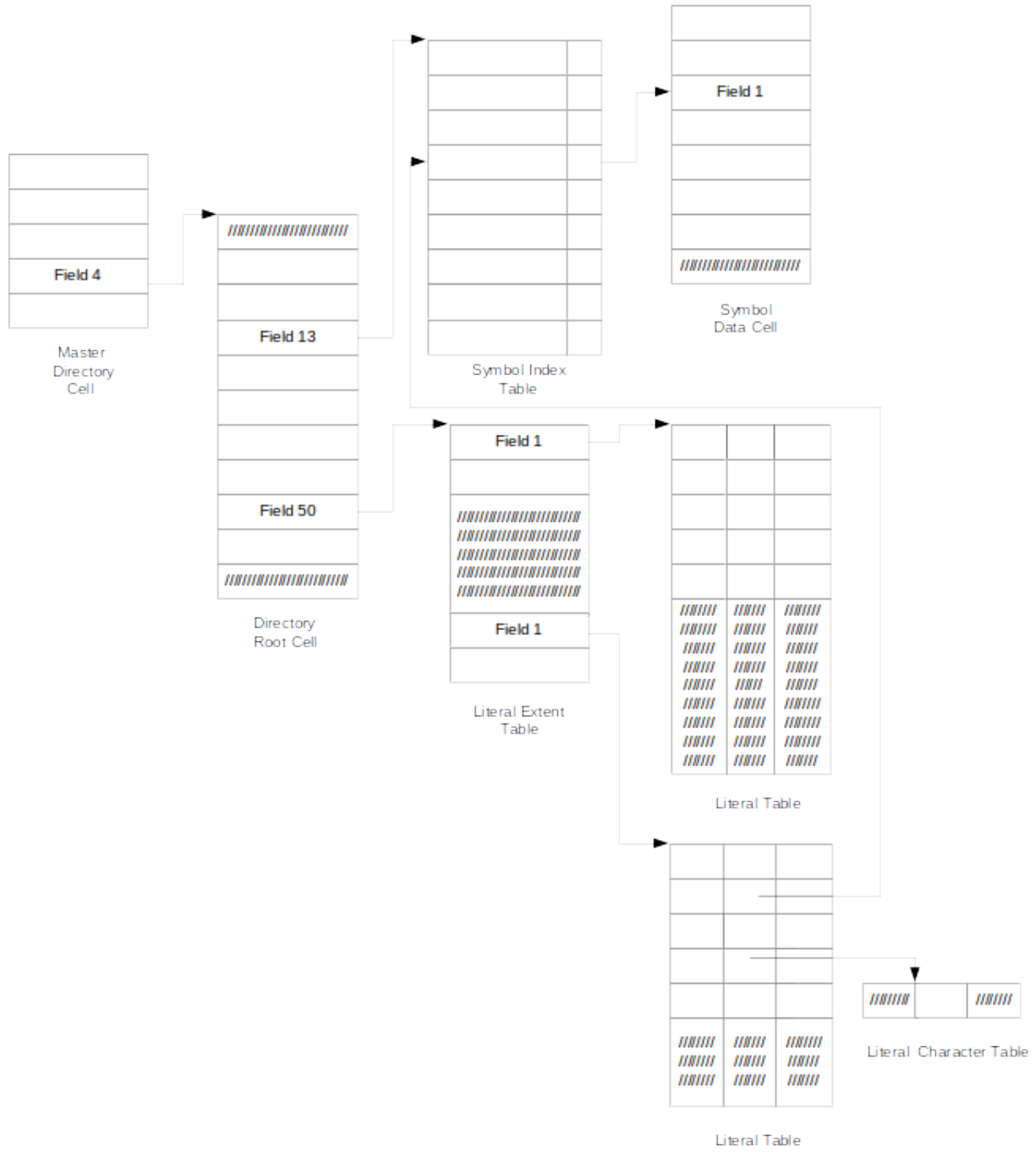


Figure 2-85 Literal Data Overview

2.2.2.2.9.2.1 Literal Extent Table

The Literal Extent Table contains pointers to the pages in the SDF member which are occupied by the Literal Table. The Literal Extent Table is pointed to by Field 50 of the Directory Root Cell. This table is present only when HALMAT is contained in the SDF member.

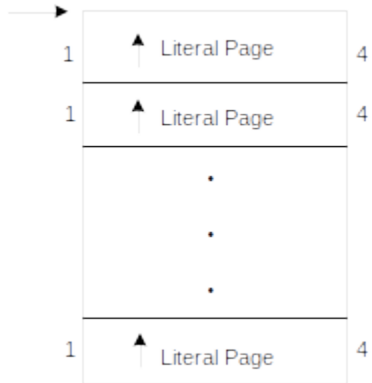


Figure 2-86 Literal Extent Table

The Literal Extent Table Field is described below:

<u>Field No.</u>	<u>Description</u>
------------------	--------------------

1	SDF Pointer to a part of the Literal Table within an SDF page.
---	--

2.2.2.2.9.2.2 Literal Tables

The Literal Table consists of four types of table entries (Character, Arithmetic, Bit, and Template) and a Literal Character Array. The Literal Table is essentially a reformatted version of the Literal Table produced by Phase 1 of the HAL/S Compiler. The SDF Literal Table differs from the compiler's table in that the three fields, LIT1, LIT2, and LIT3, are contiguous. The Literal Table and Literal Character Array are present only when HALMAT is contained in the SDF. The formats of arithmetic and bit literals are the same as those of the compiler's literal table.

	Field 1	Field 2	Field 3
1	LIT1	LIT2	LIT3
2	LIT1	LIT2	LIT3
3	LIT1	LIT2	LIT3
	.	.	.
	.	.	.
	.	.	.
N	LIT1	LIT2	LIT3

Figure 2-87 Literal Table

The fields of the Literal Table are described below:

Field No. Description

- 1 The LIT1 field contains the information as to the type of cell that is being represented. (Four byte field of which only the last byte is used.)

- 2 The LIT2 field generally contains information about the literal data item contained in the cell. In the case of a character literal, the LIT2 field contains both a length and a pointer into the array LIT_CHAR that has been copied into the SDF from Phase 1. (Four byte field.)

- 3 The LIT3 field may or may not be used in the different cells, but may either contain data (arithmetic) or length of the data (bit). (Four byte field.)

2.2.2.2.9.2.2.1 Character Literal

The Character Literal Cell contains the length of the literal character string minus one and an offset into the Literal Character Array containing the string. The Literal Character Array is declared as BIT(8).

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Field Size (in bytes)</u>
0	0	0	1A	Unused Pad Space	3
-	-	3	1B	Cell Type (0)	1
1	2	4	2A	Length -1 (1-256)	1
-	-	5	2B	↑ Pointer to LIT_CHAR Array	3
2	4	8	3	Unused Filler Area	4

Figure 2-88 Character Literal Cell

The format of the Character Literal Cell is the same as that of the HAL/S Compiler. The Character Literal Cell fields are described below:

<u>Field No.</u>	<u>Description</u>
1	LIT1 Field
1A	Unused pad area
1B	This field contains the information as to the type of cell that is being represented. (One byte field.)
2	LIT2 Field
2A	This is the length of the literal string minus one (i.e., XPL format where X'00'=1 character and X'FF'=256). (One byte field.)
2B	This field contains an offset into the array LIT_CHAR that has been copied into the SDF from Phase 1. (Three byte field.)
3	LIT3 Field
	The LIT3 field is not used in the Character Literal Cell. (Four byte field.)

2.2.2.2.9.2.2.2 Arithmetic Literal

The Arithmetic Literal Cell contains either Single or Double Precision Numeric Literals in either Fixed or Floating Point format. If needed, the LIT3 field is used as a continuation of the LIT2 field.

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Field Size (in bytes)</u>
0	0	0	1A	Unused Pad Space	3
-	-	3	1B	Cell Type (1)	1
1	2	4	2	Numeric Data (see Text)	4
2	4	8	3	Numeric Data or Unused (see Text)	4

Figure 2-89 Arithmetic Literal Cell

The format of the Arithmetic Literal Cell is the same as that of the HAL/S Compiler. The Arithmetic Literal Cell fields are described below:

<u>Field No.</u>	<u>Description</u>
------------------	--------------------

1	LIT1 Field
---	------------

1A	Unused pad area
----	-----------------

1B	This field contains the information as to the type of cell that is being represented. (One byte field.)
----	---

2	LIT2 Field
---	------------

This is the data. If the data will fit into four bytes, then the data is contained solely within the LIT2 field and the LIT3 field is unused (i.e., it is single precision). If it requires more than four bytes of storage, then the LIT3 field is commandeered and used to store the surplus data (i.e., it is double precision). (Four byte field.)

3	LIT3 Field
---	------------

The LIT3 field is used as an extension of the LIT2 field for double precision data only and is not used when single precision data is stored. (Four byte field.)

2.2.2.2.9.2.2.3 Bit Literal

The Bit Literal Cell contains up to 32 bits of information along with the appropriate length information. The LIT3 field is used to convey this length information.

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Field Size (in bytes)</u>
0	0	0	1A	Unused Pad Space	3
-	-	3	1B	Cell Type (2)	1
1	2	4	2	Bit Pattern (0-32 Bits)	4
2	4	8	3	Length (see Text)	4

Figure 2-90 Bit Literal Cell

The format of the Bit Literal Cell is the same as that of the HAL/S Compiler. The Bit Literal Cell fields are described below:

<u>Field No.</u>	<u>Description</u>
------------------	--------------------

1	LIT1 Field
---	------------

1A	Unused pad area
----	-----------------

1B	This field contains the information as to the type of cell that is being represented. (One byte field.)
----	---

2	LIT2 Field
---	------------

This field contains up to 32 bits of data that may be in a pattern for repetition. The length of this bit field is contained in LIT3. (Four byte field.)

3	LIT3 Field
---	------------

The length field specifies the bit count as determined by the source input. It is always a multiple of four for hexadecimal. For decimal literals only, the length represents the number of significant bits in the literal value. For all others, the length reflects the number of characters in the string specifying the literal, including leading zeros.

2.2.2.9.2.2.4 Template Subscript Literal Cell

The Template Subscript Literal Cell is generated by the HAL/S Compiler Optimizer during the processing of structure subscripts. This allows the computation of the product of the template width and the subscript index to be eligible as a common sub-expression.

<u>Fullword Offset</u>	<u>Halfword Offset</u>	<u>Byte Offset</u>	<u>Field Number</u>		<u>Field Size (in bytes)</u>
0	0	0	1A	Unused Pad Space	3
-	-	3	1B	Cell Type (3)	1
1	2	4	2	Template Symbol Number	4
2	4	8	3	Unused Filler Space	4

Figure 2-91 Template Subscript Literal Cell

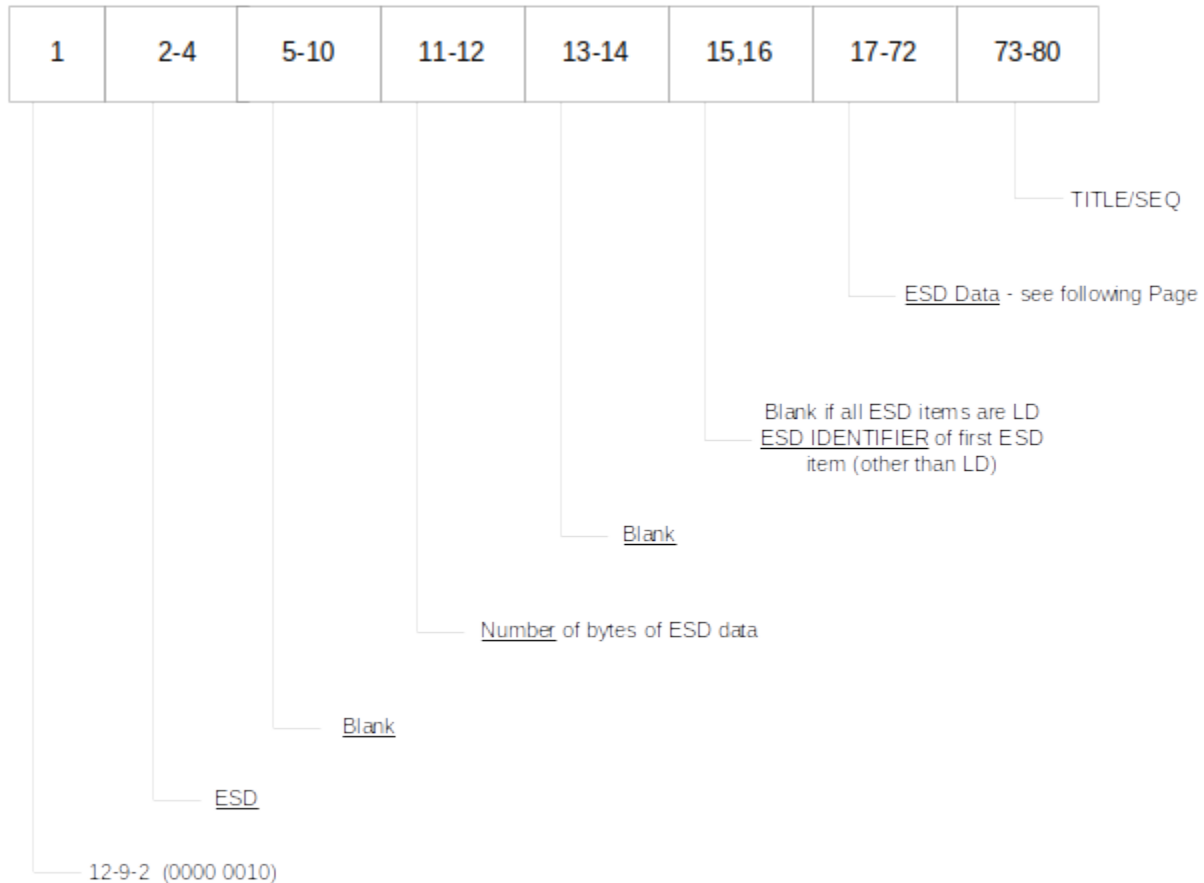
The format of the Template Subscript Literal Cell is the same as that of the HAL/S Compiler. The Template Subscript Literal Cell fields are described below:

<u>Field No.</u>	<u>Description</u>
1	LIT1 Field
1A	Unused pad area
1B	This field contains the information as to the type of cell that is being represented. (One byte field.)
2	LIT2 Field This field contains the Symbol Number (Index into the Symbol Index Table) of the Template.
3	LIT3 Field The LIT3 field is not used.(Four byte field.)

2.3 OBJECT CODE

The object program output appears in both the OUTPUT3 and OUTPUT4 data sets. Symbol table data is located at the beginning of the object deck. The format of the various records which will be found in the object programs is as follows:

OUTPUT3 and OUTPUT4 Data Stets



Notes:

1. All addresses in these formats are byte addresses.
2. The AP-101 "TITLE" name field may contain up to eight characters. Any columns remaining in the sequence field (i.e., name < 8) are used for sequencing.

Figure 2-92 ESD Output Record (Card Image)



Figure 2-93 ESD Data Item

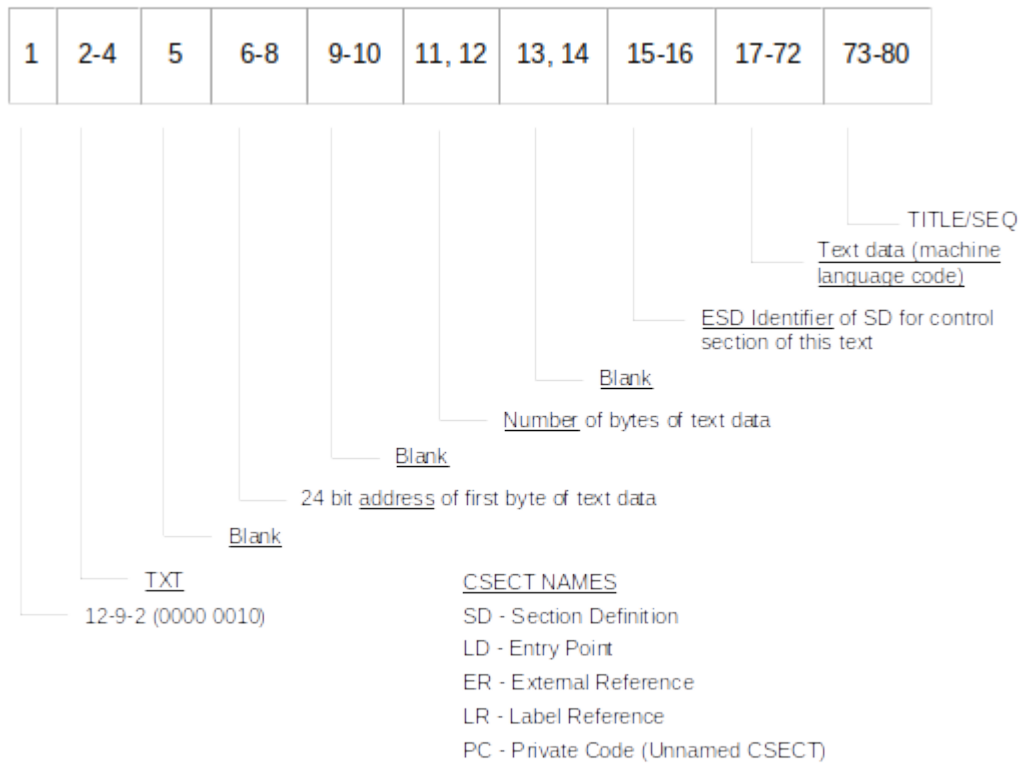
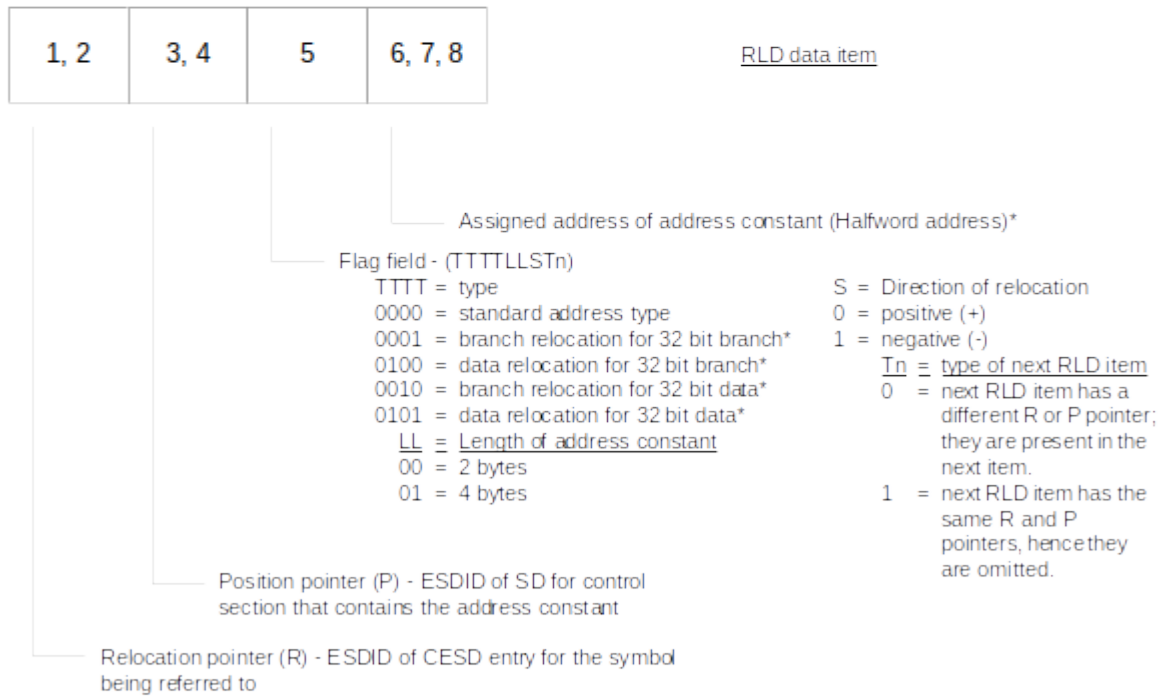
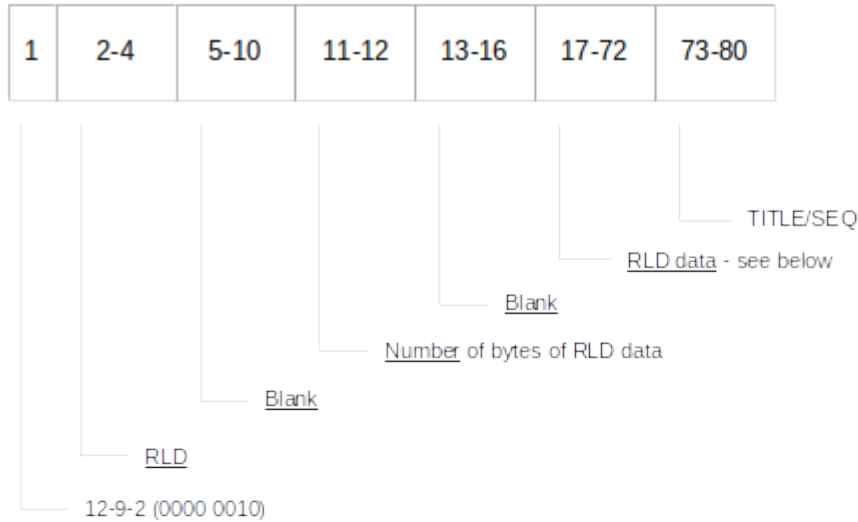


Figure 2-94 Text Output Record (Card Image)



* LL = 00

Figure 2-95 RLD Output Record (Card Image)

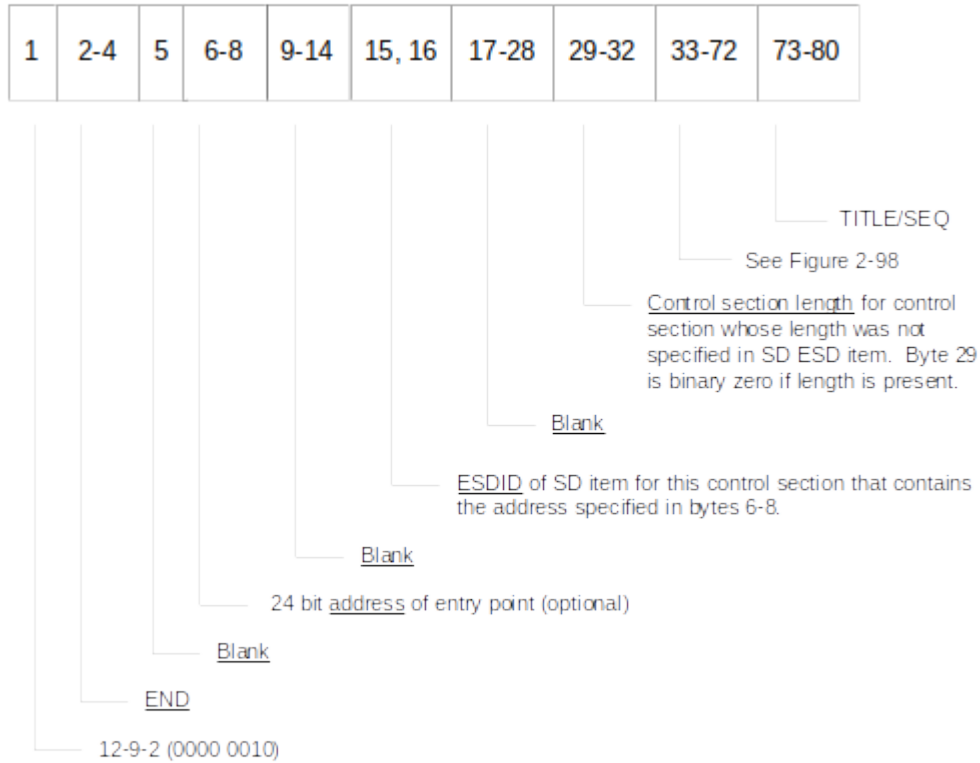


Figure 2-96 END Output Record - Type I (Card Image)

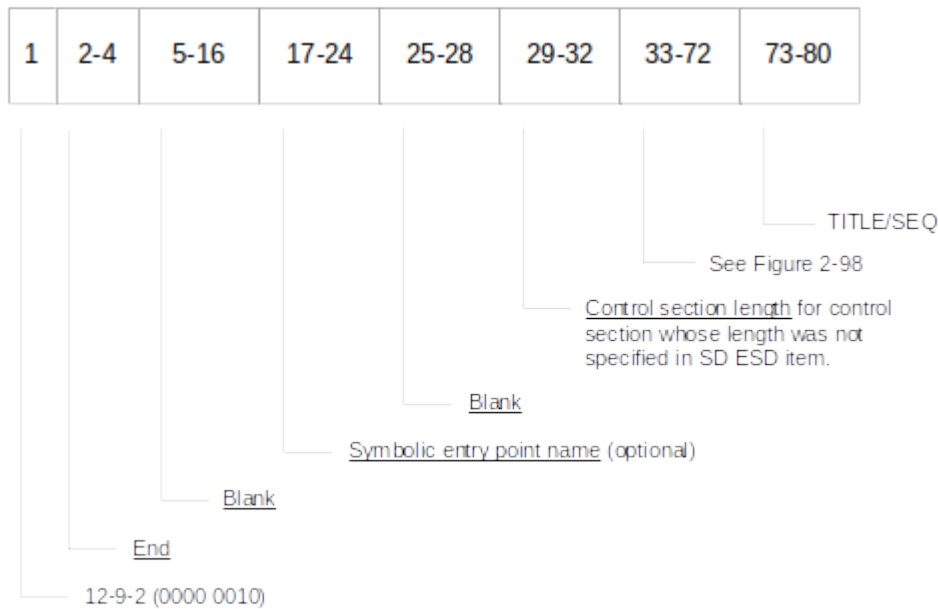


Figure 2-97 END Output Record - Type 2 (Card Image)

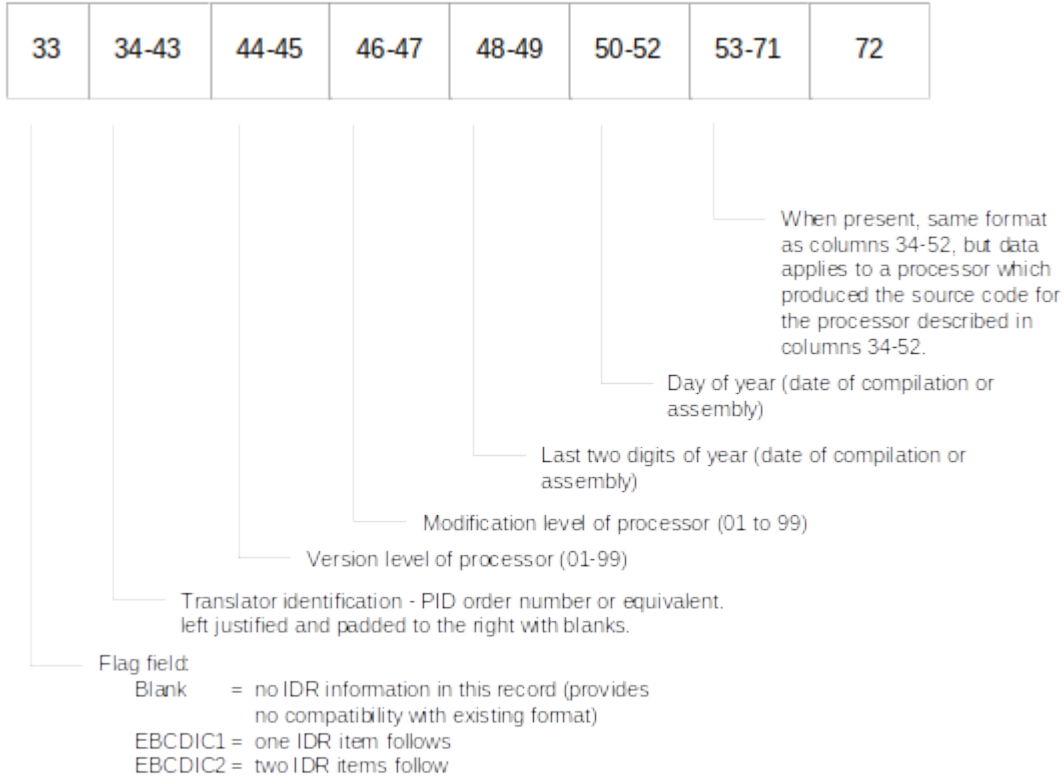
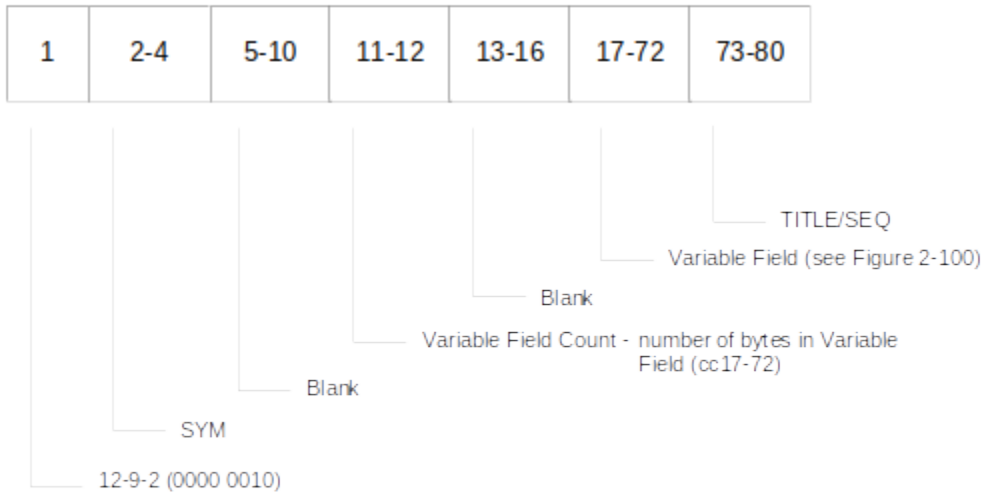


Figure 2-98 IDR Data in a Object Module END Record



Note: If requested by the user, the assembler punches out symbolic information for TESTRAN concerning the assembled program. This output appears ahead of all loader text.

Figure 2-99 TESTRAN (SYM) Output Record - (Card Image)

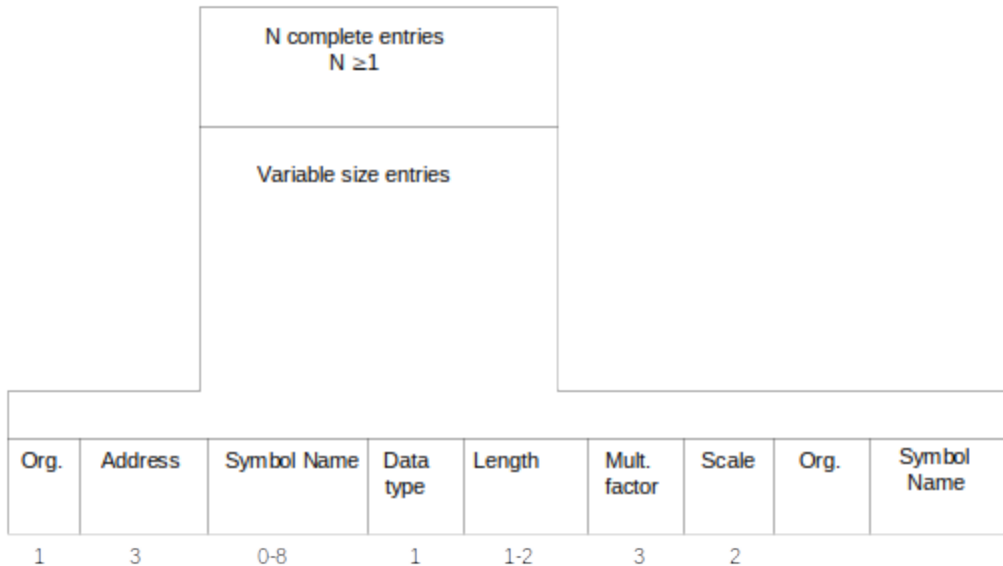


Figure 2-100 SYM Variable Field Data

The variable field (columns 17-72) contains up to 56 bytes of TESTRAN text. The items making the text are packed together. However, no item may extend across a record boundary. Data other than control section data and storage protect data is optional. Names are not allowed on store protect data. Store protect data is only valid within a CSECT.

For a COMMON section the assembler will output only one storage protect item--a storage protect off containing a zero address. For COMMON sections with the same name (including a blank name) the Linkage Editor will output only the first set of TESTRAN records (i.e., the set of records attached to that COMMON section which appears first in the input stream).

The formats of a text card and an individual text item are shown in the two figures above. The contents of the fields within an individual entry are as follow:

1.Organization (1 byte)

Bit 0:

0 = non-data type

1 = data type

Bits 1-3 (if non-data type):

000 =space

001 = control section

010 = dummy control section

011 =common

100 = instruction

101 = CCW

Bit 1 (if data type):

0 = no multiplicity

1 = multiplicity (indicates presence of M field)

Bit 2 (if data type):

0 = Always

Bit 3 (if data type):

0 = no scaling

1 = scaling (indicates presence of S field)

Bit 4:

0 = name present

1 = name not present

Bits 5-7:

Length of name minus one

2.Address (3 bytes) - displacement from base of control section

3.Symbol Name (0-8 bytes) - symbolic name of particular item

Note: The following fields are only present for data-type items.

4.Data Type (1 byte) - contents in hexadecimal

00 = character

04 = hexadecimal

08 = binary

0C = unused

10 = fixed point, full

14 = fixed point, half

18 = floating point, short

1C = floating point, long

20 = A-type or Z-type data

24 = Y-type data

80 = store protect on

84 = store protect off

5.Length is not present for store protect: (2 bytes for character, hexadecimal, or binary items; 1 byte for other types) - length of data item minus 1

6.Multiplicity - M field (3 bytes) - equals 1 if not present

7.Scale - signed integer - S field (2 bytes) - present only for F, H, E, D, P type data, and only if scale is non-zero.

AN EXAMPLE OF THE USE OF SYM CARDS BY HAL/S-FC

Assume:

A. Compilation Unit Name is COMP_UNIT, a COMSUB

B. Version number is 20

C. Stack size is 100

D. References are made to COMSUBS EXT1 of Version 10, and EXT2 of Version 100

Code SYM card information would be:

<u>NAME</u>	<u>TYPE</u>	<u>ADDRESS</u>	<u>COMMENTARY</u>
#CCOMPUN	CSECT	0	Defines CSECT
STACK	DSECT	0	
ARG1	VAR	0	Variables defined in stack
ARG2	VAR	8	
STACKEND	VAR	100	Address of 100 is stack size.
HAL/S-FC	DSECT	20	Invalid label, HAL/S-FC used to indicate beginning of version data. Address of HAL/S-FC is the version of COMP_UNIT.
EXT1	DSECT	10	Version of EXT1
EXT2	DSECT	100	Version of EXT2
HALS/END	DSECT		
#CCOMPUN	CSECT	0	Define statement labels
ST#1	LABEL	0	

ST#2	LABEL	34
ST#N	LABEL	2000
#DCOMPUN	CSECT	2010
	SPOFF	Turn off storage protect
A	VAR	2010
B	VAR	2012

The information between HAL/S-FC DSECTS and the HALS/END can be generated only by the compiler; therefore, no template checking of assembly routines can be accomplished.

The HAL/S-FC compiler must also create a Process Directory Entry (PDE) for each program and for each task. These will be grouped by the compiler and emitted under a single CSECT name (see Section 4.0, "CSECT/MEMBER NAMING CONVENTIONS" on page 1) for each program compilation.

Each PDE consists of six halfwords as described below:

- HALFWORD 1 - binary zeroes. This field is used by the FCOS to store the Process Event. This field must be on a fullword boundary.
- HALFWORD 2 - binary zeroes. FCOS will use this field (PCT field) to point to the Process Control Table.
- HALFWORD 3 and 4 - Z-CON pointing to the program or task entry point.
- HALFWORD 5 - stack address or stack size. If the stack is preallocated by the linkage editor, the address is stored here and bit 0 of halfword 6 is set to 1. If the stack is not preallocated by the linkage editor then FCOS is to allocate it at run time. The stack size is stored here and bit 0 of halfword 6 is set to zero.
- HALFWORD 6 - linkage editor and FCOS flags. Bit 0 is set to 1 if halfword 5 contains the stack address. Otherwise, it is set to zero. The low order four bits of halfword 6 contain the FCOS major function identification (MFID). The MFID can be overridden in a parm field option to any value which is ≤ 15 . The default MFIDs are based on the first character of the compilation unit name as follows:

<u>First Character</u>	<u>Major Function</u>	<u>MFID</u>
A	System Control	1
D	User Interface	2
G	GNC	3
P	PAYLOAD	5

R	RMS	5
S	SM	5
V	VU	6
any other letter	not defined	0

'This page intentionally left blank.'

3.0 AP-101 EXECUTION ENVIRONMENT

The following sections describe the AP-101 execution environment as established by FCOS and HAL/S-FC compiler conventions and design. They discuss the use of HAL/S stacks and the form of the procedure and function CALLS. Additional information in regard to memory and stack content can be found in the HAL/FCOS ICD.

3.1 AP-101 REGISTER USE

The following table describes the use of the AP-101 registers by HAL/S-FC compiler generated code:

<u>Reg.#</u>	<u>Pointer Name</u>	<u>Description</u>
0	BT	Stack register. This register points to the caller's register save area in the run time stack. In addition, all formal parameters, temporaries, and AUTOMATIC variables in REENTRANT procedures are based on this register. The lower half contains the size of the current stack frame.
1	BP	Local data (#D) addressing register. This register is used to address all of the declared variables and literals within a compilation unit.
2		Work addressing register. This register is used to access compool (#P) data, pass address parameters, dereference NAME variables, and set up any other dynamic addressing.
3	BL	Compool (#P) or local data (#D) addressing register. This register is used in SRS instructions only to address a certain subset of the local data in a block (e.g. internal procedures). When the DATA_REMOTE directive is in effect, register 2 can only be loaded with non-local data addresses (e.g. Compool) and register 3 can only be loaded with local data addresses (i.e., #D).
4	RRET	Linkage register. This register records the return address for all subroutine linkages. It may also be used for an integer accumulator.
5 - 7		Used for integer accumulators, index registers, and parameter passage where applicable.
F0 - F5		Used for floating point accumulators and parameters.
F6 - F7		Used for floating point accumulators only.

3.2 HAL/S STACK

The HAL/S stack provides temporary storage for work areas, saving of registers, call and return linkages, passed parameters, and process-specific information for blocks in a HAL/S process execution. The HAL/S stack is so designed that only the storage actually required during the execution of a HAL/S block is present and used. A stack CSECT is created by the AP101/S Linkage Editor process for every program and task in the flight software load module. How stacks are used in the procedure and function linking processes are described in the HAL/S-FC Compiler System Specification.

The format of a stack space element for an executing HAL/S block is shown in Figure 3-101 on page 3. The element is positioned in the stack space in accordance with the hierarchical scope relationship of its block with other blocks in the hierarchy. The meanings of the fields of the element are as follows:

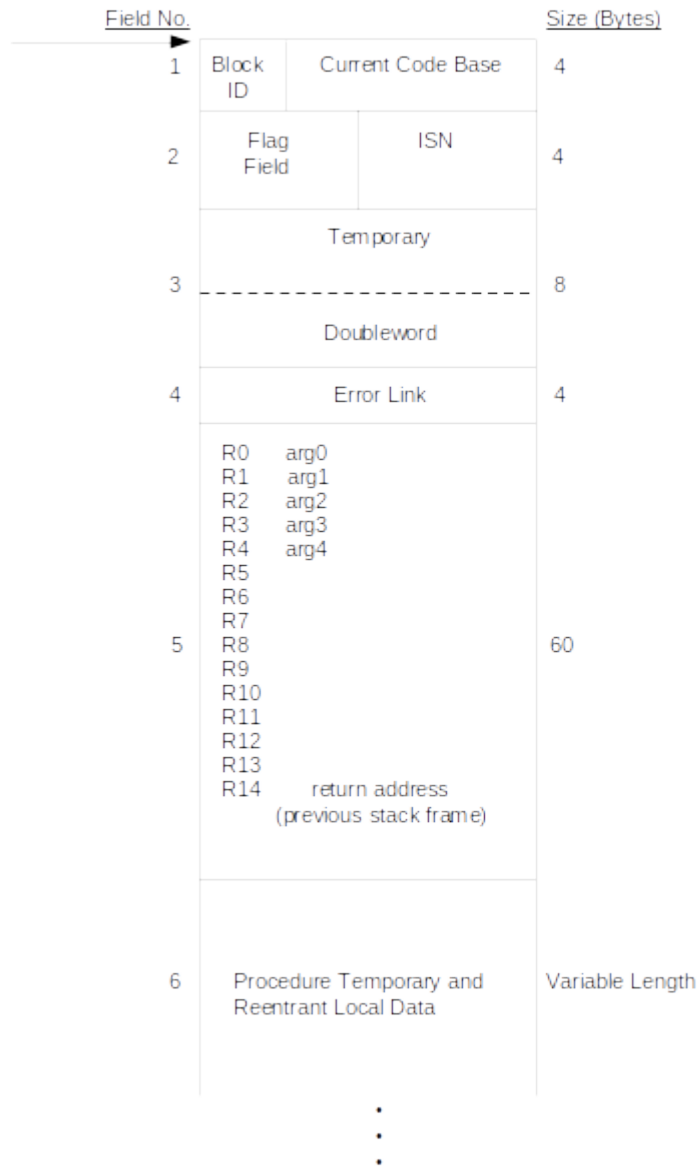
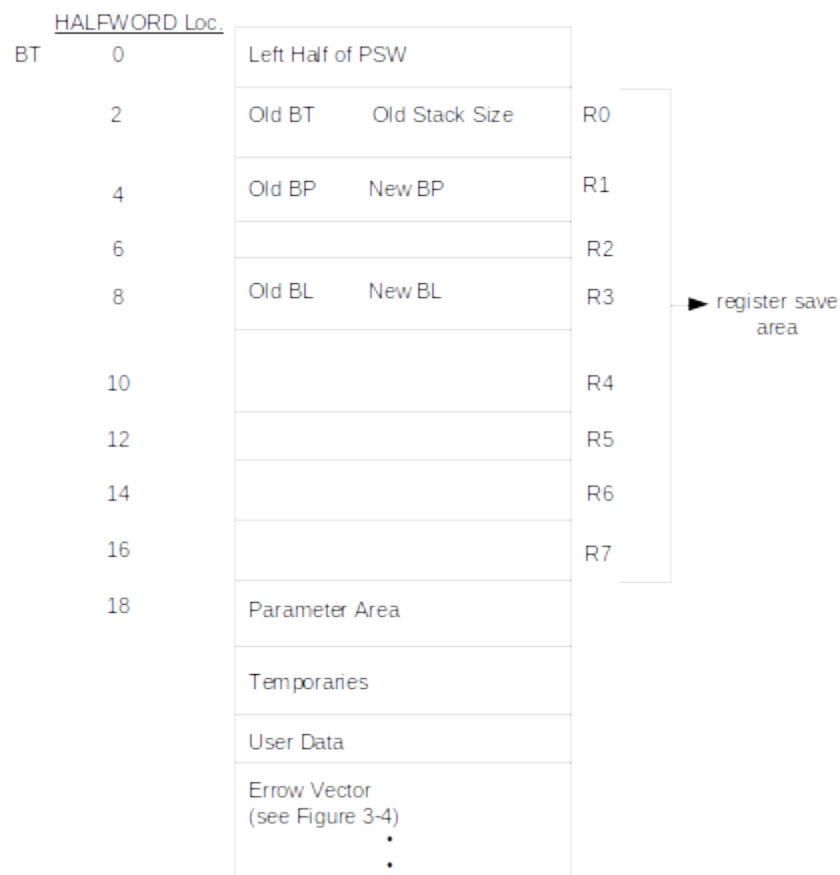


Figure 3-101 Stack Elements

<u>Field No.</u>	<u>Flag Name</u>	<u>Meaning</u>								
1 (First Byte)	Block ID	Block ID is the identification number (one byte) of the HAL/S block, and is generated as follows: <table border="1" data-bbox="812 409 1404 766"> <thead> <tr> <th><u>ID</u></th> <th><u>Block Type</u></th> </tr> </thead> <tbody> <tr> <td>n</td> <td>Program. "n" is the first number available after Include Blocks have been assigned.</td> </tr> <tr> <td>0</td> <td>COMSUBS</td> </tr> <tr> <td>>n</td> <td>Nested procedures, Tasks, and library routines.</td> </tr> </tbody> </table>	<u>ID</u>	<u>Block Type</u>	n	Program. "n" is the first number available after Include Blocks have been assigned.	0	COMSUBS	>n	Nested procedures, Tasks, and library routines.
<u>ID</u>	<u>Block Type</u>									
n	Program. "n" is the first number available after Include Blocks have been assigned.									
0	COMSUBS									
>n	Nested procedures, Tasks, and library routines.									
1 (Bytes 2-4)	Current Code Base	Current Code Base is the entry point address of the called procedure or function. It and the ID are loaded by the Procedure Caller from the linkage data supplied by the invoking procedure.								
2 (Bytes 1-2)	Flag Field	When a HAL/S process enters an EXCLUSIVE block, and successfully gains control of the block, the high order bit of the Flag Field is set to "1." This bit is used during process termination to free the EXCLUSIVE resources which a terminating process may be holding. The remaining 15 bits are zero or contain a lock group number if it is an update block.								
2 (Bytes 3-4)	ISN	Internal Statement Number of the statement referenced last by the Statement Processor.								
3	Temporary Doubleword	This field is used as a workspace by some compiler generated subroutines.								
4	Error Link	Linkage for ON ERROR processing.								
5	Save Registers									
6	Procedure Temporary and Reentrant Local Data	Provides storage space for temporary data.								

3.3 STACK AND LOCAL BLOCK DATA ORGANIZATION

The organization of a HAL/S Stack Block is shown in Figure 3-102 on page 5. The HAL/S Stack Cell contains a save area for the fixed and floating point registers, cells for various pointers, an area for temporary variables, space for user declared variables that are not to be assigned to static storage, and entries for the Error Vector. The active stack space (cell) is pointed to by the pointer BT in register R0. The back link to the previous stack, OLD BT, is established automatically when a new procedure is entered via the SCAL instruction. A pointer, NEW BL, is established for any procedure with a local block data area. If one is not present (e.g. in the case of a HAL/S library routine), the pointer is set to zero. R3 can be used as an additional base register if it is not being used for local data.



- Old BT =
Previous BT, pointer to last stack frame, dynamic stack link (Back Link). This value is zero for the process stack frame (top level)
- Old BL =
Previous R3 (information at time of call)
- New BL =
Current R3, pointer to local data area

Figure 3-102 Stack Organization Cell

The Local Block Data (see Figure 3-103 on page 6) exists to provide information for the block about the Block ID, error conditions, location of the error vector, Lock ID, and EXCLUSIVE and UPDATE SVCs. The Block ID of a hierarchical block is used by the HAL/S code and the Simulation processes to establish appropriate references in the hierarchical block. All other information in the Local Block Data is primarily used by the executing code and FCOS.

		Fields				
BL	1	Block ID			2	
	2	XU	ONERRS	ERRDISP	2	
	3	T Y P	UNUSED	RESERVE SVC #	2	only required if XU = 1
	4	UNUSED		RELEASE SVC #	2	
	5	LOCK ID			2	

Figure 3-103 Local Block Data

The meaning of the fields of the Local Block Data area are as follows:

Field No.

- | | | |
|---|--------------|---|
| 1 | Block ID | The Block ID identifies uniquely the HAL/S Block in a unit of compilation (see Section 3.2, "HAL/S Stack" on page 2). It consists of a Block Number and a Compilation Unit Number. |
| 2 | XU | EXCLUSIVE/UPDATE Block Flag.(1-Bit) Set to one if block is either UPDATE block or an EXCLUSIVE one. |
| | ONERRS | (6-Bits) The number of discrete errors for which an ON ERROR statement exists in the block. |
| | ERRDISP | (9-Bits) The displacement in halfwords from the stack register to the error vector. |
| 3 | TYP | (1-Bit) The bit will be set to one if the data variables are to be read only. It will be set to zero if the data variables are to be written. If this is a reserve/release supervisor call for an EXCLUSIVE procedure or function, the TYP field will be set to zero. |
| | RESERVE SVC# | (8-Bits) SVC number for the release supervisor call which is:
15 for a code block
16 for a data block. |
| 4 | RELEASE | (8-Bits) SVC number for the release supervisor call which is:
17 for a code block |

	SVC#	18 for a data area.
5	LOCK ID	(15-Bits) An identifier indicating which code block or data areas are being used. The identifier for a code block is the address of the EXCLUSIVE DATA CSECT generated for the requested procedure/function.

The identifier for a data area is a bit pattern indicating which data areas are to be reserved or released. The least significant bit corresponds to lock group one. If the master lock was specified the bit pattern will be all ones.

The Error Vector (see Figure 3-104 on page 8) exists for all procedures (PROGRAMS and TASKs) which contain ON ERROR type statements. The location of the Error Vector is determined by a displacement, ERRDISP in the Local Block Data member, off the pointer BT, register R0.

3.4 PROCEDURE AND FUNCTION CALLS

The process of calling internal procedures consists of parameter passing followed by a SCAL. Functions are treated similarly. The form for an external call is identical to an internal call up to the SCAL instruction. External procedures must be called via the long indirect mode. This is required because of the possibility of a bank switch in reaching an external procedure.

SCAL@# 4, #Z Subname (7) Note: Index must not be zero.

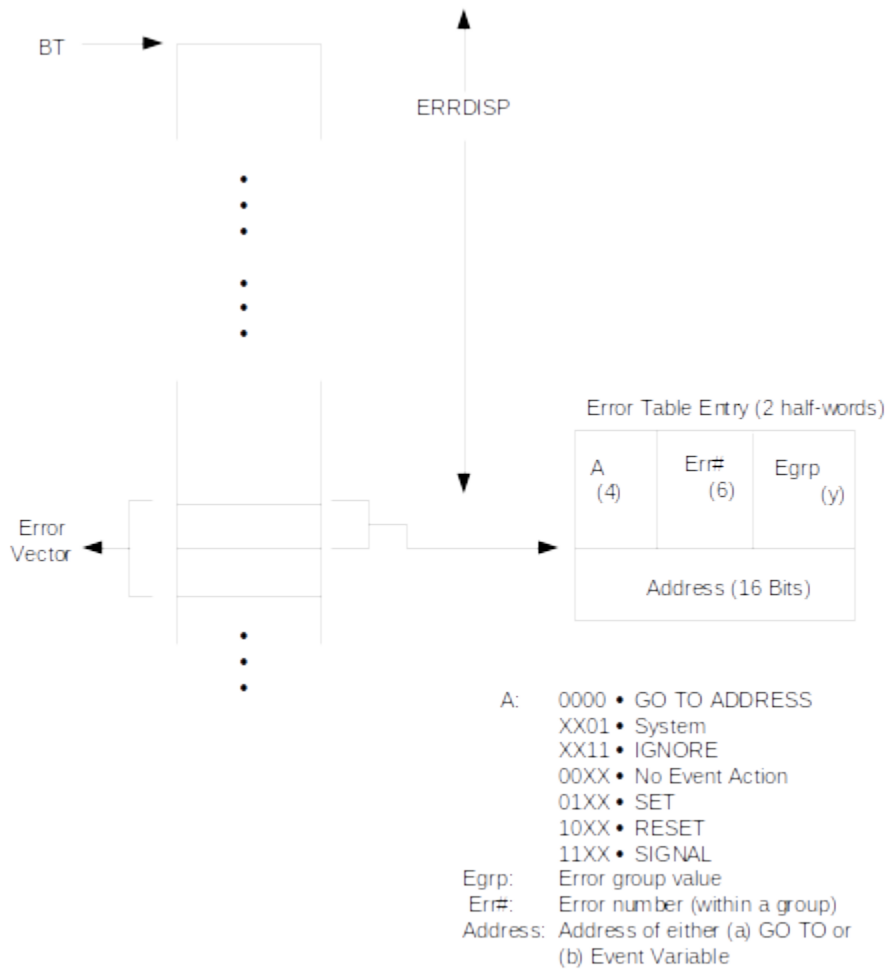


Figure 3-104 Error Vector

4.0 CSECT/MEMBER NAMING CONVENTIONS

The following sections define the HAL/S-FC CSECT naming conventions and indicate how these names are derived.

I. Name notation → CCNNNNNN

A. Code → CC

- 1st – alphabetic or national
- 2nd – alphabetic, national, or numeric

B. HAL/S Compilation Unit name → NNNNNN

- Underscores removed
- 6 characters

- Truncated or padded with blanks

II. CC for CSECT Type

A. CODE (\$ = process entry)

- | | |
|----------------------|--|
| 1.Program | \$0NNNNNN |
| 2.Task | \$cNNNNNN where c = (1-F) for a limit of 15 tasks |
| 3.COMSUB | #CNNNNNN |
| 4.Internal Procedure | anNNNNNN where a = (A-M) and n = (0-9) for a limit of 130 procedures |
| 5.Library Routine | aaNNNNNN where a = (A-Z) |

B. DATA

- | | |
|----------------|-------------------------------|
| 1.Stack | @cNNNNNN where c = (0-9, A-Z) |
| 2.DECLARE data | #DNNNNNN |
| 3.COMPOOL data | #PNNNNNN |

C. SPECIAL

1.ZCON to COMSUB or REMOTE data	#ZNNNNNN
2.ZCON for Library Routine	#QNNNNNN
3.Bank zero	#ONNNNNN
4.Process Directory Entry (PDE)	#ENNNNNN
5.Library data for Library Routine	#LNNNNNN
6.EXCLUSIVE data	#XNNNNNN

III. CC for other member types

- A. Simulation Data File ##NNNNNN
- B. TEMPLATE @@NNNNNN

IV. Placement of CSECT types

- A. CODE (\$0, \$1, A1, AA) → Sectors 2 and greater
- B. DATA (@0, #D, #P, #0, #E, #L, #X)
 - Sectors 0 and 1
 - REMOTE compool data (#P) → any sector
 - REMOTE data (#D with DATA_REMOTE) → sector 7
- C. ZCON (#Z, #Q) → first 2K of Sector 0 (protected)

APPENDIX A EXAMPLE PROGRAM AND SDF DATA STRUCTURES

Appendix A contains an example HAL/S Program, the SDF generated by this HAL/S Program, and structure diagrams illustrating some of the cell relationships for the HAL/S Program.

A.1 EXAMPLE PROGRAM

The following source program was compiled with the HAL/S-FC Compiler using the options:

TBD, TBL, TL, T=TEST TITLE

Some of the SDF data structures as well as a “marked-up” SDF member corresponding to this program are also included. This information illustrates the interaction between the different SDF cells.

```

PROG1:          PROGRAM;          000100AA
                                     000200AA
DECLARE A      INTEGER;          000300AA
DECLARE B      INTEGER;          000400AA
DECLARE C      ARRAY (5, 5, 5);   000500AA
DECLARE E1     SCALAR INITIAL (2.718); 000600AA
DECLARE K      ARRAY (2, 3);     000700AA
DECLARE PI     CONSTA (22/7);    000800AA
DECLARE STR    CHARAC (15) INITIAL 000900AA
                ('INITIAL STRING'); 001000AA
DECLARE SET_STRING CHARACTER (10); 001100AA
DECLARE TWOPI  CONSTANT          (2 PI); 001200AA
DECLARE PI2    SCALAR;           001300AA
DECLARE M      INTEGER;          001400AA
                                     001500
STRUCTURE T:   001600AA
  1 D,         001700AA
    2 E        SCALAR,           001800AA
    2 NE       NAME SCALAR,      001900AA
  1 F          SCALAR;          002000AA
                                     002100AA
DECLARE Q      T-STRUCTURE       INITIAL 002200AA
                (1, NAME(K$(2, 1)), 2); 002300AA
                                     002400AA
STRUCTURE S:   002500AA
  1 G,         002600AA
    2 H        T-STRUCTURE,      002700AA
    2 I        INTEGER,          002800AA
  1 J          NAME INTEGER;     002900AA
                                     003000AA
DECLARE R      S-STRUCTURE(5)    INITIAL 003100AA
                (5#(1, NAME(C$(1, 2, 3)), 2, 3, NAME(A))); 003200AA

```

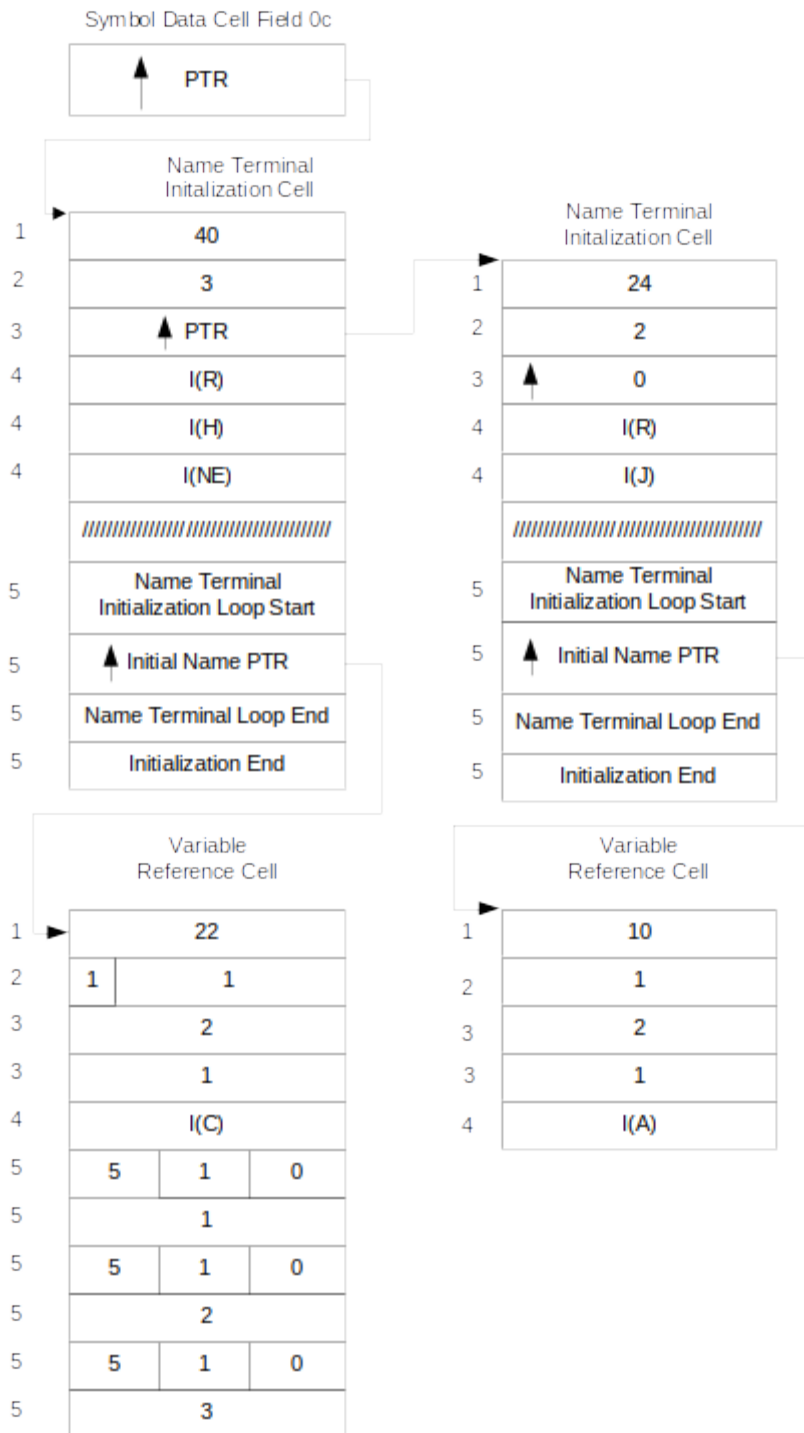
	003300AA
	003400AA
F1: FUNCTION (X, Y) INTEGER;	003500AA
	003600AA
DECLARE X INTEGER;	003700AA
DECLARE Y INTEGER;	003800AA
DECLARE W INTEGER;	003900AA
	004000AA
W = X + Y;	004100AA
	004200AA
RETURN W;	004300AA
	004400AA
CLOSE F1;	004500AA
	004600AA
	004700AA
F2: FUNCTION (Z) INTEGER;	004800AA
	004900AA
DECLARE Z INTEGER;	005000AA
	005100AA
RETURN Z;	005200AA
	005300AA
CLOSE F2;	005400AA
	005500AA
	005600AA
A, B = 1;	005700AA
	005800AA
SET.STRING = 'SET STRING';	005900AA
	006000AA
M = F1 (5 + 4, A + B) ;	006100AA
	006200AA
M = F2 (M + 3) ;	006300AA
	006400AA
A, C\$(A + B, B - Q.D.E +1, 3) =	006500AA
R.G.H.D.E\$(3;) + F1(F2 (Q.F), 4) - Q.D.E **	006600AA
	006700AA
CLOSE PROG1;	006800AA

A.2 STRUCTURE DIAGRAMS

The following SDF structure diagrams illustrate some of the more interesting SDF cell relationships. In these example diagrams, the indexes into the Symbol Index Table are indicated by the convention 'I(variable name)'. The numbers to the left of the cells correspond to the field numbers described for each in the appropriate section of this document. The SDF pointers are represented by arrows and any null pointers are set to zero.

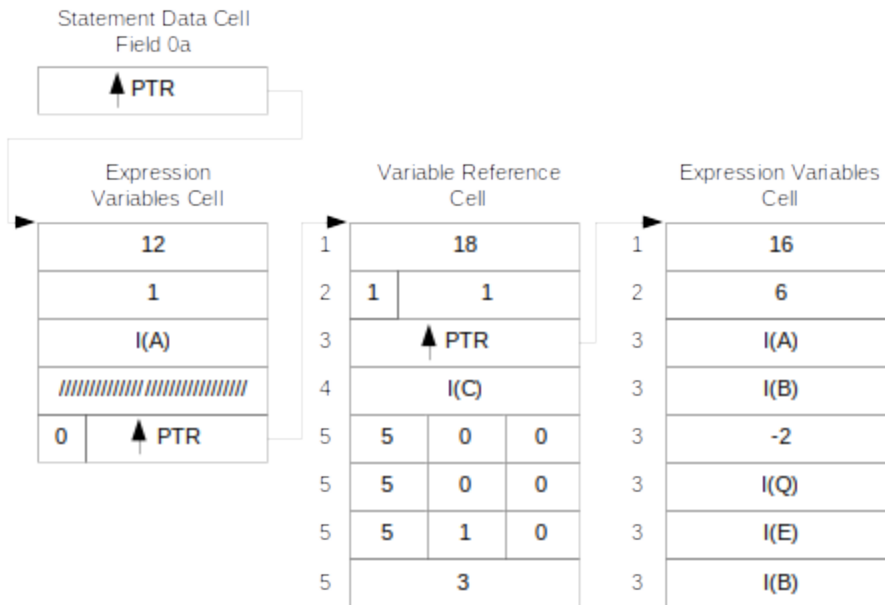
A.2.1 NAME TERMINAL INITIALIZATION CELL

In the example program presented in Appendix A.1, "Example Program" on page A-1, on Statement Reference Number 003100, the variable R is an initialized structure containing NAME terminals. Thus, the Auxiliary Symbol Information Pointer (ASIP, Field 0c) of the Symbol Data Cell points to the following cells:



A.2.2 EXPRESSION VARIABLES CELL

Statement 006500 in the example program (Appendix A.1, "Example Program" on page A-1) contains a subscripted assignment in the LHS context; therefore both the LHS Statement Variables (field 0a) and the RHS Statement Variables (field 0b) of the Statement Data Cell contain pointers to Expression Variables Cells which describe the variables. These data structures are described below:





APPENDIX B CHANGE HISTORY

The HAL-FC/SDL ICD has been revised and issued on the following dates:

Revision	Date	Sections Changed
R6	December 16, 1975	
R7	May 29, 1981	
R8	December 16, 1991	Total Reprint
R9	August 31, 1992	
R10	April 14, 1994	
R11	January 16, 1995	Total Reprint
R12	September 8, 1997	Total Reprint

This is the last page of this document.