Space Flight Operations Contract

HAL/S-FC USER'S MANUAL

PASS 32.0/BFS 17.0

November 2005

DRD - 1.4.3.8-b

Contract NAS9-20000

United Space Alliance

HAL/S-FC USER'S MANUAL

Approved by

Original Approval Obtained
Barbara Whitfield, Manager
HAL/S Compiler and Application Tools

Original Approval Obtained
Monica Leone, Director
Application Tools Build and Data Reconfiguration

DRD - 1.4.3.8-b
Contract NAS9-20000

Revision Log

The HAL/S User's Manual has been revised and issued on the following dates[1]:

Issue	Revision	Date	Change Authority	Sections Changed
29.0/14.0		10/26/98	CR12935A	5.2 - p. 5-10 8.10.3 - p. 8-19 App. B - pp. B-26, B-34
			CR12940	4.2 - pp. 4-5, 4-6, 4-8 8.1.3 - p. 8-2 App. B - p. B-10
			DR101047	8.11 -p. 8-27
			DR109063	App. B - p. B-35
			DR109076	5.2 - p. 5-10
			DR109077	App. B - p. B-27
			DR109079	App. B - p. B-59
			DR109081	8.1.2 - p. 8-2 8.11 - p. 8-27 App. B - p. B-12
			DR109083	8.11 - p. 8-27
			DR109086	App. B - p. B-16
			DR109091	4.1 - p. 4-2
			DR109092	8.1.3 - p. 8-2
			DR109097	App. B - pp. B-16, B-63
			Cleanup	Title page, Signature page, Table of Contents, List of Figures
				2.4.2 -p. 2-8 2.5.1 -p.2-19 4.1 - pp. 4-1, 4-2 4.2 - p. 4-5 5.2 - pp. 5-6, 5-7, 5-8, 5-9, 5-10 7.2.2.2 -pp. 7-3, 7-4 8.1.2 - pp. 8-1, 8-2 8.1.3 - p. 8-2 8.2 - pp. 8-3, 8-4 8.4 - p. 8-5 8.6 -p. 8-7 8.9 - p. 8-17 8.11 - pp. 8-19, 8-20, 8-21, 8-22, 8-23, 8-24, 8-25, 8-27

1. A table containing the Revision History of this document prior to the USA contract can be found in Appendix E.

Issue	Revision	Date	Change Authority	Sections Changed	
				App. B	-pp. B-12, B-35, B-59
				App. C	- pp. C-1, C-2, C-3
30.0/15.0		06/09/00	CR12214	4.2	-p. 4-8
				App. B	-pp. B-3, B-64
			CR13211	App. B	-pp. B-3, B-64
			CR13212	App. B	-pp. B-17
			CR13236	App. B	-pp. B-17
				8.11	-p. 8-20
				App. B	-pp. B-34, B-35
			CR13245	5.1	-p. 5-4
				8.10	-p. 8-19
				App. B	-p. B-62
			CR13273	4.3	-p. 4-11
			DR111314	5.1	-p. 5-6
			DR111320	8.12	-pp. 8-29, 8-30
			DR111326	8.12	-p. 8-29
			DR111337	8.7	-pp. 8-12, 8-13
				App. B	-p. B-65
			DR111340	8.12	-p. 8-29
			DR111349	4.1	-p. 4-3
			Cleanup	Preface	
				4.1	-p. 4-3
				8.9	-p. 8-18
				App. B	-p. B-12
				Index	
31.0/16.0		09/14/01	CR13220	App. B	-p. B-35
			CR13335	8.6	-p. 8-10
			CR13372	8.8	-pp. 8-15 thru 8-18
			DR111356	4.2	-pp. 4-6, 4-7
			DR111362	App. B	-p. B-4. B-75
			DR111365	4.2	-p. 4-5
			DR111367	8.1.2	-pp. 8-1, 8-2
				App. B	-p. B-21, B-24, B-60
			DR111369	8.12	-p. 8-30
			DR111371	8.12	-p. 8-33
			DR111376	8.12	-p. 8-33, 8-34
			DR111379	App. B	-p. B-30
			DR111380	App. B	-p. B-12

Issue	Revision	Date	Change Authority	Sections Changed
			Cleanup	Preface -p. 5-5 5.1 -p. 5-11 5.2 -p. 8-8, 8-9 8.6 -p. B-73 App. B
32.0/17.0		11/05	CR13538	8.7 -p. 8-10, 8-13, 8-14 8.11 -p. 8-23 8.11.1 -p. 8-23 8.11.2 -p. 8-23 (deleted) App. B -p. B-23, B-38, B-39, B-40, B-70
			CR13570	8.7 -p. 8-14, 8-15, 8-16 8.8 -p. 8-17 App. B -p. B-37, B-70
			CR13571	App. B -p. B-9, B-25, B-37, B-38, B-40
			CR13613	2.1 -p. 2-1 2.2.3 -p. 2-4 2.4.3.1 -p. 2-11 2.5.1 -p. 2-21 2.5.2 -p. 2-21 2.5.2.3 -p. 2-23 (deleted) 2.5.3 -p. 2-23, 2-24 3.3.1 -p. 3-3 5.1 -p. 5-3 5.2 -p. 5-7 6.1 -p. 6-1 6.1.2 -p. 6-2 6.1.3 -p. 6-3 6.1.4 -p. 6-3 6.2 -p. 6-4 7.0 -p. 7-1 (deleted) 8.2 -p. 8-3, 8-4 8.3 -p. 8-4 (deleted) 8.4 -p. 8-5 8.9 -p. 8-19 App. A -p. A-1, A-3, A-5 App. C -p. C-1, C-3
			CR13615	App. B -p. B-12
			CR13652	8.7 -p. 8-11 App. B -p. B-38

Issue	Revision	Date	Change Authority	Sections Changed
			CR13670	4.2 -p. 4-7 5.1 -p. 5-1, 5-5, 5-6 App. B -p. B-13
			CR13811	App. B -p.B-4, B-70, B-73
			CR13813	App. B -p. B-3, B-72
			CR13832A	5.1 -p. 5-2
			CR13956	8.2 -p. 8-4 8.13 -p. 8-34 App. B -p. B-22
			CR14216A	Preface 1.0 -p. 1-1
			DR111386	8.13 -p. 8-36 App. B -p. B-9
			DR120220	4.2 -p. 4-7
			DR120226	App. B -p. B-24
			DR120228	App. B -p. B-72
			DR120262	8.13 -p. 8-31, 8-32
			DR120263	App. B -p. B-73
			DR120268	8.13 -p. 8-36
			DR120271	8-7 -p. 8-12, 8-14
			PCR0780	2.2.1 -p. 2-2 App. B -p. B-38

List of Effective Pages

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

Page No.	Change No.
All	32.0/17.0

Preface

The HAL/S-FC User's Manual was developed by Intermetrics, Inc. and is currently maintained by the HAL/S project of United Space Alliance.

The intent of the manual is to provide the necessary information to compile and execute a HAL/S program. It is not, however, intended to serve as a guide to the HAL/S language.

At the time the *HAL/S-FC User's Manual* was origianlly written, HAL/S work was done on an IBM 360 computer and JCL job cards were used to execute jobs. Subsequently, a different mainframe computer has replaced the 360 computer and JCL statements are used instead of JCL job cards. Therefore, to be as accurate as possible throughout this document, the reader should substitute "mainframe" wherever "360" is employed in reference to the computer and substitute "statements" wherever "cards" is used in reference to JCL.

The primary responsibility is with USA, Department, 01635A7.

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

Table of Contents

1.0 INTRODUCTION 1-1
2.0 USING HAL/S ON THE AP-101 2-1
2.1 Introduction 2-1
2.2 The Steps Involved 2-1
2.2.1 Compiling 2-1
2.2.2 Linkediting 2-3
2.2.3 FC Simulation 2-4
2.3 Data 2-4
2.3.1 Datasets 2-4
2.3.2 Source Program Format 2-6
2.4 Introduction to JCL 2-7
2.4.1 The JOB Card 2-7
2.4.2 The EXEC Card 2-8
2.4.3 The DD Card 2-11
2.4.3.1 Defining Libraries of Programs: JOBLIB and STEPLIB 2-11
2.4.3.2 The DISP Parameter: Specifying the Status of Datasets 2-12
2.4.3.3 Defining Input Data 2-13
2.4.3.4 Defining Output Datasets 2-14
2.4.4 Catalogued Procedures 2-16
2.4.4.1 Writing Catalogued Procedures 2-16
2.4.4.2 Using Catalogued Procedures 2-18
2.4.4.3 Modifying Catalogued Procedures 2-20
2.5 Using Catalogued Procedures to Process a HAL/S Program 2-21
2.5.1 Introduction 2-21
2.5.2 How to Use the Catalogued Procedures 2-21
2.5.2.1 To Compile 2-22
2.5.2.2 To Compile and Link Edit 2-23
2.5.3 Standard DDnames and Outputs Associated with the Catalogued Procedures 2-23
3.0 PROCESSING PROGRAM COMPLEXES 3-1
3.1 Program Complexes 3-1
3.2 Templates: The INCLUDE Directive 3-1
3.3 Processing a Program Complex 3-3
3.3.1 When All Compilations Are Performed in One Job 3-3
3.3.2 When Compilations Are Done At Different Times 3-6
3.4 Template Generation 3-8
3.5 SDF INCLUDEs 3-9
4.0 COMPILER LISTINGS 4-1
4.1 Format of the Compiler Listing 4-1
4.2 Information Provided by the Compiler 4-4
4.3 Error Messages Produced By the Compiler 4-9
5.0 USER-SPECIFIED OPTIONS 5-1

5.1 Compiler Options 5-1
5.2 Compiler Directives 5-6
6.0 HAL/S INPUT-OUTPUT OPERATIONS 6-1
6.1 Sequential I/O 6-1
6.1.1 PAGED and UNPAGED Channels 6-1
6.1.2 Input Data Formats 6-1
6.1.3 Output Data Formats 6-2
6.1.4 JCL Considerations 6-3
6.2 File I/O 6-4
7.0 THE HAL/S USER CONTROL PROGRAM (HALUCP) 7-1
8.0 IMPLEMENTATION DEPENDENCIES 8-1
8.1 Compile Time Characteristics 8-1
8.1.1 Character Set 8-1
8.1.2 Data Size Restrictions 8-1
8.1.3 Program Organization Limits 8-2
8.2 Runtime Characteristics 8-3
8.3 Real-Time Statements 8-4
8.4 Runtime Errors 8-4
8.5 Access Rights 8-5
8.6 Language Subset Restriction Capability 8-7
8.7 %MACRO Implementation 8-9
8.8 Arrayed Addressing 0th Element and % NAMEADD 8-16
8.9 CSECT Naming Conventions 8-18
8.10 Character Code Conventions 8-20
8.11 Remote Data 8-22
8.11.1 NAME Variable Initialization Restriction 8-23
8.11.2 Other Restrictions 8-23
8.12 DENSE attribute 8-24
8.13 User Note's 8-24
Appendix A PROTOTYPE CATALOGUED PROCEDURES A-1
Appendix B COMPILE-TIME ERROR BEHAVIOR B-1
Appendix C EXECUTION-TIME ERRORS C-1
Appendix D HAL/S-FC RUNTIME LIBRARY NAMES D-1
Appendix E CHANGE HISTORY E-1

List of Tables

Table 2-1 Catalogued Procedures Available for Processing HAL/S Programs 2-21
Table 8-1 HAL/S-FC CONVENTION FOR CHARACTER INPUT..... 8-21
Table 8-2 DEU Character Set..... 8-22

This page is intentionally left blank.

List of Figures

Figure 4-1 Sample Compilation of Source File 4-4
Figure 8-1 Algorithm for Calculating the 0th Element Offset..... 8-17
Figure A-1 HALFC A-4
Figure A-2 HALFCL A-5

This page is intentionally left blank.

1.0 INTRODUCTION

The purpose of this manual is to provide the information needed by a programmer to compile and execute a HAL/S program. Compilation will take place on the IBM 360/75 computer, producing code for the AP-101 Flight Computer. Simulated execution of AP-101 programs also takes place on the IBM 360.

The manual is intended both as an introduction for a first-time user of HAL/S on the AP-101, and as a reference document to be used by experienced HAL/S programmers. It is not intended as a guide to the HAL/S language, and a knowledge of HAL/S syntax and programming techniques is assumed throughout.

Chapter 2 is directed toward first-time users of the language, and includes a discussion of the key concepts involved in compiling and running a program, as well as an introduction to the use of JCL for communicating with the 360 operating system. The use of catalogued procedures to compile, link-edit, and simulate AP-101 execution of a HAL/S program is also explained. Chapter 3 describes the procedures involved in linking together the elements of a HAL/S program complex, again directing itself to a relatively inexperienced user.

Later chapters tend to assume greater levels of user sophistication, and in particular Chapter 8 assumes a knowledge of the Support Software for the AP-101. Information is provided about the printed output automatically produced by the compiler, about various user options available for compiling programs, and about HAL/S I/O operations. Use of the HAL/S User Control Program to run AP-101 simulations on the 360 is also described in detail. Finally, a complete account of various HAL/S characteristics specific to the use of HAL/S on the AP-101 is given - including the standard character set employed, data size and program organization limits, details of the real-time implementation, and the %macros available. A series of appendices includes a great deal of useful information for reference purposes.

The user may wish to consult the following reference manual for further information about certain topics to be discussed below.

AP101S Assembler User's Guide:
USA000516

This page intentionally left blank.

2.0 USING HAL/S ON THE AP-101

2.1 Introduction

The purpose of this chapter is to enable a first-time user of HAL/S to compile and link a program for execution on the AP-101 computer. It is assumed that the user already knows the HAL/S language, as described in the Language Specification and Programmer's Guide, and is now interested in putting that knowledge to use.

Unfortunately, running a HAL/S program (or a program in any other high-level language, for that matter) is not simply a matter of typing it into the machine, pushing a button, and getting a result. The computer with which you are dealing as a physical entity does not "know" HAL/S; if it can be said to "know" anything, it knows a machine language. Furthermore**,** it typically finds itself confronted at any given moment with a large number of demands on its attention - TSO users, batch jobs, operator commands, etc. - of which your own HAL/S program is merely one. Each job it must perform requires specific amounts of storage to be set aside, library routines to be found and readied for use, and peripheral devices to be prepared for reading from or writing into.

Thus, before a HAL/S program can be executed it must be translated into a form which the machine can understand, storage for it must be allocated and other hardware preparations made, and the computer must be instructed to run the program and must then decide when and where to do so. Running a HAL/S program, therefore, requires a certain amount of communication with the computer prior to the performance of any specific operations associated with the program itself. In this chapter, the user will be shown the various forms which this communication may take.

2.2 The Steps Involved

A series of preliminary steps must be performed in order to run a program. Before the execution of the program, it must be compiled and then link edited. Only by passing through each of these three steps (compilation, link editing, and execution) can a user's HAL/S program actually be run.

2.2.1 Compiling

HAL/S is a language designed to be relatively easy for someone to use, even if he or she has no understanding of the actual hardware within the machine or how it works. HAL/S statements are given in phrases which closely resemble natural English; numbers can be specified in decimal form, character strings are written just as they are to appear, and so on. In order for a HAL/S program to be run, the statements in the program must be translated into a form in which the computer can follow them. The 360 contains a "HAL/S compiler" in order to perform this task. Itself a program written in a language called XPL, the compiler takes the user's HAL/S program, called the "source code", and produces from it a set of specific machine instructions, called the "object code". The object code is written in a form which, although not easily comprehensible to the average user, is simple for the computer to execute and allows it to carry out precisely the operations specified in the source code.

Thus, before any program can be run, it must first be compiled. Notice that although typically one may wish to run a program several times, one need only compile it once. One must be careful to distinguish between the translating of HAL/S commands into a set of instructions, and the actual carrying out of these instructions; this is precisely the distinction between compiling and executing a program. In compilation, something is done to the instructions: they are translated from HAL/S into machine language. In execution, something is done with them: they are obeyed. The input to the compilation step is a source program; the input to the execution step, if anything, is a set of data. This distinction should be kept clear throughout the following discussions.

In point of fact, the HAL/S compiler does more than simply translate. Its action consists of "phases", or passes over the source code. Only the most important will be discussed here.

Phase 1 performs a syntax check, in which each HAL/S statement in the user's program is checked for syntactical correctness, as defined in the Language Specification. The overall program structure is also checked, (i.e., that DO…END statements match and that such groups are correctly nested, that block headers have corresponding CLOSE statements, that all variables are declared, etc.). Most user errors (of a syntactical kind) are discovered in this phase: if you write DECLARE M MATRIX VECTOR, or leave out a terminating semi-colon or an END on a DO group, or CALL something which you have declared as a FUNCTION, you will hear about it at this point. Phase 1 emits a series of error messages if any errors are encountered, and it also produces a standardized and carefully formatted listing of the source code (see Chapter 4).

The next phase, Phase 1.5, is an "optimization" phase. The compiler makes a series of standard changes to the program, to the extent that these are possible, so that the time the program will take to execute is minimized. Certain kinds of egregious programming inefficiencies are thereby avoided. For example, a program containing the following code:

```
DO FOR I = 1 TO 500;
     X$I = I + SQRT(B**2-4AC);
END;
```

would be grossly inefficient, insofar as the unchanging quantity SQRT(B**2-4AC) would have to be calculated 500 times. Better code would be:

```
Y = SQRT(B**2-4AC);
DO FOR I = 1 TO 500;
     X$I = I + Y;
END;
```

But if you have written a program with the former code, Phase 1.5 of the compiler would catch it and correct it for you automatically so that the ultimate object code produced would only instruct the computer to perform the SQRT(B**2-4AC) calculation once.

In Phase 2, the actual machine instructions are generated. Again in this phase certain errors may be discovered, in which case error messages will be produced. Optionally, you may ask Phase 2 to print out the instruction set it generates from your source code; to programmers familiar with assembly language this may be very useful for debugging purposes. The essential output from Phase 2 is what is known as an "object module" - the object (machine language) code which is stored in the computer for later use and which is the subject of all later steps. Again it is important to emphasize that compilation of a program need only take place once, and that therefore only one object module need be produced. The object module, in turn, may be repeatedly employed so that more than one execution of the program takes place.

Finally, Phase 3 of the compiler produces a series of diagnostic tables giving information about the compilation, the source program, and the object code produced. These tables, called Simulation Data Files, or SDFs, help supply useful information both to the programmer and to various debugging and diagnostic systems which are available to HAL/S users. They can also be used to pass information about HAL/S programs to other HAL/S compilations which may refer to them. More detail about SDFs will be provided later (see Sections 3.5 and 4.2).

2.2.2 Linkediting

The object code produced by the compilation step, although written in the machine language used by the computer, is still not in "executable form." It must first be processed by the link editor. The reason for this is as follows:

Almost every HAL/S program contains what are known as "external references", i.e., references to code located outside the object module produced by the program being compiled. Such references may be to user-written blocks of HAL/S code, such as external PROCEDUREs or FUNCTIONs, or to other programs (see Section 3.4 for more details), or they may be to one or more of the HAL/S built-in functions (e.g., SIN, ARCCOS, MAX, DATE, etc.), or to external HAL/S variables defined in COMPOOLs. Even a program which seems totally self-contained and does not explicitly call any built-in functions will in general include external references because the object code produced by the HAL/S compiler itself always calls certain procedures in the HAL/S runtime library.

From the point of view of the computer, an external reference is an instruction within one object module which tells it to look to a separate object module to find either its next instruction, or else some piece of data. Of course, it may be the case that this other module does not exist yet, or has not been compiled yet; and even if it does, the compiler will generally not know where in the computer's memory it resides. The referencing instruction the compiler produces is therefore an open-ended one; it instructs the machine to look in another location for its next instruction, but it does not specify where precisely that location is. The external references produced by the compiler in an object module are accordingly called "unresolved ones". It is because of these unresolved external references that an object module is not yet fully executable by the machine.

Once all the external programs referred to by an object module exist, however, it becomes possible to "resolve" the external references by replacing them with specific

information as to where the next instruction is to be found. This is the task of the link-editor. The input to the link-editor is thus a series of object modules, each containing external references to some other object module in the series. These modules may be the result of compilations of user-written HAL/S programs, standard procedures called by the compiler, or members of the runtime library of built-in functions and procedures called by the compiler. The link-editor goes through each of these modules and resolves all of the external references by replacing each one with a specific location. It can therefore be thought of as literally "linking" the object modules together to form one larger unit. This unit is the output of the link-edit step; it is called a load module and consists of fully executable code.

AP-101 object modules can be link-edited by a program called LINK101. A simple set of commands, to be described below in Section 2.5, will permit link-editing to take place without any user difficulty.

2.2.3 FC Simulation

Compilation, as has been stated, can be viewed as the execution of a program (the compiler), which takes as input the source program and produces as output the object code in a machine language the computer can understand. The same is true of link editing: it takes one or more object modules as input and produces a single fully executable load module as output. It follows from this that compilation and link editing of a source program may actually take place on a different computer than the one which will ultimately be instructed to execute it. Thus, computer A may contain a compiler program which translates HAL/S source into the machine language not for computer A, but rather for a different computer B. Such a compiler in A produces object code which is not executable by A but which can be transferred, by some physical device for example, into B where execution can take place. With such a set-up, in fact, computer B need not have any compiler in it at all.

This is precisely what happens with HAL/S code written for the AP-101. The HAL/S compiler for the AP-101 (the 'target' machine) resides and is executed on the 360 (the 'host' machine). Source programs are entered into the 360, compiled and link edited there, and only then transferred in the form of completed load modules to the AP-101. This is done because the physical AP-101 computer on which programs will ultimately run is, in general, not easily available to the user.

This manual will not concern itself with the actual execution in real-time of a HAL/S program on an AP-101 computer. Rather it will concentrate on the AP-101 compiler and link editor in the 360.

2.3 Data

2.3.1 Datasets

All information to be stored by the computer can be thought of as data. This includes not only the pieces of (typically numeric) information which will be used as input to your program, but also the program itself, as well as the object and load modules produced by

the compilation and link edit steps. Data may be entered into the computer in many different ways: it may be typed directly into a terminal, produced as output from some program, or transferred from some other computer by means of tape. In the computer, data is stored in a "dataset" or a "file". A dataset is simply a collection of related data. Thus, for example, if you wanted to write a program which would examine each one of a large number of scientific measurements and calculate a set of statistics about them (mean, median, correlations, etc.), your program would be in one dataset, the input to the program would be in another, and as the program ran you would want the output to be saved in a third dataset for later reference. Object and load modules produced by the compilation and link editing of your program would exist in datasets as well.

Every dataset has a name, defined either by the user or by the system, and it is this which gives the organization of data into datasets its flexibility. Without some such naming convention, referring to a location in memory would require knowing its exact address - a project time-consuming at best, and, due to the amount of continual shifting going on in the computer's main memory, at worst impossible. Organizing data into datasets thus allows the computer to store large amounts of data while still making it possible for a user to have access to just the data he or she wants without needing to know exactly where it is. Calling datasets "files" produces a useful metaphor: just as it is easier to ask a file clerk for a folder by name rather than as "the eighteenth folder from the back of the third drawer", it is easier to refer to collections of data in the computer by dataset name than by physical location. It is harder on the file clerk, of course: but one advantage of computers is that they do not seem to mind engaging in long searches. Thus, although strictly speaking, within the machine all the data in a dataset may not actually be physically stored in the same location, it is helpful for the user to think of a dataset as a "place" in the computer's memory where a collection of related data is kept, and which can be referred to by name.

Datasets may be organized in one of two ways: sequential or partitioned. A sequential dataset is simply a collection of data, organized into lines. A partitioned dataset (or PDS) can best be thought of as a collection of sequential datasets called 'members'. Partitioned datasets consist of a directory (like a table of contents) listing the name and location of each of its members. A partitioned dataset may have several members or it may contain only one member. Each member must be sequential in organization. Libraries of programs, for example, are generally stored in partitioned datasets; and load modules (i.e., programs ready to be executed) must be so stored.

Datasets, whether sequential or partitioned, may be named by the user as he or she wishes, subject to certain restrictions. Documentation for your specific installation should be consulted. Often the system requires that the name of a dataset be followed by a qualifier indicating what sort of file it is (simple data, a load module, or object file, etc.). Members of partitioned datasets are referred to by appending their name, in parentheses, to the name of the partitioned dataset. Thus the member PROG1 of the PDS PROGLIB would be referred to as PROGLIB.LOAD (PROG1), if LOAD is the qualifier specifying a load module.

The full name of a dataset usually begins with the user's id, as this is known to the

system. Thus, a dataset called POSITIONS, of DATA type, created by user VIL1917, could be referred to as 'VIL1917.POSITIONS.DATA'.

2.3.2 Source Program Format

In order for the HAL/S compiler to translate HAL/S programs into machine language, certain coding conventions must be followed in the source code. The most important of these is the first column of each line of source text determines the type of that line as follows. A "C" in the first column indicates that this is a comment line, to be ignored by the compiler. A blank (or, alternatively, an "M"), indicates a main line of source text. If multi-line format is being used in the source program (which is rare), "E" and "S" in the first column indicate exponent and subscript lines respectively. Compiler directives, indicated by a "D" in the first column, are described in Sections 3.2, 3.5, and 5.2. In general, one should get into the habit of always skipping the first column. Lines which begin there by mistake will be flagged by the compiler with an error message and ignored.

Otherwise there are no serious restrictions on how code is to be entered. HAL/S, being stream oriented rather than card oriented, allows more than one statement to be entered on a line, as well as permitting statements to be broken up over several lines. Good programming practice would suggest, however, that one statement per line be employed as much as possible. Semi-colons rather than end-of-line characters are used as delimiters in HAL/S, and you must therefore be careful to end every statement with a semi-colon; failure to do so will inevitably be disastrous. As with skipping the first column, this should become a habit after a little experience. Blanks are used to separate words; whether one or more are used is immaterial, but at least one must be used, especially to indicate multiplication (exception: two parenthesized expressions may be multiplied without an intervening blank). Blanks are illegal within labels or identifiers. A procedure named COMPUTE ORBIT is illegal and will cause an error; one named COMPUTE_ORBIT, however, is not. Notice that wherever blanks are legal in the source text, an imbedded comment is also legal. These are indicated by delimiting the comment by the character pairs /* at the beginning and */ at the end.

2.4 Introduction to JCL

In order for any program (including compilers, link editors, as well as user-written source programs), to be run on the 360, the operating system must be informed of where the program is located, who wishes to run it, to whom the job should be charged, and at what priority it should be run. Furthermore, all the datasets involved must be explicitly defined so that the machine can find them (or create them) when the time comes. All this information is provided to the operating system with the aid of a specialized language called the Job Control Language, or JCL. Every task you want the computer to perform must be identified to it with the use of JCL. In its entirety, JCL is a complex and in many ways difficult language, but fortunately one need not know its intricacies in detail in order to be able to use it to perform the vast majority of straightforward jobs. This is particularly true because of the existence of "catalogued procedures", in which whole blocks of JCL which are used very often by programmers are saved as units. Such blocks of JCL may be then invoked in their entirety (much like PROCEDUREs in HAL/S) by being explicitly called by name. This facility allows most users interested in compiling or link-editing programs to do so with the submission of a minimum of JCL - generally one or two lines.

It is not the intention of this manual to teach the use of JCL. In fact, because of the existence of catalogued procedures, very little JCL needs to be known to compile, link edit, or execute a HAL/S program. The use of these procedures is explained in Section 2.5, where no acquaintance with JCL is presupposed. But the general introduction to JCL presented here may be helpful, especially for a user with some experience in employing the catalogued procedures who wishes to make some modifications in the catalogued JCL for his or her jobs. IBM offers a series of manuals which include a great deal more detail about the use of JCL, and which should be consulted if any major modifications are to be made.

2.4.1 The JOB Card

Every set of requests to the operating system must begin with a JOB card identifying the user, supplying accounting information, and informing the system of the storage requirements of the various steps to be performed. A typical JOB card might look like:

```
//TPS1840J JOB 7477,SLOTHROP.T,REGION=64K,TIME=(0,10),
//      PRTY=1,NOTIFY=TPS1840
```

Two slashes begin every card in JCL (except for the /* delimiter card to be discussed below). On the JOB card, a jobname should be specified immediately following the two slashes (with no intervening blank). Generally, this will be the user's id (here TPS1840) followed by some letter - but the name chosen is entirely arbitrary, as long as no other job executing on the system at the same time has the same name. After the jobname comes a blank, then the keyword JOB (identifying this as a JOB card), then another blank, and then a list of parameters. This structure:

```
//<name><operator><parameters>
```

holds for all JCL cards.

On the JOB card the first parameter should be whatever accounting information is

required by your installation - in this case an account number, followed by the user's name (7477, SLOTHROP.T). Next comes information relating to the allocation needs of the job to be run. Here it is specified that the maximum amount of main storage to be used by the job is 64K, and that the maximum amount of time to be used is 10 seconds (the format is (minutes, seconds)). The continued line in this example indicates that the job is to be executed at priority 1 (where 0 is the lowest priority), and that notification of the user TPS1840 should occur when the job has completed. Defaults exist for all of these allocation parameters, so they need not always be included.

Notice that among all these parameters no blanks are coded; none are permitted. There is none of the flexibility with respect to delimiters in JCL that there is in HAL/S. The trailing comma on the first line indicates that the line is to be continued. Similarly the leading blank, and lack of an operator such as JOB, EXEC, or DD, on the second line indicates that it is a continuation line. No statement in JCL, except comments, may continue past column 71. Comments may be coded on any line beginning with //*, or on any regular JCL line after the last operand and a blank is coded. Hence in the card:

```
//STEP1 EXEC PGM=PROG1 EXECUTE THE FIRST PROGRAM
```

the comment "EXECUTE THE FIRST PROGRAM" is understood as such. The result of these rules is that blanks are only permitted (and, in fact, are mandatory) at three places within any JCL card: immediately before the operator (JOB, EXEC, DD, etc.), immediately after the operator and before the parameter list, and immediately after the parameter list and before the comments, if there are any. Of course blanks are always permitted within comments.

2.4.2 The EXEC Card

Each job you submit to the operating system must be identified with a name on a JOB card prior to anything else. This specifies that requests for action by the system are now going to be made which are logically separate from those preceding the JOB card. But the JOB card does not inform the computer what it specifically is being asked to do. This is the function of the EXEC card, which indicates which program is actually to be run. Every job consists of one or more job steps; each of these steps requires an EXEC card to define it. A straight compilation of a HAL/S program consists of only one step, the running of a program called MONITOR which invokes the compiler and interfaces between it and the operating system. If, however, you wished to compile and link edit a program, two EXEC cards would be required - a call to the compiler (MONITOR) and a call to run the link editor (LINK101).

To do a compilation, for example, the EXEC card might be coded as follows:

```
//HAL EXEC PGM=MONITOR,REGION=350K,TIME=1,
//         PARM='LIST,SYMBOLS=500'
```

As with the JOB card, the keyword HAL immediately following the (compulsory) double slash is a name: this jobstep will be called HAL (although the job itself, as defined on the JOB card, may have a different name). The name is not required unless reference to this

step will be made in a later step - which is often the case in jobs consisting of more than one related step (e.g., compilation, link-editing, and execution). Each name given to a step in a job should be unique for that job.

After the name a blank is coded, followed by the operator EXEC to identify this as an EXEC card, and then another blank. Again, this structure is followed in all JCL cards.

The first parameter on an EXEC card must specify the task that is to be performed. This task will either be the running of a program or the calling of a catalogued procedure. A program consists simply of a block of executable code, ready for execution by the system; a catalogued procedure, on the other hand, is a block of JCL, which itself will include an EXEC card. The user's JCL, that is, can call up a catalogued block of JCL, which in turn calls some program to be run. It is this which allows HAL/S compilations, for example, to be run with a minimum of user-supplied JCL. Instead of invoking the actual compiler (MONITOR), the JCL can call a catalogued procedure (HALFC, for instance) where all the complicated JCL for a compilation has been stored, which in turn calls (on its EXEC card) MONITOR.

If a program is to be run, the parameter coded must be PGM=<name of program>; in the example, the program MONITOR is being called. If a catalogued procedure is to be called one may code either PROC=<name of procedure>, or, alternatively, the name of the procedure alone. That is

```
//STEP1 EXEC PROC=HALFC and
```

```
//STEP1 EXEC HALFC
```

are both legal, and are identical in effect: the catalogued procedure named HALFC is invoked.

The REGION and TIME parameters, like those used in the JOB card, specify the maximum amount of main storage and the time limit for the execution of the task invoked. But whereas the parameters on the JOB card give overall limits that cannot be exceeded by any step in the entire job (which may consist of several steps, and hence several EXEC cards), parameters on the EXEC card specify only limits on the particular step being defined. If you want to specify different region sizes for each step, no REGION parameter should be coded on the overall JOB card, because this causes the REGION parameter on any EXEC cards to be ignored. Here a storage size of 350K and a time limit of one minute are specified.

The programs and catalogued procedures invoked by a job step may, in many cases, permit or require arguments or parameters to be passed to them. These parameters may set initial values or specify options to be employed. For instance, the HAL/S compiler can have any of a large number of options specified to it, identifying the size of the various tables it will construct, describing the level of explicitness of the messages it emits, or instructing it to output dumps and traces. Parameters are passed to programs, as well as to catalogued procedures, by coding the PARM= parameter, with the right hand side of the equal sign consisting of the parameters to be passed, all contained within single quotes. In the example given, the HAL/S compiler is instructed to emit an

assembly list after Phase 2, and is told that space must be left for 500 symbols in the symbol table, rather than the default 200. (Section 5.1 specifies all of the available compiler options.)

In a job consisting of more than one step, it may occur that errors detected in an early step may make execution of a later step undesirable or even impossible. For example, if the job you are submitting consists of a three-step request to compile, link-edit, and execute a HAL/S program, and during compilation several serious errors in your source program are discovered, then you do not really want link-editing or execution to be attempted; any such attempt could only be expensive and fruitless. It is possible in JCL to make execution of a jobstep conditional upon successful execution of an earlier jobstep, with the criteria of "success" being user-defined. This is done using the COND parameter on the EXEC card, as follows.

Every program and procedure, when completed, produces a "condition code" which is simply an integer number indicating the outcome of the step. The COND parameter specifies an arithmetic comparison involving the condition code; if the comparison succeeds (is true) then execution of the step being defined is bypassed. The format of the COND parameter is COND=(n, comparison operator, jobstep), "n" may be any integer number; "comparison operator" is one of "GT" (greater than), "GE" (greater than or equal to), "EQ" (equal to), "NE" (not equal to), "LT" (less than), or "LE" (less than or equal to). "Jobstep", which is optional, indicates by name which previous EXEC card is being referred to. Thus, e.g., if COND = (4, NE, STEP1) is coded on an EXEC card, that step will only be executed if the condition code produced by the earlier step named STEP1 were not equal to 4. If the "jobstep" parameter is not coded, then the comparison is made with all condition codes produced by all earlier steps.

The HAL/S compiler produces condition codes of 0, 4, 8, 12, or 16, depending on the severity of any errors discovered in the source program. Error severities will be discussed in Section 3.3; suffice it to say, however, that if any code greater than 0 is produced, link editing and execution should almost certainly not take place. Therefore, JCL for the link-edit step, for instance, should always have the COND parameter coded. If the compilation step were named HAL, as in the previous example, the link-edit step might be coded as follows:

```
//LKED EXEC PGM=LINK101,REGION=100K,COND=(0,LT,HAL)
```

This would indicate that the LINK101 program should not be executed if 0 were less than the condition code produced by the step HAL (i.e., the compilation).

2.4.3 The DD Card

Each dataset required during the running of the program or procedure specified on an EXEC card must be defined for the system on a DD (data definition) card. The DD card is extremely important in JCL, and often complex in form. It may serve any of many different purposes. In order simply to execute a typical program, for example, the system needs the following information, all of which is specified on DD cards:

- it needs to be told where the actual program to be executed is to be found;
- if the program requires any input (e.g., for a HAL/S program, if it contains any READ or READALL statements) the system needs to know where to find it;
- if output from the program is to be placed in a dataset, printed on the line printer, etc., the system must be informed of this;
- if temporary datasets are needed as work areas for the running of the program, the system must be informed and must be told to delete these datasets at the end of each step.

Every DD card has the form:

```
//ddnames DD parameters.
```

The ddnames given may be user-defined, system-defined, or defined by the program being executed, depending on the request being made.

2.4.3.1 Defining Libraries of Programs: JOBLIB and STEPLIB

The two most common system-defined ddnames are JOBLIB and STEPLIB. When an EXEC card is encountered by the operating system, it searches for the program specified by the PGM= parameter in a system library (i.e., a partitioned dataset available to all users) called SYS1.LINKLIB. This library consists of a variety of system utilities - FORTRAN compilers, link-editors, sort and merge routines, etc. If the procedure called for is found there, well and good. But obviously if you wish to execute a program you have written yourself, some means is required for indicating to the system that it should look elsewhere, and for telling it where to look. Furthermore, in most installations the HAL/S utility routines - the compiler MONITOR, etc. - are found in a library separate from SYS1.LINKLIB, but available to all HAL/S users. If a catalogued procedure is being called, finally, the system must again be told where to find it.

This ability to define datasets other than SYS1.LINKLIB as a possible location for programs and catalogued procedures is provided by the JOBLIB and STEPLIB cards. A JOBLIB card defines a dataset (which must be of partitioned organization) which will be searched immediately before SYS1.LINKLIB for all programs and procedures called by EXEC cards within one job. It is coded immediately following the JOB card which defines the job. A STEPLIB card serves the same function, except that it remains in effect for only one step; accordingly, it is coded immediately following an EXEC card. The syntax for each is the same; only the JOBLIB card is shown.

```
//JOBLIB DD DSN=<name of user library>, DISP=<disposition>
```

DSN is an abbreviation for DSNAME, which can alternatively be coded as well. The function of the DISP parameter will be described below. If more than one user-defined library is to be searched, they should be concatenated together by submitting additional JCL cards immediately following the JOBLIB or STEPLIB card, using the same syntax but omitting the ddname JOBLIB or STEPLIB.

Thus, if the following JCL were submitted:

```
//TPS1840J JOB 7427,SLOTHROP.T,REGION=64K,TIME=(0,10),
//        PRTY=1,NOTIFY=TPS1840
//JOBLIB DD DSN=MYLIB.LOAD,DISP=(SHR,PASS)
//STEP1 EXEC PGM=PROG1
...other JCL associated with STEP1...
//STEP2 EXEC PGM=CHECKOUT
//STEPLIB DD DSN=CHECKS.LOAD,DISP=(SHR,PASS)
...other JCL associated with STEP2...
```

the system, when it reached the //STEP1 EXEC PGM=PROG1 card, would search first through MYLIB.LOAD and then, if unsuccessful, through the partitioned dataset SYS1.LINKLIB for a member named PROG1. When the //STEP2 EXEC PGM=CHECKOUT card is reached, the system would search through CHECKS.LOAD and then SYS1.LINKLIB.

2.4.3.2 The DISP Parameter: Specifying the Status of Datasets

The DISP parameter informs the system of the status (disposition) of the dataset referred to, both at the beginning and the end of the current jobstep. The first sub-parameter in parentheses describes the status of the dataset when the job starts. OLD indicates that the dataset already exists. NEW indicates that it does not yet exist, but rather is to be created. SHR indicates that the dataset in question is available to more than one user simultaneously. System libraries, as well as libraries such as the HAL/S runtime library which may be used by many programmers at once, typically have the disposition SHR. A dataset with this disposition should in general not be used for output; that is, it should be read from but not written into. Finally, MOD indicates that the dataset already exists, and will be modified during the course of the jobstep.

The second sub-parameter in parentheses within the DISP parameter tells the system what to do with the dataset being discussed once the jobstep is over. This may be DELETE, which causes the dataset to be deleted; CATLG, which causes it to be kept and placed in the user's catalogue; or PASS, which causes it to be kept at least until the execution of the next jobstep, where presumably it will be needed again. The PASS sub-parameter is very common, e.g., where one jobstep produces output which is then used as input for the next jobstep.

Examples of the DISP parameter will be provided throughout the following sections.

2.4.3.3 Defining Input Data

Every dataset involved in running a program may be thought of as possessing two names: a "real" or "system" name, defined when the dataset is created, and an "alias" by which it is known by the program. For example, compilation of a HAL/S program containing READ(5) statements makes an implicit reference to a "file number 5" (the compiler actually calls it "CHANNEL5"), from which input is to be read. But the user will in general have his or her input residing in a dataset which has already been named TESTDATA.DATA perhaps, and in fact may want to run the program several times, reading input from a different dataset each time. Therefore, DD cards must indicate to the operating system which actual datasets are to be associated with which "aliases", or internal filenames. Thus, if input to a HAL/S program is to be read from a dataset named TESTDATA.DATA, the following DD card should be coded:

```
//CHANNEL5 DD DSN=TESTDATA.DATA,DISP=(OLD,KEEP)
```

The ddname CHANNEL5 is the program name; the dataset specified after the DSN= parameter is the system name. DISP is indicated as (OLD,KEEP) to inform the system that the dataset TESTDATA.DATA already exists in the catalogue, and that it is to continue to exist after execution.

Every program which takes input has a series of specific program filenames with which input dataset names are to be associated. These differ from program to program, and must be determined by the user ahead of time. For a HAL/S programmer, the following names are important to know:

- the input to a HAL/S program's READ(5) statement is associated with the ddname CHANNEL5; and in general READ(n) statements are associated with CHANNELn, where n is an integer from 0 to 9;
- the primary input to the HAL/S compiler MONITOR (i.e., the program to be compiled) is associated with the ddname SYSIN.
- the primary input to the link-editor LINK101 (i.e., the object module to be linked) is associated with the ddname SYSLIN.

A complete list of ddnames for HAL/S users will be found in Section 2.5.3.

An alternate method of defining input data is available, allowing the data to be physically included within the stream of JCL itself. To do this, the DD card should be coded as //CHANNELn DD DATA, or alternatively, instead of //CHANNELn DD DATA, one may code //CHANNELn DD *. (If the input data itself contains JCL statements - or any lines beginning with // in the first two columns - the //CHANNELn DD DATA must be coded.) Immediately following this DD card should be the input, in whatever format is appropriate; and immediately following the last line of input should be the special JCL delimiter card, which consists simply of the two characters /* in the first two columns, with the rest blank. To execute a HAL/S program called GETPOS, for instance, residing in a partitioned dataset called PROGS.LOAD with the input to READ(5) statements being defined within the JCL, the following would be coded after the JOB card:

```
//RUNIT EXEC PGM=GETPOS,REGION=64K,TIME=(0,10)
//STEPLIB  DD  DSN=PROGS.LOAD,DISP=SHR
//CHANNEL5 DD *
  input
   .
   .
   .
/*
```

It is possible to specify more than one dataset to be associated as input with a particular program dataset name. Thus, for instance, one may wish to define two datasets to be used as input to READ(5) statements in a HAL/S program, or (as has already been mentioned) two program libraries to be searched for a program called on an EXEC card. This is done simply by coding the second, third, . . . , etc., datasets to be concatenated with the first one on DD cards immediately following the first one, without specifying any ddnames.

Thus, if three input datasets are to be used for a READ(5) statement, the following JCL would be coded:

```
//CHANNEL5 DD DSN=INPUT1.DATA,DISP=OLD
//         DD DSN=INPUT2.DATA,DISP=OLD
//         DD DSN=INPUT3.DATA,DISP=OLD
```

Notice that this means that, unless you wish to specify such a concatenation, a ddname must always be specified on a DD card.

2.4.3.4 Defining Output Datasets

The conventions for defining the datasets into which output is to be placed are similar to those for defining input datasets. Again, each program and procedure has associated with it a set of ddnames or aliases for the various outputs produced. The most important ones for HAL/S programmers are as follows:

- the output specified by a HAL/S program with a WRITE(6) statement is associated with the ddname CHANNEl6; and again in general WRITE(n) statements are associated with the ddname CHANNELn;
- the HAL/S compiler may produce several pieces of output, depending on the options specified at compile-time (see Section 5.1). The most important of these are a formatted source listing associated with the ddname SYSPRINT, and an object module associated with the ddname OUTPUT3 (Other outputs produced by the compiler will be described in Section 2.5.3).
- the link-editor LINK101 produces, among other things, an executable load module, associated with the ddnames SYSLMOD.

Often the dataset to which you wish output to be sent will not exist yet; the system must then be instructed to create it. The first subparameter of the DISP parameter in that case

must be coded as NEW, and the second subparameter as DELETE, CATLG, or PASS, depending on what is to be done with the dataset at the end of the step. Furthermore, information must be given to the system as to how much storage the dataset will require, where it is to be kept, its internal format, etc. This manual will not attempt to describe the various JCL parameters (e.g., VOLUME, SPACE, UNIT, RECFM, etc.) for the passage of such information; the appropriate IBM manuals should be consulted if you wish to create permanent datasets in this way. If a dataset already exists, however, no such information need be specified to use it for output; note, however, that anything in such a dataset before execution of the program will be erased in the course of producing output unless MOD has been specified as the first DISP subparameter.

Sometimes one wishes to employ datasets merely as temporary work areas. In that case it is not necessary to code any DSN parameter at all (although information about location, storage, etc., must still be provided). This will cause the dataset to be deleted at the end of the jobstep in which it is defined. Coding a DSN name with DISP= (NEW, DELETE) would have the same effect.

On many occasions, however, datasets are used to communicate data from one jobstep to the next, in which case one might want to use a temporary dataset which would be deleted at the end of the job, not the jobstep. In this case, one would code a special dataset name indicating to the system that the dataset is a temporary one by beginning the name with a double ampersand (&&). At the end of the job, the system automatically deletes all datasets whose names begin with those two characters. In each jobstep but the last, the second subparameter of the DISP parameter should be coded as PASS to indicate that the dataset is to be made available to future jobsteps.

Although often one wishes to keep the output from a program in a safe, permanent, and machine readable form by placing it in a dataset, sometimes one only wants to look at it once, and perhaps to keep a hard copy of it. This can be done by employing the SYSOUT parameter on the DD card for such output. In most implementations, SYSOUT=A causes the output data to be printed on a line printer. If SYSOUT is coded, of course, the DSN parameter should not be coded. To complete the example begun in the previous section, assuming that you wanted output from the WRITE(6) statements in the GETPOS program to be sent to the line printer, you would code:

```
//RUNIT EXEC PGM=GETPOS,REGION=64K,TIME=(0,10)
//STEPLIB DD DSN=PROGS.LOAD
//CHANNEL5 DD *
   input
// *
//CHANNEL6 DD SYSOUT=A
```

Once a ddname has been associated with a dataset on a DD card, that ddname, rather than the dataset name, may be used later in a JCL sequence to refer to the dataset. This is particularly useful in a job consisting of more than one step, where the output from one step serves in turn as input for a later one. Since ddnames need only be unique within a jobstep, not a job, naming conventions are used to direct the system to the correct card defining the dataset. If in jobstep STEP1 the ddname OUTPT is associated with a

dataset, RESULTS.DATA, which is then to be used in jobstep STEP2 as input to a HAL/S program with READ(5) statements, the JCL would be as follows:

```
//STEP1    EXEC PGM=FIRSTPROG
//OUTPT    DD DSN=RESULTS.DATA,DISP=(OLD,PASS)
//STEP2    EXEC PGM=SECONDPR
//CHANNEL5 DD DSN=*.STEP1.OUTPT
```

As this example shows, the syntax for DSN parameter references to datasets defined in an earlier jobstep is "*.stepname.ddname". A reference to a dataset defined in the same jobstep is simply coded as "*.ddname" (Note that in the example the second subparameter of the DISP parameter had to be coded as PASS in order to make the dataset available to the system in the next step).

On occasion, one may wish to refer to a dataset which has not yet been defined, but which will be defined later on in the flow of JCL cards (This often occurs when catalogued procedures are being used: see below). In this case, one may code, instead of DSN, a DDNAME parameter, which associates with the ddnames specified other ddnames to be defined later. For example, if one were to code

```
//CHANNEL5 DD DDNAME=SYSINP
```

this would mean that the input dataset for channel 5 will be defined later on in the JCL sequence, on a DD card with ddname SYSINP - for instance,

```
//SYSINP DD DSN=TESTDATA.DATA,DISP=(OLD,PASS)
```

2.4.4 Catalogued Procedures

Catalogued procedures are blocks of JCL which are stored in a library and may be called up by any user. This facility makes it possible to save commonly used sequences of JCL so that the user need not repeat them on each occasion. Procedures may be called as they are, or the call may modify them and set or change values of various parameters specified within them. Such modifications are only effective for that single call; thus catalogued procedures may allow for a great deal of flexibility in their use.

2.4.4.1 Writing Catalogued Procedures

The syntax for the JCL within a catalogued procedure is precisely the same as the JCL syntax for regular jobs. A procedure may consist of more than one step, and so more than one EXEC card may be coded. There are two exceptions, however: the JOB card is not permitted in a catalogued procedure, and a catalogued procedure may not call another catalogued procedure on an EXEC card - only programs may be called.

A sequence of characters within a statement in a catalogued procedure may be defined to be a symbolic parameter of that procedure by preceding it with a single ampersand (&). The sequence must not consist of more than seven characters, and the first character must be alphabetic (or #, @, or $).

Suppose, for example, that a catalogued procedure RUNTEST executes a program called RUNNER, but that the maximum region size and the maximum execution time are to be specified by the user when the procedure is called. The EXEC card for this step might then be coded as follows:

```
//STEP EXEC PGM=RUNNER,REGION=&MAXREG,TIME=&MAXTIME
```

When this statement is executed, the system will substitute for &MAXREG and &MAXTIME the values which have been specified for them when the procedure was called (See Section 2.4.4.2 for information on how these values may be specified).

The minimal use of blanks in JCL means that under certain conditions it might be ambiguous to the system where the sequence of characters defining a symbolic argument (i.e., beginning with an ampersand) actually ends and where the next parameter begins. In cases where such an ambiguity would otherwise arise, the end of an argument name must be indicated by a period. Thus, for instance, if a procedure is to execute a program called TESTPROG, but the library of load modules where it resides is to be user-specified, the STEPLIB card defining the library might be coded

```
//STEPLIB DD DSN=&LIBNAME..LOAD,DISP=(OLD,PASS)
```

where the first period indicates the end of the symbolic argument LIBNAME, allowing the second one to be correctly interpreted as separating the two parts of a qualified dataset name.

Often in a catalogued procedure, one may wish to set default values for the various arguments. This can be done by coding a PROC card as the first card in the procedure. The syntax for this card is:

```
//<procedure-name> PROC <default-values>
```

If one wanted to set default values for the RUNTEST procedure mentioned above, for example, the following PROC card might head the procedure:

```
//RUNTEST PROC MAXREG=64K,MAXTIME=(0,10)
```

Coding this card would mean that the MAXREG argument would be set at 64K and the MAXTIME argument at (0, 10) unless a user specified them differently when he or she called the procedure. Note that here the leading ampersand and trailing period are not coded.

It is also possible to indicate in a PROC statement that certain parameters are not to be coded at all unless the user so specifies. For example, one may wish to send parameters to a program only under certain conditions.

Something like these two JCL cards would then be submitted:

```
//ABLE PROC RUNPARMS=
//GO EXEC PGM=BAKER,REGION=64K,TIME=(1,0),PARM=&RUNPARMS
```

The effect of these is to execute the program BAKER with the parameters to be passed to the program being specified by the user when the procedure is called. Rather than setting a default, however, the PROC card here indicates that, if no parameters are specified by the user, the PARM parameter on the EXEC card is to be ignored (i.e., no special parameters are to be passed).

2.4.4.2 Using Catalogued Procedures

Once a catalogued procedure has been written and stored in a procedure library, the user need not worry about its details, although it will still be necessary to know what arguments the procedure expects and what defaults it has set up. This information should be available in an easily accessible form at your installation.

To call a catalogued procedure and cause it to be executed, you need only specify its name on an EXEC card within your own JCL. The procedure name should be followed by a series of parameters specifying the values you wish to assign to the various symbolic arguments in the procedure. To call the RUNTEST procedure described in the previous section, for example, the following line of JCL might be coded:

```
//GO EXEC RUNTEST,MAXREG=16K,MAXTIME='(2,0)'
```

The system would then automatically substitute these values for the symbolic arguments &MAXREG and &MAXTIME wherever it finds them. MAXTIME's value is coded in apostrophes because it contains 'special characters'. These are the comma, period, slash (/), apostrophe, left and right parentheses, asterisk, ampersand, plus sign, hyphen, equals sign, and blank; any argument value containing them must be enclosed within apostrophes to prevent ambiguities.

Notice that values must be specified on the EXEC card invoking the procedure for each symbolic argument found in the procedure, unless a default value has been set in a PROC statement beginning the procedure. Some values may be specified as null, in the same way that this is done on the PROC card. If the RUNTEST procedure were called as follows, for example,

```
//GO EXEC RUNTEST,MAXREG=,MAXTIME='(2,0)'
```

a null value would be transmitted to the procedure as the MAXREG parameter. The result would be to have the maximum region used for your job set to whatever the default value is at your installation.

If a catalogued procedure can stand alone as a fully correct sequence of JCL, it is possible to execute it using only two lines of user-specified JCL - a job card and an EXEC card. To call such a procedure named INFO, for instance, user TPS1840 might code only the following:

```
//TPS1840I JOB 7477,SLOTHROP.T
//GO EXEC INFO
```

(This assumes that the system default values for job time, region size, priority, etc., are

desired, and that no values for symbolic arguments are to be passed to the procedure). When the EXEC card is reached, the system simply executes the procedure, and that's that. On the other hand, many catalogued procedures are not able to stand alone in this fashion, and require additional user-supplied JCL to allow them to run. For example, the procedure HALFC which runs HAL/S compilations calls the compiler MONITOR, which expects to find a DD card named SYSIN indicating the location of the HAL/S program to be compiled. Since HALFC presumably will be used to compile many different programs, the SYSIN DD card is not included within the procedure. It could be included by using a symbolic argument to stand for the dataset name which the user would have to supply when calling the procedure; but in general in this situation it is easier to leave the card out entirely and allow it to be added by the user in his or her own JCL.

Care must be taken in specifying the ddnames on such a user-supplied addition to a procedure's JCL. The DD card should be coded immediately following the EXEC card which calls the procedure, and the ddname given on the card should be <procstepname>.<procddname> where <procstepname> is the name specified on the EXEC card in the procedure to which this DD card refers, and <procddname> is the ddname the procedure expects. Thus, to specify the program to be compiled in the HALFC procedure, assuming that the EXEC card in the procedure is called HAL, a user might submit a job as follows:

```
//TPS1840C  JOB 7477,SLOTHROP.T,REGION=64K,TIME=(0,10)
//COMPILE   EXEC HALFC
//HAL.SYSIN DD DSN=<location of user's program>
```

More than one DD card may be added by a user to a catalogued procedure in this fashion. For instance, a procedure named HALFCLG can be used in some installations, to compile, link edit, and then execute a user's program. The procedure consists of three steps (i.e., contains three EXEC cards): HAL (which calls the compiler), LKED (which calls the link editor), and GO (which runs the program). The compile step expects a SYSIN card indicating the program to be compiled, as above, while the run step expects a CHANNEL5 card indicating where the input to any READ(5) statements in the program is to be found.

The following JCL would then be submitted to call HALFCLG (after an opening JOB card):

```
//COMPRUN     EXEC HALFCLG
//HAL.SYSIN   DD DSN=<name of the source program dataset>
//GO.CHANNEL5 DD DSN=<name of input dataset>
```

(In fact, the way the HALFCLG program is written, GO.SYSIN could be coded as well as GO.CHANNEL5. This is because the DDNAME parameter is used to associate the CHANNEL5 ddname with the SYSIN ddname: the card used in the procedure is //CHANNEL5 DD DDNAME=SYSIN. This allows the user to employ the more familiar SYSIN ddname. The double use of SYSIN is unambiguous because they are only

referred to in the user's call with prefixes HAL or GO).

2.4.4.3 Modifying Catalogued Procedures

In addition to being able to add statements to incomplete catalogued procedures, you may also override statements within them as much as you wish. The naming conventions here are similar. To override a DD card in a catalogued procedure, you should code a DD card in your own JCL with ddname <procstepname>.<procddname>, where <procddname> is the ddname of the DD card you want to override, and procstepname is the name of the step in the catalogued procedure in which it occurs. Only those parameters you wish to change need be specified on the new DD card. When the system encounters a line in the catalogued JCL for which you have coded a modification in this way, it automatically makes the modification, ignoring any parameters on the catalogued card which you wish to change and reading them from your card instead.

The HALFCLG procedure, for instance, puts the load module from the linkage editor into a temporary dataset for passage to the GO step. Since this dataset, called &&HALMOD(GO) in the procedure, is defined as temporary by the two ampersands beginning its name, after execution of the procedure it no longer exists. This may not be a good idea, if you are planning after a single computation to run the program several times without recompiling it. The line in HALFCLG defining this dataset appears within the step named LKED, and runs as follows:

```
//SYSLMOD DD DSN=&&LOADMOD(GO),DISP=(,PASS),
```

followed by storage information.

(Note that the DISP=(,PASS) parameter really abbreviates DISP=(NEW, PASS), taking advantage of the fact that NEW is the default for the first subparameter of the DISP parameter.) In order to override this line so as to create a permanent dataset for later use, you would code, after the EXEC HALFCLG statement,

```
//LKED.SYSLMOD DD DSN=HALMOD(GO),DISP=(NEW,CATLG),
```

and any storage information you wish to change.

The missing ampersands, and the new CATLG disposition, inform the system that this new partitioned dataset named HALMOD is to be permanent. Again, the LKED.SYSLMOD ddname is crucial here, for it indicates to the system which card in the procedure you wish to override (in this case, the SYSLMOD DD card in the LKED step). Note that only the parameters you wish to change need be specified when you are overriding a line. It should be mentioned here that all modifying JCL statements must come before any additional JCL, and must preserve the order in which the ddnames occur in the catalogued procedure.

2.5 Using Catalogued Procedures to Process a HAL/S Program

2.5.1 Introduction

Catalogued procedures may be used to minimize the amount of JCL you need to submit in order to perform a number of possible tasks with respect to a HAL/S program. These tasks include compilation and link-editing. The rest of this section will assume that a regular complement of these catalogued procedures is available at your installation. Prototype versions of these procedures are listed in Appendix A. Table 2-1 gives the names of the procedures and their functions. The naming conventions for these procedures should be clear: each name begins with HALF (for HAL/S-FC) and has the letter C, or L, appended to it to refer to compilation or link-editing, respectively.

HALFC	Compiles a HAL/S program.
HALFCL	Compiles and link-edits a HAL/S program.

Table 2-1 Catalogued Procedures Available for Processing HAL/S Programs

2.5.2 How to Use the Catalogued Procedures

Only the simplest and most direct uses of the catalogued procedures will be described here. More complex situations, e.g., if more than one compilation unit is involved, will be discussed below in Chapter 3.

Any job, including one involving a catalogued procedure, must begin with a JOB card. This must be user-supplied, and must provide the usual information needed by your installation. A typical JOB card might be:

```
//TPS1840 JOB 7477,SLOTHROP.T,REGION=64K,TIME=(1,30),
//            PRTY=0,NOTIFY=TPS1840
```

But the requirements and even syntax of the JOB card may differ from installation to installation.

The procedure itself is called on an EXEC card which should have the following form:

```
//ANYNAME EXEC <name of procedure>
```

ANYNAME may in fact be replaced by any name whatsoever, and may even be omitted, as long as at least one blank is placed between the '//' and the EXEC operator. <name of procedure> is simply the name of the procedure being called.

It is possible (but not necessary) to pass a series of options to the compiler. A discussion and complete list of compiler options will be found in Section 5.1. These may be passed by coding on the EXEC card, immediately after the name of the procedure, a comma and then OPTION='<compiler options>'. The list of options must be enclosed in apostrophes, and each one must be separated from the next by a comma. If no options are coded, defaults are specified in MONITOR's option processor which are sufficient for most users.

The following JCL card, for example, calls for the compiling and link editing of a program

with the LISTING2 compiler option on and the SYMBOLS option set to 300.

```
//COMPLINK EXEC HALFCL,OPTION='LISTING2,SYMBOLS=300'
```

2.5.2.1 To Compile

Simply to run a compilation, the HALFC procedure should be called. The EXEC card will be:

```
//COMPILE EXEC HALFC
```

with any desired compiler options specified with OPTION=. This should be followed by a DD card describing to the system the location of the source program you wish compiled. The ddname associated with this by the cataloged procedures is HAL.SYSIN. The source program may be included within the user's JCL itself, in which case the format would be:

```
//HAL.SYSIN DD *
     .
     .
     .
     source program
     .
     .
     .
/*
```

Or else the program may reside in some catalogued dataset, in which case the format would be:

```
//HAL.SYSIN DD DSN=<dataset name>,DISP=(OLD,KEEP)
```

The primary output of the HALFC procedure is an object module, placed in a temporary dataset named &&HALOBJ. Note that since this dataset is defined by the procedure as temporary, it will be deleted at the end of your job. If you wish to store it for later use, modification of the JCL procedure will be required (see Section 2.4.4.3).

2.5.2.2 To Compile and Link Edit

To compile and link edit a HAL/S program in order to produce an executable load module, the HALFCL procedure should be called. The EXEC card will be:

```
//COMPLINK EXEC HALFCL
```

(with any desired compiler options specified with OPTION=).

This should be followed by a DD card describing to the system the location of the source program you wish compiled and linked. The format of this card is precisely the same as the HAL.SYSIN DD card described above in Section 2.5.2.1.

The primary output from the HALFCL procedure is an executable load module, placed in a temporary partitioned dataset named &&LOADMOD in a member called GO. Note that since this dataset is defined by the procedure as temporary, it will be deleted at the end of your job. If you wish to save it for further use, an additional DD card modifying the procedure will be required (see Section 2.4.4.3).

2.5.3 Standard DDnames and Outputs Associated with the Catalogued Procedures

Complete listings of the various catalogued procedures available may be found in Appendix A, along with a detailed description of their workings. The following table lists the standard ddnames and outputs associated with each of the processes called by the procedures. Users unfamiliar with the HAL/S compiler and link editing systems should ignore (or at least postpone reading) this section.

Compilation (Step HAL of Procedures HALFC and HALFCL)

Program: MONITOR
User Options: OPTION

DD names

STEPLIB	Defines a load module library where MONITOR is found.
SYSIN	The HAL/S program to be compiled.
PROGRAM	A dataset defining the compiler to be used.
SYSPRINT	A formatted listing of the source program, which is directed to the line printer.
LISTING2	An unformatted listing of the source program (optional), directed to the line printer.
OUTPUT3	The object module produced by the compilation, which is kept in a temporary dataset named &&HALOBJ for passage to the link editor.
OUTPUT4	A second copy of the object module directed to the card punch (optional).

OUTPUT5	The Simulation Data File produced by Phase 3, which is kept in a temporary dataset called &&HALSDF for passage to later compile steps.
OUTPUT6	The template library produced by the compiler, which is kept in a temporary dataset named &&TEMPLIB for passage to later compile steps.
ERROR	The error library required by the compiler.
FILE1-FILE6	Temporary work files.

Link Editing (Step LKED of Procedure HALFCL)

Program: LINK101
User Options: none

DD names

SYSLIN	The object module to be link edited.
SYSPRINT	The diagnostic output dataset, directed to the line printer.
SYSLIB	The HAL/S runtime library of built-in functions and procedures.
SYSLMOD	The load module produced by the link editor, which is kept as a member of a temporary dataset called &&LOADMOD for passage to an execution step.
SYSUT1	Intermediate work dataset for temporary storage purposes.

3.0 PROCESSING PROGRAM COMPLEXES

3.1 Program Complexes

In the HAL/S system, a compilation unit is an independently compiled sequence of HAL/S statements. Such units may be PROGRAMs, PROCEDUREs, FUNCTIONs, or COMPOOLS. PROGRAMs, of course, are always separately compiled. Procedures and functions may be nested within programs, in which case they are compiled along with their parent program, but they may also be independently compiled in order to allow them to be shared by several programs. This would be useful, for instance, if many programs you write will employ a particular procedure, say a simple text editor named EDIT. It would be highly inefficient to include the source code for EDIT in each program you write prior to compilation. A much better solution would be to compile EDIT once, independently, producing an object module which may then be linked to the object code produced by each later compilation of a new program. Procedures and functions which are compiled separately in this manner are generically called COMSUBs.

A COMPOOL is a block of data which may be shared by several programs. COMPOOLs contain no executable code; they simply define data. Thus, for instance, you may wish to run one program to compute some velocity and position vectors, and then run a separate program which takes those vectors and issues commands to a navigation system. The variables storing the velocity and position vectors would then be placed in a COMPOOL in order that both programs will be able to access them. The user should consult the Language Specification, Chapter 3, or the Programmer's Guide, Chapter 15, for more detailed information about the form and use of COMSUBs and COMPOOLs.

A program complex is a collection of compilation units. Typically, the individual units in a program complex will reference each other (e.g., many units may reference data in a COMPOOL unit, or some units may CALL or SCHEDULE other units). Such references are identical to those employed when the units referenced are not independently compiled. Thus, if a program is to include a call to the independently compiled procedure EDIT, a CALL EDIT statement is coded (although the procedure itself is not); similarly, variables defined in a COMPOOL may be employed by name in a unit referencing that COMPOOL as if they had been DECLAREd in the unit itself.

3.2 Templates: The INCLUDE Directive

Compilation of each of the units in a program complex occurs separately. When a unit is compiled which includes references to other units in the complex, information must be given to the compiler about these other units. First of all, it must be informed that another unit is in fact being referred to, and that the programmer has not simply forgotten to declare a variable or to include the code for a procedure or function. This is necessary in order to enforce the strict HAL/S rules as to the predictability and stability of data types, arrayness, etc., particularly when function invocations and procedure calls are made. Furthermore, the compiler must be given information about the data type of all variables which are referenced in COMPOOLs, and about the number and data type of all variables passed into COMSUBs as arguments or returned from them as assign parameters or function values.

Such information is provided to the compiler by a compilation unit TEMPLATE. Every compilation unit which makes references to other independently compiled units must include such a template describing each other unit referenced. The format for such TEMPLATEs is described in detail in the Language Specification, Section 3.6, and the Programmer's Guide, Section 15.4, but, in fact, they need never be manually coded, for the HAL/S-FC compiler automatically produces a TEMPLATE describing each unit it compiles. Templates so produced are placed in a file which may then be used in subsequent compilations. Thus, the necessary information about interfacing with COMSUBs and COMPOOLs is provided (by the user) simply by directing the compiler to retrieve the proper template from these files. In fact, another (and in many cases more efficient) method exists for passing such information to the compiler; see Section 3.5 for details.

Retrieval of the template for a previously compiled unit takes place through the use of the INCLUDE compiler directive. Compiler directives are instructions, coded as part of a HAL/S program, which make requests directly to the compiler; they have no function in execution. A line of HAL/S code is specified to be a compiler directive by placing a D in the first column of the source text. A complete list of compiler directives will be found in Section 5.2; here we will be concerned only with one form of the INCLUDE directive. In order to instruct the compiler to include a template for a COMPOOL named DATALIST, for instance, the following line would be coded prior to the first line of the program being compiled:

```
D INCLUDE TEMPLATE DATALIST
```

Remember that in this case column 1 must not be blank, but must have the D coded in it. When it comes across this directive, the compiler begins a search through certain libraries (see Sections 3.3.1 and 3.3.2 for details) to find the template for the compilation unit named DATALIST which it then reads as if it were part of the source code.

Templates, when produced by the compiler, are given characteristic names which are derived from, but not identical with, the user-supplied block name. The first six characters of the block name, with underscores eliminated, are prefixed with the characters @@ to form the template name. Thus, compilation of a COMPOOL named TEST_DATA would result in a template named @@TESTDA. The name specified on an INCLUDE TEMPLATE statement, however, should be the real block name; the compiler automatically computes the template name to be searched for. The template for TEST_DATA would thus be included by coding simply INCLUDE TEMPLATE TEST_DATA. If for some reason one wishes to refer to a template by its literal name, "INCLUDE literal name" (i.e., without the keyword TEMPLATE), should be coded. Thus, e.g.,

```
D INCLUDE @@TESTDA
```

would have the same effect as:

```
D INCLUDE TEMPLATE TEST_DATA.
```

More information about template generation will be found in Section 3.4. The INCLUDE directive has other uses as well, and may in general be used to merge any part of a source program with another one; see Section 5.2 for details.

3.3 Processing a Program Complex

This section describes the concepts as well as the JCL needed to process a program complex, with special attention to the compilation step. The simplest kind of processing is that which takes place when each unit in the complex is processed within a single job; but the units may also (more typically) be compiled at different times, with the resulting templates and object modules being saved, and with final linking postponed until all compilations are complete. The simple type is described in Section 3.3.1, the more complex type in Section 3.3.2, where the template look up conventions which are employed are also explained.

3.3.1 When All Compilations Are Performed in One Job

Compilation of a program complex, whether this occurs within one job or not, actually consists of a series of compilations, since each unit in the compilation must be compiled separately. Every such compilation results in the generation of a template for that unit. Since compilations of units which reference other units must include templates for each external unit referenced, the order of compilation is important; COMPOOLs and COMSUBs should always be compiled before the compilation of any unit referencing them.

When compilation of an entire complex takes place within one job, the JCL for each compilation should be such that the object module produced is saved until all the units are compiled. Furthermore, all templates produced should be directed via JCL to a common template library so that compilations of later units in the process can reference the necessary template. The catalogued procedure HALFC is written in such a way that these things take place automatically. Once compilation of each unit in the complex has been completed, link editing can take place in the normal way.

In addition to templates, each compilation also produces a Simulation Data File of symbolic data, which is required for functional simulation of the AP-101 on the 360. These SDFs may themselves be used instead of templates to pass information to the compiler about external references (see Section 3.5). Again, JCL should be provided in order to save the individual SDFs produced by each compilation until the entire complex has been processed. The HALFC procedure does this automatically as well.

To illustrate the procedure for creating an executable program complex in one job, consider the following example. There are four independently compilable units in the complex as follows:

Unit 1 - A COMPOOL named DATA defined as:

```
DATA: COMPOOL;
      DECLARE I INTEGER INITIAL(4);
      DECLARE S SCALAR INITIAL(3.2);
CLOSE DATA;
```

Unit 2 - A PROCEDURE named PROC1 defined as:

```
D INCLUDE TEMPLATE DATA
    PROC1: PROCEDURE(X);
       DECLARE X SCALAR;
       WRITE(6) 'THE ANSWER IS:',(X+S);
       RETURN;
    CLOSE PROC1;
```

Unit 3 - A PROGRAM named PROG1 defined as:

```
D INCLUDE TEMPLATE DATA
D INCLUDE TEMPLATE PROC1
     PROG1: PROGRAM;
        CALL PROC1(S);
     CLOSE PROG1;
```

Unit 4 - A PROGRAM named DRIVER defined as:

```
D INCLUDE TEMPLATE DATA
D INCLUDE TEMPLATE PROC1
D INCLUDE TEMPLATE PROG1
    DRIVER: PROGRAM;
      CALL PROC1(S);
      SCHEDULE PROG1;
    CLOSE DRIVER;
```

Since each of the units in this complex reference all of the units listed above it, it is necessary to compile the units in precisely the order given here so that template information produced in a given step is available when needed by subsequent compilations. The following JCL would be coded in order to compile and link-edit this complex:

```
//TRYIT JOB <parameters>
//STEP1 EXEC HALFC
//HAL.SYSIN DD *
```

```
    <unit 1 source>
/*
//STEP2 EXEC HALFC
//HAL.SYSIN DD *
    <unit 2 source>
/*
//STEP3 EXEC HALFC
//HAL.SYSIN DD *
    <unit 3 source>
/*
//STEP4 EXEC HALFCL
//HAL.SYSIN DD *
    <unit 4 source>
/*
```

STEP1 above compiles the COMPOOL named DATA. Because of the way the HALFC catalogued procedure is written, the object module produced by the compilation is placed in a temporary file named &&HALOBJ which is automatically passed on to subsequent steps in the job. Recall that temporary datasets are automatically deleted only at the end of a job, not a jobstep. A template is also produced by the compilation, and this too is placed in a temporary file (named &&TEMPLIB), which is passed to later jobsteps, as is the simulation data file produced by Phase 3 of the compiler (placed in a file named &&HALSDF).

STEP2 compiles the PROCEDURE named PROC1. Note that the source code for this step includes a "D INCLUDE TEMPLATE DATA" compiler directive, which requests the compiler to locate and read the template for the COMPOOL named DATA.

The automatic production of such a template in STEP1, and its retention in the dataset &&TEMPLIB, makes this possible for the compiler (See Section 3.3.2 for more information on template lookup). The object code produced by the compilation of PROC1 is now added to the &&HALOBJ file, and similarly the new template and SDF produced are added to the existing template and SDF files. All these files once again are available to later steps.

STEP3 takes place in a manner similar to STEP2, this time compiling the PROGRAM named PROG1. This compilation requires access to both of the previously compiled units. The source code to be compiled, therefore, includes two INCLUDE directives to cause the compiler to retrieve the necessary information. Again, object code, templates, and SDFs are produced, added to the existing files, and passed along.

STEP4 invokes the HALFCL procedure because this step will perform the last of the compilations, to be followed by a link edit step. The source code for the fourth unit once again contains all of the INCLUDE directives needed to retrieve the templates for the previous compilations which will be referenced by DRIVER. The HALFCL procedure, in its compilation step, adds the object code for DRIVER to the &&HALOBJ file, produces an SDF, and also produces a template for DRIVER. Production of the template in this case is unimportant because it will be required by no subsequent steps. HALFCL then proceeds to link-edit the object file together with the necessary library routines to form a load module suitable for execution.

The SDF files created and passed by each step are available in this last step if simulation, diagnostic information, debugging, etc., are desired. Employing the HALFCLG procedure will allow simulated execution of the complex to take place under the control of the HAL/S User Control Program (see Chapter 2.4.4.3).

After linking or simulation, all temporary datasets used to collect object modules, templates, and SDFs would be deleted by the operating system. If any of these are to be saved, JCL informing the system of this should be coded. The load module produced would also be deleted, so that JCL modifications should be made if future execution of the compiled complex will be desired (see Section 2.4.4.3).

3.3.2 When Compilations Are Done At Different Times

A complicated program complex, particularly one still growing and developing, and possibly involving different programmers working on different units, will typically not be compiled and linked in a single job as described in the previous section. More likely, compilation of the units involved will take place at different times, and final linking of the entire module will be postponed until all compilations are complete. Conceptually, the tasks involved in this process are very similar to those outlined above for the simpler case. In particular, careful attention must be paid to the order in which units are compiled, in order to ensure that the templates to be INCLUDEd by a particular unit are available when needed.

But in this case an additional difficulty arises. In the previous section, all templates produced were placed into a specific temporary dataset &&TEMPLIB, and it was here that the compiler searched for each template referred to by an INCLUDE statement. In this case, however, the template produced by one compilation must be saved in a permanent dataset in order that it may be referenced by other compilations which may run weeks, months, or even years later. Furthermore, during these later compilations, the compiler must be informed where such templates are to be found; it can no longer simply assume that they will be found in the &&TEMPLIB dataset.

In order to save templates produced in one compilation so that they may be available to the compiler at a later date, the OUTPUT6 card of the compile step in the various catalogued procedures should be modified. It is this card which defines to the compiler where the templates it generates should be placed. The dataset on which a template is to be saved must have partitioned organization. In general, the best idea is to save all templates for a single complex as members of the same dataset. If templates are to be

saved in a catalogued PDS named TEMPLS, for instance, the following line should be coded in the JCL calling the catalogued procedure:

```
//HAL.OUTPUT6 DD DSN=TEMPLS,DISP=(OLD,PASS),
//      <other parameters if needed>
```

Note that the member name of the TEMPLS partitioned dataset should not be specified; it will be computed by the compiler. Consult Section 3.5 for more details about template generation.

When a unit is to be compiled which contains INCLUDE directives and therefore requires access to templates produced by previous compilations, an INCLUDE DD card may be coded in order to inform the compiler where these templates are to be found. Compilation of a unit referencing a template in the library RESIDE, for example, might require the following lines in the JCL:

```
//HAL.INCLUDE DD DSN=RESIDE,DISP=(OLD,PASS),
//    <other parameters if needed>
```

The dataset referred to in an INCLUDE DD card must have partitioned organization. Again, it should be referred to in the JCL by dataset name only; member names should never be specified. When the compiler comes across an INCLUDE statement in the source text (e.g., D INCLUDE TEMPLATE TESTDATA or D INCLUDE @@COMDAT), it searches the PDS defined on the INCLUDE DD card for a member with name found or implied on the INCLUDE statement (e.g., in these cases, @@TESTDA or @@COMDAT). If more than one template library is to be searched, the different libraries may be concatenated by following the normal rules for such concatenation.

```
//HAL.INCLUDE
//    DD DSN=<templib1>
//    DD DSN=<templib2>
            .
            .
            .
//    DD DSN=<templibn>
```

Note that if an INCLUDE request cannot be satisfied by referencing the INCLUDE DD card, or if no such card is coded, an attempt will be made to find the member needed by searching the dataset defined on the OUTPUT6 DD card instead. This search technique allows multiple-compilation jobs to reference template libraries created automatically in earlier steps without the need to code an INCLUDE DD card. This facility was employed, for instance, in the example provided in the previous section.

Furthermore, it follows that even in most of the cases where compilation of the units in a complex takes place at different times the template library for the complex need only be specified on the OUTPUT6 card for each compilation. No INCLUDE DD card generally needs to be coded, since even in its absence the OUTPUT6 dataset is checked for templates to be INCLUDEd. The OUTPUT6 DD card, however, defines a output dataset,

and so only one dataset may be specified on it; concatenation is impossible. Thus, if for some reason more than one template library has to be referenced for a particular compilation, an INCLUDE DD card must be used.

3.4 Template Generation

When compilation of a unit takes place, a template for that unit is automatically produced, and placed as a new member into the partitioned dataset defined by the OUTPUT6 DD card. We assume from here on that this partitioned dataset is a template library whose members consist of templates for all of the units in the program complex. In that case, it may occur - since units are often compiled, revised, and recompiled several times - that a template for the unit being compiled already exists in the template library.

When compilation takes place, therefore, the compiler ascertains first of all whether a template for the block being compiled already exists. It searches for such a template first on the dataset defined by the INCLUDE DD card, and then on the one defined by the OUTPUT6 DD card, just as it does when it encounters a D INCLUDE directive (see Section 3.3.2). Our assumption again is that the template library is defined on the OUTPUT6 DD card, and therefore that in fact no INCLUDE DD card is coded.

If no such template exists, then it is added as a new member of the OUTPUT6 dataset. The following message appears in the output listing:

```
*****TEMPLATE LIBRARY MEMBER <membername> NOT FOUND-ADDED
```

If the template does already exist, and the new template is different from the old, the old one is replaced in the template library by the new template. The compiler emits the message:

```
*****TEMPLATE LIBRARY MEMBER <membername> FOUND AND CHANGED
```

If the new template is the same as the old one, then no replacement occurs and the compiler emits the message:

```
*****TEMPLATE LIBRARY MEMBER <membername> FOUND-CHANGE NOT REQUIRED
```

Automatic updating of members in a template library is therefore ensured. Any error discovered in a unit during Phase 1 of the compilation will inhibit the generation of a template for that unit, and so no adding or replacement in the template library will occur. The following message will be emitted by the compiler:

```
*****COMPILATION ERRORS INHIBITED TEMPLATE GENERATION.
```

Each compiler-produced template has attached to it a version number between 1 and 255. When the template is first generated the version number assigned is 1. Every time it is replaced in the template library by an updated template the version number is incremented by 1. If the version number exceeds 255, it is reset to 1, and thereafter begins being incremented again.

The <membername> under which a template is stored in a template library is the computed template name, as described in Section 3.2 (e.g., the template for a block

named FUNC_ONE would be stored in a template library member named @@FUNCON). The format of templates produced by the compiler is exactly as described in the Language Specification, Section 3.6. It is followed by a statement (actually a compiler directive) specifying the version number.

For example, the template for the PROC1 procedure described in Section 3.3.1 would look like this:

```
PROC1:  PROCEDURE(X);
DECLARE X SCALAR;
CLOSE PROC1;
D VERSION #
```

will be a generally unreadable character whose hexadecimal value is the version number. For more information about the VERSION compiler directive, see Section 5.2.

Caution must be taken when an INCLUDE DD card is used to specify the location of a template library of a complex instead of an OUTPUT6 card. For even in this case the template produced by the actual compilation being performed will still be written to the OUTPUT6 dataset. If an earlier version of this template exists on one of the datasets specified on the INCLUDE DD card, then although the compiler will find it, recognize that it has to be updated, and emit the correct message, replacement will not necessarily take place since the new template will be placed in the OUTPUT6 dataset, and not in the INCLUDE dataset where the old one was found. The user should, therefore, make sure that if a template already exists for the unit actually being compiled, the library containing that template is specified on the OUTPUT6 card, instead of (or in addition to) being specified on the INCLUDE card.

3.5 SDF INCLUDEs

When compilation of a block of HAL/S code takes place, the Simulation Data File (SDF) produced by Phase 3 of the compiler contains a large amount of information describing the format of the block, the variables it involves, the data types of any formal parameters and function returns, etc. In particular, therefore, it contains all of the information which is normally placed into that block's template, and which must be placed in the code for any other block referencing that one. It is possible to instruct the compiler to INCLUDE not a template, but an SDF, and so to read this information directly from the Simulation Data File. The result in certain cases can be a significant saving in time, particularly when large COMPOOLs are being included.

No change needs to occur within the HAL/S source code itself. This means that with the possible exception of a modification in JCL, the use of this new facility can remain completely transparent to the user as long as an SDF file is available to the compiler. When the compiler comes upon a D INCLUDE TEMPLATE compiler directive it will begin its search by looking through the SDF library, analogous to the template library. This search goes through the partitioned dataset specified on a DD card with ddname HALSDF, and then through that specified on the OUTPUT5 DD card. If both of these

searches are unsuccessful, it then goes on to search for a template in the regular way (i.e., by referencing the INCLUDE DD and OUTPUT6 DD cards).

If all compilations are taking place in a single job, (see Section 3.3.1), no new JCL is needed to allow SDFs to be included; the catalogued procedures will do this automatically. Otherwise, as is the case for templates, JCL modifications must be made, either by modifying the HAL.OUTPUT5 DD card to refer to a permanent dataset, or by adding a HAL.HALSDF DD card. In either case, the dataset referred to should be a partitioned dataset serving as a library of SDFs for the program complex. All JCL modifications and considerations here are precisely analogous to those discussed in Sections 3.3.2 and 3.4 above for OUTPUT6 and INCLUDE DD cards.

Two new optional subcommands ("SDF" and "NOSDF") for the INCLUDE directive are now available with respect to the facility for SDF INCLUDEs. "D INCLUDE TEMPLATE <block name> NOSDF" may be coded to instruct the compiler not to look for any SDFs but to engage only in a search for templates. This may be used for example when you know that no SDFs produced for the unit being INCLUDEd are available. Furthermore, since only SDFs produced by compiler release 17 or later possess the correct structure for use by this new facility, the compiler checks the SDF version for $\geq$ 17. If true, it uses the SDF; otherwise, it searches the template library. In addition, although compilations producing object code for one machine generate templates which may be included by other compilations producing object code for a different machine, this is not the case with SDFs; again, the NOSDF subcommand should be coded to force template searching only. If you specify "SDF", the compiler will check to see that any SDFs searched are compatible with both the target machine or current compiler release. If there is any code discrepancy between the SDF and program complex, the compiler will go directly to search for the appropriate template.

The second subcommand now available can be used only when the block being referenced is a COMPOOL. COMPOOLs in many applications may be extremely large, and individual programs or COMSUBs will typically reference only a few of the variables included in them. The templates for a COMPOOL by its very structure is just as large as the COMPOOL itself, and thus much extraneous information is passed to the compiler when a COMPOOL template is INCLUDEd. By coding:

```
D INCLUDE SDF <name of COMPOOL>:<variable list>;
```

it is possible to instruct the compiler to retrieve only the information actually needed from the SDF for the external COMPOOL. <variable list> in this case is simply a list of one or more symbols defined in the COMPOOL named. Structure templates may also be included (and indeed must be if structure variables employing them are to be included). This is done by coding STRUCTURE <template name>. If <variable list> is long, it may be continued onto several successive 'D' cards following the INCLUDE SDF directive.

For example, suppose a COMPOOL named ALPHA contains a large number of variables as well as structure templates, and that information about only the variables BETA_STRUC, GAMMA, and DELTA are needed for the compilation of some procedure. BETA_STRUC is defined in the compool as being a (qualified) BETA-STRUCTURE;

GAMMA is defined as being a GAMMA-STRUCTURE; and so is unqualified. DELTA is a scalar. Instead of coding

```
D INCLUDE TEMPLATE ALPHA
```

which would cause the template for the entire COMPOOL to be INCLUDEd, one can now code:

```
D INCLUDE SDF ALPHA: STRUCTURE BETA,
D STRUCTURE GAMMA, BETA_STRUC,
D GAMMA, DELTA;
```

Notice that in order to employ this new SDF look-up facility, the compiler directive must have the form INCLUDE SDF <block name>. If INCLUDE <member name> is coded without the SDF keyword, search for an SDF will not take place (see Section 5.2 for more information on this form of the INCLUDE statement).

This page is intentionally left blank.

4.0 COMPILER LISTINGS

Following the Phase 1 compilation of a HAL/S program, the compiler produces a standardized, formatted listing of the source code. In addition, large amounts of other types of information are listed, some automatically, some under the control of user options. If errors in the source program were encountered during compilation, error messages are produced as well.

4.1 Format of the Compiler Listing

The HAL/S compiler has been designed to provide standard, automatic annotation of its output listing in order to enhance the readability of HAL/S source code. The HAL/S system allows each programmer to enter programs in a free-form input, without stringent rules regarding indentation, statement labeling, number of HAL/S statements per line, etc. The compiler, after generating object code, then edits the source program and prints it in such a way that all program listings will follow the same coding rules. Programmer flexibility is thus made possible without any loss in program readability or any increase in the difficulty of program maintenance.

One element in the compiler's standard formatting is the automatic expansion of single-line input into the full HAL/S multi-line form. User-written statements such as

```
C =((X$1)(A$l)+((X$2)**2)(A$2)+((X$3)**3)(A$3))/Y**3;
```

are printed in the much more natural and easily readable mathematical notation:

$$C = (X_1\ A_1 + X_2^{\,2}\ A_2 + X_3^{\,3}\ A_3)/Y^3;$$

Main lines of text are indicated in the compiler listing by an M at the beginning of the line; exponent and subscript lines are indicated by E and S respectively. Exponents raised to powers, or subscripted subscripts, etc., are indicated by more than one E or S line. Subscripted exponents, however, and subscripts raised to powers, are not expanded to multi-line format, in order to prevent confusion. Comment lines are indicated with a C in the compiler listing, and compiler directives are indicated with a D.

Exponent (E) lines are also used to print the characteristic HAL/S data type overpunches, indicating at each point what data type a variable is understood to be. The table below lists the overpunch characters and their corresponding data types.

CHARACTER	DATA TYPE
*	Matrix
-	Vector
.	Bit
,	Character
+	Structure

Exponent (E) lines are also used to print the overpunch for the "escape" character

translation (see Section 8.11).

CHARACTER	ESCAPE LEVEL
_	Level One
=	Level Two

The mark supplied is determined from the totally subscripted form of the variable. Thus the overpunch may be changed by subscripting. For instance, a matrix variable subscripted down to a particular row of the matrix (M$(2,*), for example) receives a vector mark; a matrix subscripted down to a single element receives no overpunch at all. Variables which possess arrayness are indicated by being placed in brackets ('[' and ']'), while those with multiple copies due to structure copyness are placed in enclosing braces ('{' and '}').

HAL/S is a block oriented language, and the logical indenting of program blocks can do a great deal to enhance the comprehensibility of a program's structure. The compiler automatically indents logically connected sections of program in the output listing. The body of a DO...END group, for example, is always indented, as are the THEN and ELSE clauses of any (compound) IF-THEN or IF-THEN-ELSE statements. The result is a carefully indented listing in which the logical structure of the program is immediately apparent. Programs indented in this way are always much easier to read and easier to understand, and they conform conceptually to the requirements of structured programming practice.

In addition to automatic indention, the compiler enhances program comprehensibility by printing additional information to the right of each line of a compilation unit. In most cases, this is the "current scope" which gives the name of the HAL/S block (PROGRAM, PROCEDURE, FUNCTION, or TASK) to which that line belongs. The following statements have information other than the current scope printed at the end of the line. Exceptions and more details for these cases are listed in Section 2.6.1 of the *HAL/S-FC Compiler System Specification*. 1) Cases specified in a DO CASE group within a block are counted and the case number is printed in the scope field. 2) For END statements, the statement number of its corresponding DO statement is printed in the scope field. 3) For IF-THEN statements followed by a simple DO statement, the two statements are printed on the same line if space allows and only the statement number for the IF-THEN is printed to the left of the statement. For clarification, the statement number for the DO is printed in the scope field in the form "DO=ST#X" where X is the statement number. 4) For IF-THEN-ELSE statements followed by a simple DO statement, the DO is printed on the same line as the ELSE if space allows. The value that appears in the current scope field will be truncated if it exceeds the maximum length.

Any identifier which has been defined in a REPLACE statement as a label to be replaced by text is indicated in the compiler listing by underlining. However, the replacement text-- that is, the text as it is actually constructed by the compiler -- is not printed.

If the replacement text contained exponents or subscripts, blank E or S lines will be

printed for each level.If, however, you place a cent (¢) sign around a REPLACE label in any statement referencing it, the original text which was replaced will be printed; for instance:

```
REPLACE ADD BY 'A+B';
K=¢ADD¢;
```

will output as:

```
K=A+B;
```

in the formatted listing.

With one exception, no more than one HAL/S statement ever appears on a line in the output produced by the compiler. An IF-THEN statement followed by a simple DO statement will appear on the same line, unless the line is too long. Sometimes a single HAL/S statement is broken up over more than one line. This occurs if the statement is too long, of course; but it also occurs when it will aid comprehensibility of the program. Thus, in an IF-THEN-ELSE statement, each clause is placed on a separate line; also compound declarations are broken up to make them easier to read. The compiler automatically numbers HAL/S statements (not lines) and prints the number of the current statement to the left of each M line. In instances where the line following a DO CASE statement contains a label, a sequencing error will occur, giving the same statement number to two HAL/S statements (DR 103418, 10/22/90). Errors discovered by the compiler are flagged in the listing and an error message is generated; see Section 4.3 for more information.

A sample compilation listing is given in Figure 4-1.

```
SRN STMT    SOURCE  CURRENT SCOPE
 000002      1 D|   INCLUDE TEMPLATE C2713A                              |AB|
             *****INCLUDED FROM SDF ##C2713A *****
             *****RVL  CATENATION NUMBER 1 *****
000009         C|                                                        |AB|
000010       2 M|    C12713:                                            |AA|C12713
000010       2 M|    PROGRAM;                                           |AA|C12713
000020         C|                                                        |AA|C12713
000021         C|                                                        |AA|C12713
000030       3 M|   REPLACE ONE BY "1";                                 |AA|C12713
000070       4 D|   INCLUDE C2713B NOLIST                               |AB|C12713
            *****START OF INCLUDED MEMBER, RVL , CATENATION NUMBER 1*****
             *****END OF INCLUDED MEMBER, RVL , CATENATION NUMBER 1*****
000080       6 M|     DO CASE I;                                        |BA|C12713
000090       7 M|   1   K = ONE;                                        |BA|CASE 1
000100       8 M|   1   K = 2;                                          |BA|CASE 2
000110       9 M|     END;                                              |BA|ST#6
000120         C|                                                        |AA|C12713
000170      10 M|     IF K = 1 THEN DO;                                 |AA|DO=ST#11
000180      12 M|   1    IF J = 2 THEN DO;                             |AA|DO=ST#13
000190      14 M|   2       IF I = 4 THEN DO;                          |AA|DO=ST#15
000200      16 M|   3         I = 1;                                   |AA|C12713
000230      17 M|   2       END;                                       |AA|ST#15
000240      18 M|   2       ELSE DO;                                   |AA|C12713
000250      19 M|   3         I = 16;                                  |AA|C12713
000280      20 M|   2       END;                                       |AA|ST#18
000281      21 M|   1     END;                                         |AA|ST#13
000290      22 M|   1     ELSE                                         |AA|C12713
000290      22 M|   1       DO FOR I = 1 TO 3;                         |AA|C12713
000300      23 M|   2         K = I J;                                 |AA|C12713
000320      24 M|   1       END;                                       |AA|ST#22
000330      25 M|      END;                                            |AA|ST#11
               E|      *    *                                         |  |
000341      26 M|      M1 = M2;                                        |AA|C12713
000342      27 M|      M1    = M2   ;                                  |AA|C12713
               S|        1,3     2,1                                  |  |
               E|      -       -                                     |  |
000343      28 M|      M1    = M2   ;                                  |AA|C12713
               S|        1,*     2,*                                  |  |
000350      29 M|   CLOSE;                                             |AA|C12713
```

Figure 4-1 Sample Compilation of Source File

4.2 Information Provided by the Compiler

Much information is provided in the compilation listing in addition to the formatted source text. Each major phase (Phases 1, 2, 3, and 4) of the compiler produces a listing. The Phase 1 listing begins with a "complete list of compile-time options in effect", listing all of the compiler options chosen by the user or else the defaults which were employed instead (see Section 5.1 for a list of the available options and defaults).

The option listing is followed by the formatted source code. After each code block has been printed (i.e., after each PROGRAM, FUNCTION, PROCEDURE, TASK, or UPDATE) a "BLOCK SUMMARY" is provided which offers a summary of significant

actions taken within the block: such as, which PROGRAMS and TASKs were scheduled, terminated or canceled and which had their priorities updated; which event variables defined outside the current block were changed or used in event expressions; which external COMSUBs were called and which PROCEDUREs and FUNCTIONS defined in the same program but outside the current block were called; which errors were sent by SEND ERROR statements; which variables, replace statements, or structure templates defined in a COMPOOL were employed, and which variables, replace statements, or structure templates defined in the same program but outside the current block were used.

Any variables which are assigned within the block appear with a * after the name. Variables which are local to the block (i.e., are DECLARED within it) are not included; neither are locally defined procedures, functions, replace statements, structure templates, event variables, etc. This is because the block summary is intended to make clear only the impact of a block on its external environment. Also, the summary for a block will not repeat information already provided by the summary of a block nested within it (i.e., if procedure A contains procedure B which schedules program C, procedure B's block summary will indicate that program C was scheduled, but procedure A's will not).

Following the source program, an overall "COMPILATION LAYOUT SUMMARY" is provided, indicating the way in which PROGRAMS, TASKs, PROCEDUREs, FUNCTIONs, and UPDATE blocks were defined. The indentation level in this listing expresses graphically the nesting level of the block within the overall program. This serves to give a quick overview of the program structure. Such a listing can be of assistance not only as a documentation aid, but also as a guide to locating the definition of procedures and functions which have been diagnosed as undefined by the compiler.

The compiler also produces a complete Symbol and Cross Reference table. This table is divided into two parts. The first part is the "STRUCTURE TEMPLATE SYMBOL AND CROSS REFERENCE TABLE" which lists all structure templates from the source program alphabetically in a complete and fully cross-referenced form. The identifiers defined within each template are listed beneath it. The second part is the "SYMBOL AND CROSS REFERENCE TABLE" which contains every variable, statement label, and block name from the source program alphabetically in a complete and fully cross-referenced form.

Each line in the cross reference table provides several pieces of information about the specific identifier described. Under DCL, the number of the statement in which the identifier is first declared is printed. For explicitly DECLAREd variables, this is the number of a DECLARE statement; otherwise it is the number of the statement in which the identifier is first used. Under NAME, the name of the identifier is listed, preceded by an asterisk if the variable was implicitly defined (e.g., inline functions). Identifiers which appear in structure templates as structure terminals or minor structures are indented one space from the structure template name under which they are defined.

Under TYPE, the TYPE of each identifier is described. For variables, both data type and dimension are listed; e.g., INTEGER, 3-VECTOR, 6x2 MATRIX, CHARACTER(80),

STRUCTURE(5) - (this last indicating copyness). Arrayness is indicated, but not array size, as by SCALAR ARRAY or BIT(8) ARRAY, for example. NAME variables are indicated by preceding their type with NAME, as in NAME SCALAR. Structure templates are described as such, while minor nodes and structure terminals within them are listed with a structure level number along with their types, and are furthermore indented to indicate the hierarchical organization of the template of which they form a part. PROGRAMS, PROCEDUREs, TASKs, and COMPOOLs are listed as such; FUNCTIONs are listed along with the type of the value they return - 2-VECTOR FUNCTION, for example. UPDATE block names are listed as UPDATE LABEL; statement labels as STATEMENT LABEL. REPLACE macros and their arguments are also listed, as REPLACE MACRO and MACRO ARG.

The "ATTRIBUTES AND CROSS REFERENCE" field indicates first the declaration attributes of variables or block names. Here the size of arrays is indicated, as are the precision specification, lock group number, and the presence of any attributes such as INITIAL, DENSE or ALIGNED, STATIC or AUTOMATIC, ACCESS, REENTRANT, EXCLUSIVE, REMOTE, and/or LATCHED. Variables defined in TEMPORARY statements have TEMPORARY in this field. Parameters to functions and procedures are listed as INPUT PARM or ASSIGN PARM. Procedures and functions themselves may be identified as EXTERNAL. Structures are listed as <template name>-STRUCTURE. REPLACE macros have MACRO TEXT ="<text>" listed in the attribute field. Variables (other than 16-bit NAME variables and structure templates) which are REMOTE due to their COMPOOLS being included REMOTE will have the attributes of INCREM and REMOTE. 16-bit variables that are included REMOTEly will only have the INCREM attribute set (not the REMOTE attribute). For structure templates, all structures declared using the template are also listed in this field.

Immediately following the attributes, a complete list of cross references to the identifier appears, after XREF=. These indicate each place in the program at which the identifier was referred to, and take the form N XXXX. XXXX here is a four digit specification of the Internal Statement Number (ISN) and N is a cross reference flag which specifies the way in which the identifier was used. A value for N of 0 indicates the ISN on which the identifier is first defined; 1 indicates the ISN in which it is used in a subscript; 2 indicates an ISN in which it is referenced (e.g., appears on the right of an equals sign or in a WRITE statement) and 4 indicates an ISN in which it is assigned into (e.g. appears on the left of an equals sign or in a READ statement). If more than one of these things occurs in a single line, the sum of the applicable flags is printed; thus, if ISN 56 in a program is "K=K+J;" the cross-reference information for variable K would be 6 0056, the 6 indicating that reference (2) and assignment (4) take place there. The definition information will be printed separately if the variable is referenced in the same statement in which it is defined (e.g. NAME nodes that point to the structure template name, compound or factored declares that reference variables declared in the same statements, or TEMPORARY loops that both define and assign variables).

Similarly, when a structure template is defined and referenced in a COMPOOL and later referenced in a compilation unit that includes the COMPOOL, there will be two XREF

entries created for the structure template at the INCLUDE statement: one definition XREF and one reference XREF. For all other data typed variables, there will only be a definition XREF for the INCLUDE statement.

Warning messages may be inserted following the list of cross-references. "NOT USED" will appear if the identifier appears in a DECLARE statement but is neither referenced nor assigned. "NOT REFERENCED" appears if the identifier is assigned a value but is never used in any reference context. Both of these describe situations which are unlikely but will cause no difficulty in execution. If a variable is referenced but never assigned into or initialized, however, clearly an error has occurred, and execution will not be meaningful. In that case the message will be "*****ERROR*****REFERENCED BUT NOT ASSIGNED." It is not always possible to determine at compile time if a variable has been initialized before being referenced. Refer to Section 8.13 of this document for additional information and examples.

Identifiers which are structure terminals or minor nodes are listed twice in the cross-reference table - once as part of a structure template and once as an actual structure. The second time no attribute or cross-reference information is given and instead "*****SEE STRUCTURE TEMPLATE <template name>" is printed. In many cases a structure terminal or node will never be explicitly referenced or assigned, although implicitly it will be used in statements involving the overall structure. Therefore, the warning messages omitted by the compiler take on a more cautious tone when referring to such identifiers - "POSSIBLY NOT USED" or "POSSIBLY NOT REFERENCED", etc., are printed out.

Following the cross-referenced table a "Built-In Function Cross Reference" table is printed. This simply lists all of the HAL/S built-in functions which are used in the program, with information as to where and how they are used printed in the same N XXXX format described above.

Next the compiler prints a report on the actual size reached by various compiler tables for which a type 2 compiler option exists to control their allocated size (see Section 5.1). The report shows the requested size, the amount actually used, and a <-<-<- flag to indicate if the initial allocation was exceeded and automatically extended. This information may be needed to change the requested sizes to reflect actual requirements more closely. SYMBOLS refers to the number of identifiers used in the program; MACROSIZE to the total number of characters used in replace macro definitions; LITSTRINGS to the number of characters used in character literals; XREFSIZE to the number of entries in the cross-reference table; and BLOCKSUM to the size of the tables used in collecting information for BLOCK SUMMARIES. If the LISTING2 compiler option was specified, an unformatted verbatim listing of the source program is now printed. Then a series of performance statistics useful for those involved in work on the compiler is listed; these are of no concern to the ordinary programmer, and will not be discussed. A "Summary of Errors Detected in Phase 1" is also printed; see Section 4.3 for more information. This ends the listing generated by Phase 1 of the compiler.

If the compiler option LIST has been specified, a completely formatted listing of the object module produced by the compiler will also be printed. First, a list of all control

section names needed by the compiled program is emitted, in a format typical of IBM/360 compilers and translators. Then the object code is listed. This listing shows the complete code generated for each control section in both hexadecimal and pseudo-assembler formats. As each new HAL/S statement is encountered in the production of object code, a line is printed in the format:

```
ST#n    EQU    *
```

where n is the HAL/S statement number. A LABEL field is also given in order to indicate both HAL/S label names and internal branch points. A SYMBOLIC OPERAND field gives information about the symbolic operand referenced by the instruction. Following this is a map of relocation information included in the produced object module, and then more information about compiler performance. An error summary for Phase 2 is also printed; see Section 4.3. Toward the completion of Phase 2, specific compiler performance statistics (useful only to those involved in compiler development) are printed, as are optional table sizes.

The Relocation Dictionary (RLD) information is then printed in a table format. The table contains the Position Pointer (POS.ID), the Relocation Pointer (REL.ID), FLAGS, the ADDRESS, and the CSECT for both POS.ID and REL.ID. POS.ID is the External Symbol Dictionary Identification Number (ESDID) of the Section Definition (SD) for the control section that contains the address constant. REL.ID is the ESDID of the External Symbol Dictionary (ESD) entry for the symbol being referred to. FLAGS is the flag field in the form TTTTLLSN, where TTTT is the type, LL is the length of the address constant, S is the direction of relocation and N is the type of the next RLD item. For BFS, the CSECT for REL.ID is blank for LEC entries.

A Phase 3 compiler listing is produced, informing the user of any INCLUDEd COMPOOLs which are not used in the program, and whether the Simulation Data File for the compilation unit was created or replaced.

Following the Phase 3 listing, an advisory section is printed. This section contains information for the programmer about improvements that could be made to the source code of the program.

Finally, in Phase 4, if the TABDUMP or TABLST compiler option was specified by the user, a listing of the SDF produced by Phase 3 is printed. The listing consists of a Translation Table which gives the identification codes (decimal and hexadecimal) assigned to blocks, symbols, and statements in the program, listing the associated block name, symbol name, and, if present, SRN, together with the virtual memory pointer to the corresponding file node; and then a hexadecimal dump of the file, page by page. If TABLST rather than TABDMP was specified as an option, a formatted listing of this information is provided.

Note that if NOTABLES was specified as an option, Phase 3 and Phase 4 are not executed: no Simulation Data File is produced, and no listing is printed either. See Section 5.1 for information about all these options.

4.3 Error Messages Produced By the Compiler

When Phase 1 of the HAL/S compiler (the syntax checking phase) detects an error condition, a diagnostic message describing the error is placed in the primary (formatted) source program listing at the point where it was detected. These error messages take the following form:

```
*****c ERROR #n OF SEVERITY s.*****

*****text of error message
```

After all error messages other than the first one in a given compilation, the following line is also printed:

```
*****LAST ERROR WAS DETECTED AT STATEMENT m*****
```

The letter c here represents a mnemonic error name uniquely identifying this message; n indicates the number of errors which have so far been identified in the compilation; s is a severity code; and m is the HAL/S statement number of the most recent previous statement that received an error message.

The compiler can recognize any one of a large number of error conditions, and responds by searching through the built-in ERROR library to find the correct message. Each error (and each message) has a unique mnemonic designation or name used by the compiler to search the ERROR library. The designation may be somewhat helpful to the programmer because it describes in symbolic form the type of error encountered. The error name consists of one or two letters, followed by a one or two digit number. The first letter designates a "major error class" (e.g., errors related to assignment statements, subscripting errors, etc.), while the second letter, if present, indicates a sub-class further describing the error. The number specified simply indicates members of a class - subclass group.

The severity indication s in the error message shows the effect of the error on the compilation process. The possible severities and their effects are as follows:

- s = 0 - warning; compilation proceeds normally, execution still possible.
- = 1 - error; compilation proceeds, execution prevented.
- = 2 - severe error; syntax check (Phase 1) continues, code generation (Phase 2) prevented.
- > 2 - abortive error; compilation halts immediately.

No template is generated if a severity 1 (or greater) error occurs in Phase 1. No SDF is generated if a severity 1 (or greater) error occurs before Phase 3.

A complete list of error groups and subgroups, as well as each of the actual error names and messages possible and their severities, can be found in Appendix B, along with more detailed explanations of many of the messages and suggestions as to how the errors may be fixed.

In many cases when the compiler encounters an error condition, it makes some arbitrary

but reasonable assumptions as to what actually was meant by the statement in which it found an error. Such assumptions allow syntax checking to continue so that other errors may be discovered. They may also, however, cause the compiler to find "spurious" errors later on in the program, as a result of the compiler having made the wrong assumption. Thus, if an identifier is discovered which has not previously been declared, under most conditions the compiler, after emitting a "DU1 UNDECLARED IDENTIFIER" message, treats it as a scalar. If however, the identifier had actually been meant by the programmer to represent a BOOLEAN, for instance, and if a later statement attempted to assign the value "TRUE" to it, another error message would result from the compiler's incorrect assumption. Error conditions which are especially likely to have this effect are flagged with the message "ERROR RECOVERY MAY CAUSE SUBSEQUENT SPURIOUS ERRORS"; when debugging, programmers should remain alert to such a possibility in order not to waste time in a futile attempt to discover the source of a spurious error.

Near the end of the Phase 1 source listing and table printout, a summary of detected errors will be printed, in the following form:

```
x ERRORS WERE DETECTED.  THE MAXIMUM SEVERITY WAS y.  THE LAST ERROR WAS
DETECTED AT STATEMENT z.

     *****SUMMARY OF ERRORS DETECTED IN PHASE I*****

       ERROR# 1 of SEVERITY s1 AT STATEMENT e1

       ERROR# 2 of SEVERITY s2 AT STATEMENT e2 etc.
```

Phase 2 of the HAL/S compiler (the code generation phase) may also produce some error messages. These messages have the form:

```
         <text of message>

  ***c ERROR n of SEVERITY s DURING CONVERSION OF HAL STATEMENT x***
```

where x is the HAL/S compiler-assigned statement number to which the error refers. The mnemonic error names for Phase 2 errors are very similar to those for Phase 1 errors, except that the number is always three digits instead of one or two. They too are listed and explained in Appendix B. For the most part, Phase 2 errors will only occur if Phase 1 errors were detected, and indeed the result of these same errors being reencountered as code generation proceeds. Resolving the problems which led to the various Phase 1 errors, therefore, will often cause Phase 2 errors to disappear.

In addition to printing out the Phase 2 error messages, the compiler also produces a disposition message indicating the total effect of all Phase 2 errors on the compilation.

If there are attempts to downgrade an error and the downgrade was successful, then the error message will still be printed, along with a message of the form:

```
******   THE FOLLOWING ERROR WAS DOWNGRADED FROM A SEVERITY ONE

ERROR TO A SEVERITY ZERO ERROR   ******

          <text of message>
```

The severity will be changed to zero to allow compilation to continue. However, if an attempt is made to downgrade an error that does not have a severity of one, then a severity two error message will be produced. If downgrades are attempted then a downgrade summary is produced in Phase 2, unless compilation is to be halted, then the summary will be printed from the latest phase.

The form of the Downgrade summary is:

```
************   DOWNGRADE SUMMARY   *************

***ERROR NUMBER c AT STATEMENT NUMBER n***

***WAS DOWNGRADED FROM A SEVERITY ONE ERROR TO A SEVERITY ZERO ERROR
MESSAGE***
```

The letter c here represents a mnemonic error name uniquely identifying this message; n indicates the statement number that this error occurred in.

If the downgrade was not successful the following messages will appear in the downgrade summary:

```
***********  DOWNGRADE SUMMARY   *****************

***ATTEMPTED DOWNGRADES THAT WERE NOT DOWNGRADED****

**ERROR NUMBER c FOR STATEMENT NUMBER n WAS NOT DOWNGRADED,

REMOVE DOWNGRADE DIRECTIVE AND RECOMPILE***
```

If a downgrade is unsuccessful due to it not being needed, then compilation will return a failure code but the object code and SDF will still be generated.

For further information concerning downgrading error messages, see Section 5.2 concerning compiler directives.

This page is intentionally left blank.

5.0 USER-SPECIFIED OPTIONS

Much of the compiler's activity is potentially under user control. This is particularly true of the compilation listing, but also holds with respect to many elements of the compilation process proper. The user exercises such control by specifying compiler options and directives. Compiler options are called for by the user in the JCL for the compilation step by being coded on the EXEC card defining that step. Compiler directives, on the other hand, actually appear in the source code to be compiled. Such directives are signaled to the compiler by being placed in "compiler directive lines" which are indicated by coding a D in the first column. Two of these - INCLUDE and VERSION - have already been partially described in Chapter 3. This chapter describes all of the various options and directives available to the user.

5.1 Compiler Options

Compiler options must be specified in the PARM field of the EXEC card in the user's JCL which actually invokes the HAL/S compiler. In all cases, options are separated in the PARM field by commas (and no blanks).

Two distinct types of options exist. Type 1 options are controlled by the presence of keywords in the PARM field, and take on the Boolean values "on" or "off". To specify that a particular option is desired - i.e., that its value should be "on" - simply its name should be coded in the PARM field. To specify that its value should be "off" - i.e., that it is not desired - its name preceded by the characters 'NO' should be coded. The LIST option mentioned above in Section 4.2, for instance, is a type 1 option: either one desires to have a listing of the object code produced and so specifies LIST, or else one does not, and so specifies NOLIST.

Type 2 options, on the other hand, have numeric or character "values" which may be set by the user. They are specified by coding what looks like an assignment statement in the PARM field. The LITSTRING option is of type 2, for instance; a user would specify, e.g., that no more than 75 character literals will be permitted in his or her program by coding LITSTRING=75 in the PARM field. (Note that the amount used for each character string in the program is 1 more than the number of characters in the string).

It is not necessary to code a value for every available option in the JCL. Each option has a default value defined for it, and in the absence of any reference to an option in the PARM field the compiler assumes that the default value is desired. The default for the LIST option, for instance, is "off"; hence if neither LIST nor NOLIST is coded in the user's JCL, NOLIST will be assumed and no object code listing will be produced.

Many options have alternate, shorter spellings which may be used interchangeably with the standard option names in the user's JCL. LIST, for example, may also be coded as L; SYMBOLS may be coded as SYM. To turn a type 1 option with such an alternate form "off", N should be written instead of NO before the shorter name: "NL" therefore is equivalent to NOLIST. Note that "NOL" or "NLIST" will not be recognized by the compiler.

The following complete list gives the name, alternate name, default value, and function of each available option, first for type 1 and then for type 2.

Type 1 Options

(Alternate) Keyword	Default	Function
ADDRS (A)	off	Causes the Simulation Data Files to include statement address information.
DECK (D)	off	If specified, an additional copy of the object file produced by Phase 2 is produced and sent to the dataset defined on the OUTPUT4 DD card.
DUMP (DP)	off	Requests the compiler to produce a memory dump if internal compiler failures occur; useful only for compiler generation and debugging.
HALMAT (HM)	off	If specified, HALMAT and literal tables are included in the SDF generated for this compilation. NOHALMAT will reduce the size of the SDF considerably.
HIGHOPT (HO)	off	The high optimization option (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. The high optimization option should not be used if any of the following conditions exist within the compilation unit. a) A NAME variable is used to modify data that is of a type different from that which the NAME variable was declared to point. This can occur if the variable that the NAME points to is modified via a %NAMEADD or %NAMECOPY. If the NAME variable is in a COMPOOL, the %NAMEADD or %NAMECOPY can occur in any compilation unit that includes that COMPOOL. b) A %COPY is used to copy data beyond the bounds of the destination variable listed in the %COPY statement. This can occur when the %COPY destination resides in a COMPOOL or when a variable count is used.
LFXI (NONE)	on	The LFXI option causes the following: 1) Use of the LFXI instruction to load a register with integer literal values, if the values are > = -2 and < = 13, excluding 0. If a literal value is outside this range or NOLFXI is selected, the LHI instruction will be used to load the register. In both cases, a literal 0 is handled by the ZH instruction.

Type 1 Options

(Alternate) Keyword	Default	Function
		2) Use of the LFLI instruction to load a register with scalar literal values (1.0, 2.0,...15.0). If a literal value is outside this range, or is not a whole number with a zero fraction, and is not equal to 0.0 or NOLFXI is selected, the LE instruction will be used to load the register. A value of 0.0 will use the SER instruction to clear the register.
LIST(L)	off	Produces an assembly listing from Phase 2 of the compiler (see Section 4.2).
LISTING2(L2)	off	Causes unformatted source listing to be generated (see Section 4.2) and sent to the dataset defined on the LISTING2 DD CARD.
LSTALL (LA)	off	Debug. Gives assembly and HALMAT in listing.
MICROCODE (MC)	on	Allows use of instructions which only exist on late versions of the Space Shuttle GPC. This includes SCAL, SRET, MVS, MVH, and BIX.
PARSE (P)	off	Debug. Gives parsing location information if an error is encountered.
REGOPT (R)	off	Used to indicate to the compiler that register optimization is desired. Allows borrowing of register 3 for addressing COMPOOL data. Unlike the SDL option below, the stack setup is performed, allowing execution on the APES simulator. If the DATA_REMOTE directive is in effect, register optimization is not performed, even if REGOPT and/or SDL is specified.
SCAL (SC)	on	Coding NOSCAL specifically inhibits the use of the SCAL and SRET instructions for subroutine linkage, even if the MICROCODE option was also chosen. MICROCODE and NOSCAL together thus cause HAL/S linkage to be used instead of the SCAL/SRET instructions. If NOMICROCODE was specified, neither SCAL nor NOSCAL has any effect (see note above with respect to error handling). Implemented in BFS only.

Type 1 Options

(Alternate) Keyword	Default	Function
SDL (none)	off	Used only to indicate to the compiler that this compilation is for SDL (Software Development Lab) use. Certain actions specific to SDL operation are automatically performed, such as inclusion of SRNs, Change Authorization Fields, and source record revision indicators in the primary listing. As with the REGOPT option above, this option indicates to the compiler that register optimization is desired. If the DATA_REMOTE directive is in effect, register optimization is not performed, even if REGOPT and/or SDL is specified. One side effect is that the stack setup necessary for the APES is not performed. Declaration of REMOTE (#R) data is restricted (see section 8.12).
SREF (SR)	off	Causes only those variables from an included EXTERNAL COMPOOL which are actually referenced by the unit being compiled to be printed in the cross reference listing.
SRN (none)	off	This should be specified if the source program is line numbered. It causes the compiler to scan columns 1-72 only, allowing columns 73-78 to be used for line numbers.
TABDMP (TBD)	off	Causes Phase 4 of the compiler to produce a hexadecimal dump of the Simulation Data File.
TABLES (TBL)	on	If not specified, Phases 3 and 4 of the compiler are inhibited. Simulation Data File generation does not take place, and FC real-time simulation, debugging, etc., are not possible (although normal execution can still occur).
TABLST (TL)	off	Causes Phase 4 of the compiler to produce a formatted dump of the Simulation Data File. If NOTABLES is specified, this will have no effect.
TEMPLATE (TP)	off	Causes the generation of a template for the compilation unit.
VARSYM (VS)	off	If specified, SYM records will be included in the object module produced by the compilation.
ZCON (Z)	on	Causes calls to out-of-line routines (external references) to be performed via long indirect address constants.

Type 2 Options

Standard Form (Alternate)	Default	Function
BLOCKSUM= (BS=)	400	Specifies the initial size of the internal compiler table used for collecting information for printing block summaries. As this value is exceeded, the system will automatically allocate more space for it within the available memory.
CARDTYPE= (CT=)	null	Allows statements with non-standard characters in the first column to be treated in the standard HAL/S fashion. A mapping of non-standard into standard types (E, M, S, C, D and blank) is specified by coding pairs of characters. E.g., if "CT=XCYM" were coded, any lines with X in the first column would be read as comments (C), while lines with Y in the first column would be read as regular main source code lines (M).
COMPUNIT= (CU=)	0	Specifies a compilation unit number, which is made available in the SDF and in the Block Data Areas, and which allows an analysis of active blocks in a core dump.
DSR= (none)	1	Specifies the value to be used for the data sector register in the right hand halfword of the R2 operand of the MVH instruction. The compiler will use this value explicitly when it is not possible to use a standard z-type address constant in which the DSR field is filled in by the link editor.
LABELSIZE=(LBLS=)	1200	Specifies the initial number of labels which can be recognized by the compiler. As the value is exceeded, the system will automatically allocate more space for it within the available memory.
LINECT= (LC=)	59	Sets the maximum number of lines to be printed on any one page of the primary or secondary (unformatted) source listing.
LITSTRING= (LITS=)	2500	Specifies the maximum total number of characters permitted in all character literals in the program to be compiled. (Note that the amount used for each character string in the program is 1 more than the number of characters in the string).

MACROSIZE= (MS=)	500	Specifies the initial number of characters allowed in the text of all REPLACE macro definitions. As the value is exceeded, the system will automatically allocate more space for it within the available memory.
MFID=	null	Allows specification of the Major Function ID. Implemented in PASS only.

Type 2 Options

Standard Form (Alternate)	**Default**	**Function**
OLDTPL=	0	Determines the logical record length of the TEMPLATE file. When OLDTPL is not specified, an 80 byte TEMPLATE is generated. If OLDTPL='Y', then, if SRN or SDL option is specified, a 72 byte TEMPLATE is generated, otherwise an 80 TEMPLATE file is generated. Implemented in BFS only. As of 30v0/15v0 this option is no longer valid, and all TEMPLATE files are created with an 80 byte record length.
PAGES= (P=)	2500	Sets the number of pages to be used for the primary compilation listing.
SYMBOLS= (SYM=)	200	Specifies the initial size of the symbol table to be used by the compiler. As the value is exceeded, the system will automatically allocate more space for it within the available memory. Each symbol requires 53 bytes of storage plus 1 byte of storage for each character in the symbol name.
TITLE= (T=)	null	Specifies a (1 to 60 character) title to be printed by the compiler as a header at the top of each page of the listing. The header should be specified precisely as it is to appear; no quotes are necessary.
XREFSIZE= (XS=)	2000	Specifies the initial number of cross reference table entries available to the compiler. As the value is exceeded, the system will automatically allocate more space for it within the available memory. Each entry requires four bytes of storage.

5.2 Compiler Directives

A compiler directive is a specialized HAL/S command which serves to specify information to the compiler about the listing and other auxiliary functions of the compilation. Directives appear as actual lines within the source code, and are recognized by the compiler by a D in the first column of the source line. The following compiler directives have been defined for the HAL/S compiler.

1. The DEVICE directive has the form:

```
D  DEVICE CHANNEL=n <option>
```

This directive sets the mode of the specified channel (referred to via the CHANNELn DD card) to be the mode indicated by the <option>. The <option> may be "PAGED", "UNPAGED", or null (in which case UNPAGED is assumed).

Character strings produced as output by channels which are designated as UNPAGED will appear within quotation marks. Channels designated as PAGED, on the other hand, will write a character string as output simply as is. In the absence of a DEVICE directive, the compiler determines a default mode for a given channel in the following way. Channels used only in WRITE statements are presumed to be PAGED, while those used in READ or READALL statements are presumed to be UNPAGED. Thus, if output channel 6 is referred to by WRITE(6) statements only, it will be PAGED by default. If character data written to channel 6 is later input to the program via a READ(5) statement, then since READ expects character strings to appear within quotation marks, a runtime error will occur. This may be avoided by coding D DEVICE CHANNEL= 6 UNPAGED.

No more than one DEVICE directive may be specified for each channel. Channels which are used to READ data may not be defined as PAGED.

Note that mode setting of channels is strictly a compile-time action; when more than one compilation unit is involved, incorrect channel definitions may cause mode conflict at runtime. More information on the effect of mode setting may be found in Sections 6.1.1 and 6.1.3.

2. The INCLUDE directive has the form:

```
D  INCLUDE <name> <option>
    or
D  INCLUDE TEMPLATE <unit name> <options>
    or
D  INCLUDE SDF  <unit name>  <options>
```

One use of the INCLUDE directive is to reference needed templates or SDFs for programs, compools, or comsubs compiled separately from the main compilation unit. Details on this particular aspect of the INCLUDE directive may be found in Chapter 3.

The INCLUDE directive may also be used as a way of avoiding the repeated insertion of a common chunk of source code into a program. This facility allows one to include a given code sequence more than once without calling a PROCEDURE, and may also be used to assist in program structuring. The facility may be used in two ways:

a. The source code to be repeatedly inserted may reside as a member of a partitioned dataset. In that case, this dataset should be defined in the JCL with the same kind of HAL.INCLUDE (or HAL.OUTPUT6) DD card used to define the template library (see Section 3.3.2). Standard JCL concatenation may be used; e.g., if a PDS named TEMPLIB contains templates to be referenced by the compilation, and another one named OTHERLIB contains other bits of source code, then the following two lines of JCL would be needed:

```
// HAL.INCLUDE DD DSN=TEMPLIB,<other parameters>
//             DD DSN=OTHERLIB,<other parameters>
```

Note that the dataset referred to must be the entire library, not the specific member desired.

At the point in the source text where the other text is to be inserted, an INCLUDE directive should be coded, in the following form:

```
D INCLUDE <name> <option>
```

<name> is the precise member name of the file in the INCLUDE library which is to be included here. The name must be a 1-8 character string. The only <option> available is NOLIST, in which case the included text is not printed in the primary compiler listing; if no option is coded all included text is listed.

When the compiler encounters such a directive, it searches through the dataset(s) defined on the HAL.INCLUDE DD card for a member with the required name; if none is found, it also searches through the dataset defined on the HAL.OUTPUT6 DD card (see Section 3.3.2).

b. The user can also specify HAL/S source to be in-stream without having to pre-define the text in an INCLUDE library partitioned dataset. The DEFINE and CLOSE directives are used first to define the text to be included, and then later INCLUDE directives referencing this text.

To define a local INCLUDE member, the following directives are used:

```
D DEFINE <member name> [LIST]
                .
                .  <text for member>
                .
D CLOSE <member name>
```

The specified member name may be from 1 to 8 characters long, and must be specified on both the DEFINE and CLOSE directive. The optional LIST parameter will cause the source text to be listed in the same manner as comment cards (i.e., no reformatting will take place since no compilation takes place at this time). The default is not to list the source lines, since they will be listed when INCLUDEd later.

The normal INCLUDE directive is then used to retrieve the defined member:

```
D INCLUDE <member name> [NOLIST]
```

To use this facility, the following JCL card should be added to the HAL compile step:

```
//HAL.OUTPUT8   DD UNIT=SYSDA,SPACE=(TRK,(2,2,1)),
//              DCB=(RECFM=B,LRECL=80,BLKSIZE=1680)
```

This causes locally defined included text to be temporarily placed on an OUTPUT8 dataset; if such a card is present, any later INCLUDE directives cause the compiler to search OUTPUT8 before it searches any INCLUDE libraries.

It is permissible to have INCLUDE directives specified within local DEFINEd sequences. However, the text to be included must be previously defined, either as a previous local definition, or as a member of an existing INCLUDE library. Such INCLUDEd text actually becomes a part of the member being defined. It is not permissible to have INCLUDE directives within INCLUDEd sequences.

c. SUMMARY

The following table summarizes the various possible forms of the INCLUDE directive, and indicates which DD names are searched, and in what order, to find the code which is to be included:

`INCLUDE <name>`	`OUTPUT8,INCLUDE,OUTPUT6`
`INCLUDE SDF <block name>`	`HALSDF,OUTPUT5`
`INCLUDE TEMPLATE <block name>`	`HALSDF,OUTPUT5,INCLUDE,OUTPUT6`
`INCLUDE TEMPLATE <block name> NOSDF`	`INCLUDE, OUTPUT6`

<name> must be the explicit member name under which the text to be included is filed. <block name> is the name of a compilation unit; the compiler automatically computes the explicit member name and searches for it (see Sections 3.2 and 8.10 for information about the member names under which templates and SDFs are stored).

The following options are available for use with the INCLUDE compiler directive:

NOLIST	Indicates that the INCLUDEd source is not to be listed in the primary compilation listing.
REMOTE	Indicates that the external COMPOOL being INCLUDEd is to be addressed via ZCONS (INCLUDE TEMPLATE and INCLUDE SDF only).
NOSDF	Forces the INCLUDE to be from a template; the SDF library is not searched (INCLUDE TEMPLATE only).

In addition, the INCLUDE SDF directive will also support the following syntax:

```
D INCLUDE SDF <compool name> <options>:<name list>;
```

where <name list> is a list of one or more symbols defined in the COMPOOL named. See Section 3.5 for more details.

3. The PROGRAM directive has the form:

```
D PROGRAM ID=<id>
```

This directive provides a Program Identification Name to be used by the compiler to determine access rights to controlled resources as described in Section 8.6.

4. **The VERSION directive** has the form:

```
D VERSION @
```

where "@" is a character whose hexadecimal value is the version number. See Section 3.4 for a more detailed description of version numbers, and the form of this directive. Because generation of version numbers takes place automatically, the only situation where a version directive might be employed would be in a user-creation of a template.

5. **The EJECT directive has the form:**

```
D EJECT
```

and causes the compiler to begin a new page before resuming generation of the primary source listing. The EJECT directive is not itself printed on the listing.

6. **The SPACE directive** has this form:

```
D SPACE [<n>]
```

where n is an integer. This directive causes the compiler to skip <n> lines prior to generation of the next line in the primary source listing. If <n> is greater than 3, it will be set to 3. The SPACE directive is not printed on the listing.

7. **The DEFINE directive** has the form:

```
D DEFINE <name> <option>
```

This directive is used when a local INCLUDE member is placed within the body of a HAL/S program. It delimits the statements following it as that which can be INCLUDEd (very much like a HAL/S subprocedure) a number of times at different points in the program.

See "the INCLUDE directive" (2), above, for more information.

8. **The CLOSE directive** has the form:

```
D CLOSE <name>
```

This directive specifies the end of the text to be INCLUDED which began with the DEFINE directive. See "the INCLUDE directive" (2), above, for details.

9. **The DOWNGRADE directive** has the form:

```
D DOWNGRADE <error number>
```

where error number is the mnemonic error name which uniquely identifies the error to be downgraded. An example of an error to downgrade is as follows.

```
D DOWNGRADE Z01
```

This directive allows the user the ability to downgrade a severity one error message to a severity zero error message. The directive must be before the statement in which the error to be downgraded occurs. More than one directive may be applied to a single HAL/S statement if more than one severity one message is produced by the statement. If the directive is placed in the middle of the source for a HAL/S statement, the directive

will apply to the statement in which the directive is found. There is a limit of twenty-one downgrades per compilation unit.

10. The DATA_REMOTE directive has the form:

```
D DATA_REMOTE
```

or

```
DATA_REMOTE
```

This directive tells the compiler that all local data declared in the compilation unit (#D CSECT data) could be placed in a sector other than sector 0 or 1. This is useful in freeing up 16-bit addressable memory to meet the requirements of large program data areas, but forces the compiler to generate less efficient object code in order to address local data. The less efficient code is due to Data Sector Extension (DSE) manipulation and changes in register allocation.

The directive must be placed before the PROGRAM/PROCEDURE statement in the compilation unit, and is illegal in a COMPOOL compilation. More than one instance of the DATA_REMOTE directive occurring in a compilation unit has no ill effect; it is the same as using one directive.

When the DATA_REMOTE directive is used, there are two restrictions which are imposed on all local (#D) data, due to the fact that a 32-bit fullword (19-bit address) is now required to access this data:

a) Local data cannot be assigned to a 16-bit NAME variable by a NAME assignment statement, NAME INITIAL statement, %NAMECOPY, or %NAMEADD, as the target must be a 32-bit NAME REMOTE variable.

b) Local data cannot be declared as EVENT type, since SVC parameter lists only use 16-bit address fields.

An error message will be emitted if any of the above restrictions are not met. Also, the REMOTE attribute is ignored for non-NAME variables of a compilation unit with the DATA_REMOTE directive in effect.

Notes

- CONSTANTs declared in a COMPOOL as type INTEGER DOUBLE, SCALAR, and CHARACTER are actually located in the including compilation unit's #D CSECT. Therefore, if that compilation unit contains the DATA_REMOTE directive, those CONSTANTs will be treated as #D data even though they were declared in a COMPOOL. The above restrictions will be imposed on those CONSTANTs, and the CONSTANTs will be addressed in the same way that #D data is addressed.
- Data declared AUTOMATIC in a REENTRANT procedure is allocated on the runtime stack, not the #D CSECT, so the DATA_REMOTE directive has no effect on such data.

This page is intentionally left blank.

6.0 HAL/S INPUT-OUTPUT OPERATIONS

6.1 Sequential I/O

Sequential I/O is that part of the HAL/S input-output capability consisting of the READ, WRITE, and READALL statements. It is the only I/O capability available on the FC. This section will describe the types of files available for sequential I/O, the input data types expected, standard formatting performed on output, and some JCL considerations.

Note that the I/O operations described here are those performed when simulated FC execution is taking place. This form of output is intended only for program checkout purposes and is in no way meant to correspond to the actual flight computer I/O structure. In fact, all I/O operations for FSW are setup by the HAL/S-FC runtime library and actually performed by the operating system.

In certain cases, these operations may not be precisely identical to those described in the Language Specification.

6.1.1 PAGED and UNPAGED Channels

As described in the Language Specification, Section 10.1, files used for sequential I/O are considered by the HAL/S system to be in either PAGED or UNPAGED mode. The file mode determines both the effect of various I/O control functions (LINE, PAGE, etc.) and the format in which output is produced. The I/O control functions are described in Section 10.1.3 of the Language Specification, while output formats will be described below, in Section 6.1.3.

Default modes are assigned to files depending on their use: files used only as output channels are presumed to be PAGED, while those used for input to READ or READALL statements are presumed to be UNPAGED. For simple applications, these defaults are satisfactory, but in cases where a file used for output at one point in a program is to be used for input later on, some changes may be needed. Information on how to specify PAGED or UNPAGED mode for a particular I/O channel may be found in Section 5.2(1).

6.1.2 Input Data Formats

The system expects any data which serves as input to READ statements in a HAL/S program to be in the following form:

1. SCALAR and INTEGER data types may be entered in either a whole number or floating point format. The whole number representation consists of a string of decimal digits preceded by an optional - sign. Conversion to mantissa-exponent form takes place for scalar types.

   The floating-point representation is either:

   ddd.dddd

   or

E
dddd.dddd B ±dd
H

where d is a decimal digit. The decimal point may appear in any position. E,B, and H represent the exponent digits to be powers of 10, 2, and 16 respectively. There is no limit on the number of digits in the integral, fractional, or exponent parts although runtime errors may occur if the value is internally unrepresentable (see Section 8.2 for the maximum and minimum representable values). For INTEGER types, the representation is rounded to the nearest integral value. The floating-point representation may be prefixed by a + or - sign to indicate the sign of the value. In the absence of such a prefix, the value is assumed to be positive.

2. CHARACTER types are represented by a string of characters (from the HAL/S extended set) enclosed in apostrophes. The number of characters may vary between zero (a "null string") and 255. Within the string, apostrophes must be represented by an apostrophe pair.

   Note that the READALL statement may be used to allow input of character strings without requiring them to occur within apostrophes. See Language Specification, Section 10.1.1.

3. BIT types may be represented by a string of '1's and '0's enclosed in apostrophes. Embedded blanks are ignored. The number of bits may vary between one and 32. An alternate representation allows bit strings to be implicitly expressed by coding a floating-point number in the format described in (1), above. The number is rounded to the nearest integral value, and the binary representation of the resulting integer is taken.

6.1.3 Output Data Formats

Output from WRITE statements will be produced in the following form:

1. Integers - each integer, no matter what its precision, is printed in an 11-character field. The number is right justified, and leading zeros are suppressed. A minus sign precedes the number if necessary.
2. Scalars -
   - Single precision: a 14-character field is printed as follows:

     sd.ddddddE±dd

     where s is a blank or minus sign.
   - Double precision: a 23-character field is printed as follows:

     sd.ddddddddddddddddE±dd
3. Bit Strings -
   - If the output channel is PAGED, the output consists of a string of binary digits corresponding to the bit string. Leading binary zeroes are not suppressed. The field width is equal to the number of binary digits in the string plus an inserted

blank following every fourth digit (to enhance readability). This form may not be used later as input to READ statements.

- If the output channel is UNPAGED, the output consists of the string of binary digits plus inserted blanks enclosed within apostrophes. The field width is equal to the total of the number of digits, blanks and two apostrophes. This form is compatible with later use in READ statements.

4. Character Strings -
   - If the output channel is PAGED, the output merely consists of the string of characters comprising the value. The field width is equal to the number of characters in the string. This representation is not compatible with later use as input to READ statements. It is compatible with READALL, however.
   - If the output channel is UNPAGED, the string of characters is enclosed in apostrophes, and all internal apostrophes are converted to apostrophe pairs. The field width is equal to the total number of characters in the string, including added apostrophes. This form is compatible with later use as input to READ statements.

6.1.4 JCL Considerations

The device numbers used in WRITE, READ, and READALL statements are limited to the range from 2 to 9. The ddname associated with device number n (where n is within this range) is CHANNELn.

The HAL/S I/O routines make certain assumptions about the characteristics of the datasets which are defined on the various CHANNELn DD cards. The user may supply any, all, or none of the DCB attributes for each CHANNELn DD card. The following restrictions and defaults are defined:

RECFM - the record formats acceptable to HAL/S are:

F FB FA FBA FBSA

V VB VA VBA

U UA

Machine carriage control (M) is acceptable but is subject to the interpretation described below, under "carriage control".

Variable length records may be read as input, but the 4 byte descriptor field on each record will not be available to the programmer. Thus, the effective length of a variable record will be four less than the length read. Variable length record specification on output will cause records to be written in variable format. Format U records (undefined record format) on output will be written in the proper form, but all records will have the same LRECL (length).

If no RECFM is supplied, the record formats assigned are FBA (for PAGED files), and FB (for UNPAGED files).

LRECL - if not supplied by JCL or dataset attributes, the default is 133 (for PAGED files)

and 80 (for UNPAGED files).

BLKSIZE - if no blocksize is supplied in the JCL or the dataset label, then the maximum block size of the particular device is found. The BLKSIZE is set to the largest multiple of LRECL which is less than or equal to this block size. Note that for tapes, this maximum size is 32767 bytes which would require a sizable buffer area to be taken out of main storage.

Carriage Control - PAGED files with "A" specified in the RECFM cause carriage control characters to be automatically generated on output. If a PAGED file has "M" specified, no such automatic character control generation takes place, and it is the user's responsibility to ensure that the first character of each output line has the proper control character. Control characters are not permitted on UNPAGED files used for output. Carriage control characters encountered during input are available to the programmer; i.e., scanning of the input begins at the carriage control character.

6.2 File I/O

File I/O is not supported by the HAL/S-FC runtime library. If a FILE I/O statement is compiled, unresolved external references will occur at link edit time.

7.0 THE HAL/S USER CONTROL PROGRAM (HALUCP)

This section was deleted by CR13613.

This page intentionally left blank.

8.0 IMPLEMENTATION DEPENDENCIES

The HAL/S language as available on the AP-101 has several implementation dependencies. These dependencies arise due to the specific hardware and software facilities available on the machine. Certain additional facilities, not discussed in the Language Specification, are also available. This chapter describes a number of these implementation dependent features.

In particular, discussed here are the character sets available on the AP-101 and the limitations imposed by the HAL/S-FC compiler on the size of data items and the complexity of programs (Section 8.1); the specific runtime characteristics of the FC (Section 8.2); a discussion of runtime errors (Section 8.4); details about specification of access rights (Section 8.5), a list of available %macros (Section 8.7); a description of the conventions followed by the compiler in naming control sections (Section 8.9); a list of the character code conventions used in this machine (Section 8.10); and a discussion of the effect of specifying data items as REMOTE (Section 8.11).

8.1 Compile Time Characteristics

8.1.1 Character Set

The character set specified in the HAL/S Language Specification document is available in its entirety on the AP-101. The HAL/S-FC compiler will therefore recognize this full character set. The internal coding scheme for the characters is NASA DEU code. No other coding scheme is recognized. See Section 8.10 for more information on character codes.

8.1.2 Data Size Restrictions

1. Up to 74 decimal digits may appear in an arithmetic literal preceding the exponent field.
2. There is no limit to the number of exponent modifiers which may be appended to an arithmetic literal, so long as the total length of the literal does not exceed 256 characters. However, application of each exponent to the mantissa to which it belongs may not cause generation of a number outside the range 10**-78 to 10**75.
3. Bit literals may not indicate values requiring more than 32 bits.
4. Bit variables may have declared lengths of from 1 to 32 bits.
5. Character literals may not contain more than 256 characters.
6. Character variables are limited to declared lengths of 255 characters.
7. Comment brackets (/*...*/) may not enclose more than 255 characters in any one comment.
8. Arrays are limited to three dimensions.
9. Each declared dimension in an array is limited to the range 2 to 32767 and the maximum number of allowable elements is 32767.

10. The number of declared multiple copies allowed for a structure variable is limited to the range 2 to 32767 and the maximum number of allowable elements is 32767.
11. The maximum declarable dimension of a vector is 64.
12. The maximum declarable row or column dimension of a matrix is 64.
13. The maximum level of allowed subscripts (S lines) or exponents (E lines) is 11 on a single statement.
14. The valid range of error groups for user defined errors is 1 to 127. The valid range of error numbers is 0 to 62. See Appendix C for a list of system-defined errors.

8.1.3 Program Organization Limits

1. No more than 8191 user-defined symbols should be defined in one compilation unit. More symbols will be accepted by the compiler, but incorrect cross-reference information may result.
2. The number of cases in a DO CASE statement is limited to 256.
3. The depth to which code blocks may be nested is limited to 16 levels.
4. The maximum number of nested DO-END groups is 24.
5. The maximum level to which function invocations may be nested is 20.
6. No more than 32,767 literals may be defined in any one compilation unit.
7. The maximum number of labels recognized by the compiler is under the control of the "LABELSIZE=" compiler option. See Section 5.1.
8. The maximum number of characters which may be defined in all character string literals in one compilation is under the control of the "LITSTRINGS=" compiler option. See Section 5.1.
9. There is no compiler-imposed limit on the number of COMPOOL blocks appearing in a program complex except the limit implied by the number of user-defined symbols allowed (specified by the "SYMBOLS=" compiler option).
10. The maximum number of user-defined symbols allowed in one compilation is under the control of the "SYMBOLS=" compiler option. See Section 5.1.
11. The maximum number of cross-reference list entries allowed in a compilation unit is under the control of the "XREFSIZE=" compiler option. See Section 5.1.
12. The maximum number of parameters allowed in the definition of any REPLACE macro is limited to 12.
13. The maximum size of any argument used in the expansion of a REPLACE macro is limited to 250 characters.
14. The depth to which macro expansions may invoke other macros may not exceed 8 levels of nesting.
15. The maximum number of external names (ESDs) allowed in one compilation is limited to 400, of which no more than 256 may be CSECTs defined by the compilation unit.

16. The maximum number of "ON ERROR" statements which may be potentially active at any time in any compilation unit is limited to 100.
17. The number of arguments indicated in a subroutine invocation may not exceed 100. This limit includes arguments of nested function invocations; i.e., function invocations which are themselves part of a list of arguments.
18. The NONHAL attribute is not implemented.
19. Because of compiler limitations, large statements will be processed normally, but will not be optimized.

8.2 Runtime Characteristics

The hardware available on the AP-101, as well as the design of operating system support software, determine the kind of storage available at run-time for various data types, and influence the way conversions between data types are made. These characteristics are described below.

1. Single precision integers are represented as l6-bit signed quantities, and hence INTEGER SINGLE variables cannot take on values outside the range -32,768 to 32,767.
2. Double precision integers are represented as 32-bit signed quantities, and hence INTEGER DOUBLE variables cannot take on values outside the range -2,147,483,648 to 2,147,483,647.
3. Single precision scalar values are represented internally in the standard AP-101 floating point format using 32 bits (1 sign, 7 exponent, 24 mantissa). SCALAR values outside the range $\pm$10**-78 to $\pm$10**75 are therefore unrepresentable. 7 significant digits are maintained.
4. Double precision scalar values are represented internally in the standard AP-101 double precision format using 64 bits (1 sign, 7 exponent, 56 mantissa). The available range for double precision scalars is thus essentially the same as that for single precision scalars, but 16 significant digits are maintained.
5. The dynamic size of character variables may not exceed 255.
6. The length of bit variables must be in the range 1-32.
7. Runtime conversion of double precision scalar values to single precision scalar values is performed by truncating the right-most 32 bits from the double precision mantissa.
8. Runtime conversion of double precision integer values to single precision integer values is performed by eliminating the left-most 16 bits of the double precision value.
9. Runtime conversion of integer values to scalar values is performed by converting the integer value to a double precision scalar value retaining all significant digits. If the final result of the conversion is to be single precision scalar, the standard double-to-single precision scalar conversion is then applied to the intermediate scalar value.
10. Runtime conversion from scalar to integer results in an error condition if the scalar value cannot be represented in the integer form.

11. Runtime conversion from single precision integer to double precision integer is performed by propagating the sign bit of the single precision value throughout the 16 high order bit positions of the double precision value.
12. Runtime conversion of single precision scalar to double precision scalar is performed by padding 32 zero bits to the right of the single precision mantissa.
13. Runtime conversion of double precision scalar to character produces a 23 character field with two exponent digits and 17 fractional digits. A double precision scalar literal converts to a single precision character unless the precision of the literal is in some way specified (e.g. with a CONSTANT declare or with a shaping function). See related user's note 17. See Section 6.1.3 for format.
14. Runtime conversion of single precision scalar to character produces a 14 character field with two exponent digits and 8 fractional digits. See Section 6.1.3 for format.
15. Runtime conversion of a single or double precision integer to a character string results in a variable length field of up to 11 characters. See Section 6.1.3 for format.
16. Runtime conversion from a character string to an integer or scalar can take place only if the character string is in one of the standard input formats for scalar or integer data types. See Section 6.1.2 for details.
17. The DATE built-in function returns a double precision integer whose decimal value is YYDDD where YY are the year and DDD represents the day of the year (i.e., February 1, 1978=78032).
18. The RUNTIME and CLOCKTIME built-in function returns a double precision scalar. RUNTIME returns the simulated elapsed time.

8.3 Real-Time Statements

This section was deleted by CR13613.

8.4 Runtime Errors

Each HAL/S runtime error is given a mnemonic name, consisting of an error group number and an individual error number within that group. This is true both of system-defined errors and of those which are user-defined (see Chapter 9 of the Language Specification for more information about error definitions). A complete list of system-defined errors may be found in Appendix C.

The response to an error condition encountered at runtime is also either under user or system control. If an "ON ERROR..." statement for that error appears in the program, the clause following ON ERROR determines the action. If this clause is "SYSTEM", or if no such statement occurs for a particular system-defined error, then a standard action is taken. These various default system actions are also listed in Appendix C. Otherwise the clause after ON ERROR defines some specific action to be taken in lieu of the standard one. This may be some HAL/S statement to be executed, or else may be the keyword "IGNORE". Thus three types of action may occur - "SYSTEM action",

"statement action", and "IGNORE action".

8.5 ACCESS Rights

As described in the Language Specification, it is possible to place managerial restrictions on variables declared in COMPOOLS and on external code blocks by means of the ACCESS attribute. The HAL/S-FC compiler enforces such restrictions in the following manner.

Any COMPOOL variable, and any external routine, to which the ACCESS attribute has been applied is considered to be restricted for the compilation unit which is being compiled. The restriction is slightly different for variables than for blocks:

1. Variables with the ACCESS attribute may not have their values changed (but may, e.g., be referenced).
2. Block names may not be used at all.

These restrictions may be selectively overridden for individual variable and block names. The list of these ACCESS controlled names which are to be made available to the unit being compiled is provided to the compiler via an external dataset. This external dataset is known as the Program Access File (PAF). The PAF must have partitioned organization and is specified by a JCL card provided to the compilation step in the following format:

```
//HAL.ACCESS DD DSN=<PAF name>,<other parameters>
```

where the <PAF name> is the dataset name of the PAF without any member specification. This card should follow the EXEC card which invokes the compiler. Note that a PAF need not be defined if the access control facilities of the HAL/S compiler are not being used.

Each member of the PAF contains information about ACCESS controlled names which are to be made available to one unit of compilation. The member to be used is specified to the compiler by using the PROGRAM compiler directive in the primary input stream:

```
D PROGRAM ID=<id>
```

The <id> field of the directive is a 1 to 8 character Program Identification Name (PIN) which is the name of the member of the PAF which is to be processed for the current compilation's ACCESS information. The appearance of the PROGRAM directive in the compiler's input stream causes immediate processing of the PAF member specified. Therefore, the PROGRAM directive must follow any COMPOOL or COMSUB templates which may specify ACCESS-controlled data. Also, the PROGRAM directive must appear before the definition of the block which is the primary unit of compilation. In general, the input stream seen by the compiler should look like the following:

```
<COMPOOL or COMSUB template>
  .
  .
  .
```

```
  <COMPOOL or COMSUB template>
D PROGRAM=<id>
M PROG_NAME: PROGRAM;
    .
    .
    .
```

The format of an individual PAF member is specified below.

1. Column 1 of each record is ignored except when it contains the character "C", in which case the entire record is ignored.
2. The portion of each record which is processed is the same portion which is processed in the primary compiler input (generally columns 2-80).
3. COMPOOL elements which are to be made available to the compilation are specified as:

   ```
   <COMPOOL-name>(<var-name>,<var-name>,...,<var-name>)
                 or
   <COMPOOL-name>($ALL)
   ```

   The first format specifies access to particular variables within the named COMPOOL. The second format specifies access to all variables within the named COMPOOL.
4. Access to external block names is specified as:

   ```
   $BLOCK(<block-name>,<block-name>,...,<block-name>)
   ```
5. Blanks are allowed anywhere in the record, except that names may not be broken by a blank.
6. Either of the constructions (3) or (4) above may span more than one record.
7. The name of the particular COMPOOL in form (3) above may appear more than once; i.e., the variables in a particular COMPOOL do not have to be specified at one time. Similarly, the form $BLOCK may appear more than once.

Some validity checking is performed by the compiler while processing the PAF member. Warnings are issued for the following conditions:

1. A syntax error on a PAF record - the bad record is printed.
2. Names mentioned in the PAF have not yet been defined (possibly caused by faulty positioning of the PROGRAM directive).
3. Elements of $BLOCK in the PAF are not block names.
4. Requests are made for names which are not ACCESS protected.
5. The variables named have been found, but not within the COMPOOL specified.
6. The <COMPOOL-name> given is not the name of a COMPOOL.
7. Elements of an ACCESS-protected COMPOOL block are mentioned in the PAF before the COMPOOL block itself is freed for use.

If, at the time the PROGRAM directive is encountered, no ACCESS-controlled variables have been declared, the PAF is not opened. If a user does not require access to any ACCESS-controlled variable, the PROGRAM directive and associated PAF members may of course be omitted.

8.6 Language Subset Restriction Capability

A language subsetting facility exists in the HAL/S AP-101 system whereby specific constructs in the HAL/S language may be defined as illegal. If a HAL/S program using such a construct is compiled, Phase 1 of the compiler will emit a warning message.

Constructs which may be so restricted fall into two classes:

1. Specific productions in the HAL/S grammar (which is shown in Backus-Naur notation in Appendix G of the HAL/S Language Specification);
2. Specific built-in functions from the set defined in Appendix C of the HAL/S Language Specification.

The compiler is informed which productions and built-in functions are to be illegal by having this information provided in a special member (named $$SUBSET) of the Program Access File described in Section 8.5. If the subsetting capability is to be used, therefore, an ACCESS DD card must be provided in the user's JCL as described above.

The contents of $$SUBSET specify which productions and built-in functions are to be illegal. The following rules govern the format of this file:

1. Line 1 is a title line which is transferred by the compiler in its entirety to the formatted listing output. Its contents are arbitrary.
2. Any subsequent line beginning with C is considered a comment line and is ignored; otherwise, it is scanned.
3. On scanned lines, blanks are ignored.
4. Text can be continued arbitrarily from line to line.

5. The syntax for defining illegal productions is:

   ```
   $PRODUCTIONS(<number>,<number>,...)
   ```

   where <number> is an integer whose value lies in the range 0 through the number of productions in the grammar. See Appendix G of the HAL/S Language Specification for the current working grammar.
6. The syntax for defining illegal built-in functions is:

   ```
   $BUILTINS(<name>,<name>,...)
   ```

   where <name> is the name of a built-in function.
7. Either construct can appear any number of times, and in any order.

Example:

A $$SUBSET with the contents shown below will make illegal all real-time constructs, including UPDATE blocks, and any associated built-in functions:

```
  HAL/S-360 WITH NO REALTIME CONSTRUCTS
C THE FOLLOWING PRODUCTIONS ARE ILLEGAL
  $PRODUCTIONS(67,68,71,74,201,219,313,315,371,
               381,433,434,437,438)
C THE FOLLOWING BUILTINS ARE ILLEGAL
  $BUILTINS(PRIO,NEXTIME,RUNTIME)
```

Note that if, e.g., 438 is specified, it is not necessary to specify 439-441, since (as Appendix G of the HAL/S Language Specification shows) these latter define <SCHEDULE HEAD> recursively and ultimately rest on the non-recursive definition of 438.

An additional useful capability exists. It is possible to specify in $$SUBSET a maximum length which bit strings (literals or variables) may take on any compilation. The syntax for this is:

```
$BITLENGTH <n>
```

where <n> is the maximum length bit string allowable and must be less than or equal to 32 (the maximum allowable on the host computer).

At the beginning of execution, Phase 1 of the compiler determines which productions and/or built-in functions are to be made illegal by scanning member $$SUBSET of the dataset defined on the ACCESS DD card. If no such card was specified, or if member $$SUBSET for any reason cannot be accessed, no language restrictions are taken to be in force. Phase 1 of the HAL/S compiler emits the message:

```
*** NO LANGUAGE SUBSET IN EFFECT ***
```

If member $$SUBSET is successfully accessed, Phase 1 emits the message:

```
*** LANGUAGE SUBSET IN EFFECT: <title line>
```

During the subsequent compilation of the user HAL/S program, if an illegal production or built-in function is encountered, Phase 1 emits one of the following severity 1 error messages:

```
*** ERROR XS1: THE FOLLOWING FUNCTION IS NOT IMPLEMENTED IN
THE CURRENT LANGUAGE SUBSET: <name>
*** ERROR XS2: THE ABOVE CONSTRUCT IS NOT IMPLEMENTED IN
THE CURRENT LANGUAGE SUBSET.(VIOLATION AT PRODUCTION nn)
```

During the scanning of the member $$SUBSET by Phase 1, several errors may be detected. These are flagged with the generic message:

```
1*** SUBSET ACQUISITION ERROR -
```

and one of the following specific messages:

```
1. PREMATURE EOF
2. BAD SYNTAX:<line of bad text>
3. UNKNOWN FUNCTION:<function name>
4. UNKNOWN PRODUCTION:<production number>
```

8.7 %MACRO Implementation

The following %MACROs are recognized by the HAL/S-FC compiler and generate the indicated code.

%SVC. The %SVC statement generates a true SVC instruction in which the address portion points to the operand specified.

Operation	Code
%SVC(a);	SVC a
	.
	.
a	DC <data>

User note: A character string is not a valid argument for the %SVC statement, which requires an address of an operand. A statement like %SVC('A') results in an 0C1 abend of the compiler (DR 58890, 5/9/88).

%NAMECOPY. This operation works in the same manner as assignment of NAME variables except that the operands must be structures. The first operand, the receiver, must be the NAME of a structure. The second operand, the source, must be a structure or the NAME of a structure.

The address to be stored into the receiving NAME variable is adjusted such that later, normal use of that operand will be successful. This occurs when structures with copies are involved. The address maintained for a multi-copy structure points to a fictitious "0th" element so that HAL/S subscripting which starts at one can be done efficiently. Adjustment for the various combinations of structures with and without copies and of varying copies is handled automatically by the compiler.

Operation	Code
%NAMECOPY(NST1,ST);	LA R,ST STH R,NST1
%NAMECOPY(NST1,NST2);	LH R,NST2 STH R,NST1
%NAMECOPY(RNST1,RST);	L R,ZCON → RST ST R,RNST1
%NAMECOPY(RNST1,RNST2);	L R,RNST2 ST R,RNST1
%NAMECOPY(RNST1,ST);	LA R,ST OHI R,x'8000' (PASS Only) IAL R,x'0000'(linker fills in sector number) ST R,RNST1
%NAMECOPY(RNST1,NST2);	LH R,NST2 SRA R,1 SRR R,31 OHI R,x'8000' (PASS Only) ST R,RNST1

where:

NST1, NST2	is a NAME Structure
RNST1, RNST2	is a NAME REMOTE Structure
ST	is a locally declared Structure
RST	is a REMOTE Structure

A REMOTE source cannot be copied into a non-REMOTE destination.

The programmer may use %NAMECOPY in a way that bypasses the type checking protection provided by the HAL/S language. See the HIGHOPT option in section 5.1 for a detailed explanation.

%COPY. The %COPY statement can be used to move data from one location to another without regard to data types. The general form is:

```
%COPY(dest,source,count);
```

where: source is the variable name from which data will be moved; dest is the variable name into which data will be moved; and count is optional and if present indicates the number of halfwords to be moved from source to dest. If count is omitted, the size of the source operand is used to determine a count. The count can be a literal or a variable.

NAME variables are dereferenced in all cases. Use of a NAME variable as dest or source operand will refer to the data pointed to by the NAME variable. The count may also be a NAME variable in which case the count is taken from the storage location pointed to by the NAME variable.

The move halfword instruction (MVH) is used to implement the %COPY statement when feasible. If possible, the compiler will optimize the %COPY sequence by performing LED, STED, L, ST or LH, STH sequences instead of using the MVH instruction. This optimization occurs when the compiler is able to determine that no more than eight

halfwords need to be moved and that the data alignments match for both source and receiving data areas. When the destination operand of a %COPY statement is REMOTE, a call to MSTR is used to implement the move. Since MSTR expects ZCON inputs, the compiler will perform a YCON to ZCON conversion if the source operand is LOCAL data. The following examples show some of the possible sequences.

Operation	**Code**	
%COPY(A,B,12)	L	R,=YCON(A,12)
	L	R2,=ZCON(B,DSR)
	MVH	R,R2
%COPY(A,B,7);	LED	FR,B
A, B fullword aligned	STED	FR,A
	L	R,B+4
	ST	R,A+4
	LH	R,B+6
	STH	R,A+6
%COPY(C,D,7);	LH	R,D
C,D halfword Aligned	STH	R,C
	LED	FR,D+1
	STED	FR,C+1
	L	R,D+5
	ST	R,C+5
%COPY(E,F,2);	LH	R,F
E,F with non-matching alignments	STH	R,E
	LH	R,F+1
	STH	R,E+1
%COPY(G,H);	L	R4,ZCON -> H
where G and H are REMOTE	LFXI	R5,Length
	L	R2,ZCON -> G
	SCAL@#	MSTR
%COPY(I,J);	LA	R4,J
where I is REMOTE and J is local	OHI	R4,"8000" (PASS only)
	IAL	R4,"0000"
	LFXI	R5,Length
	L	R2,ZCON -> I
	SCAL@#	MSTR

Warning: Neither the source nor receiving data will be checked for LOCK violations outside of an UPDATE block. However, if %COPY does appear within an UPDATE block, the LOCK groups of the variables named will be used.

User Notes for %COPY

The compiler emits the severity 2 error FN106 ('Element or CSECT boundary exceeded for source of %COPY') when the source argument of the %COPY is an element of an

indefinite array (i.e. declared with arrayness (*); legal only in procedures and functions). For example, the following test case gets error FN106:

```
DR46624: PROCEDURE(SOURCE_ARRAY);
            DECLARE SOURCE_ARRAY ARRAY(*) INTEGER;
            DECLARE DESTINATION INTEGER;
           %COPY(DESTINATION,SOURCE_ARRAY$(1),1);
CLOSE DR46624;
```

(DR 46624, 5/9/88).

%COPY statements with variable subscripts will not get the warning FN107 ('Source/Destination of %COPY involves runtime addressing') if the source/destination argument is not local, i.e. it is declared in an included member. Otherwise, %COPY statements with variable subscripts get the warning (DR 50430, 5/9/88).

If an argument to a %COPY statement is an ARRAY terminal of a multi-copy structure, then the error message "ZC1 -THE COMPILER CANNOT HANDLE THE MULTI-COPIED STRUCTURE REFERENCE BECAUSE OF IMPROPER SUBSCRIPTING OR A NON-CONTIGUOUS REGION SPECIFICATION" will be emitted by the compiler. In the following HAL/S program, the statements containing the %COPY are flagged by a "ZC1" error message (DR54123, 3/9/85).

```
DR54123: PROGRAM;
     DECLARE   I INTEGER;
     STRUCTURE S RIGID:
             1 A ARRAY(3) INTEGER,
             1 B ARRAY(3) BIT(16);
    DECLARE S1 S-STRUCTURE(3);
    %COPY(I,S1.A$(2),1);
    %COPY(I,S1.A,1);
CLOSE DR54123;
```

The ZC1 error will also be emitted when an argument of the %COPY is a NAME node of a multi-copied structure which has an asterisk (*) as the structure subscript. (DR111337, 06/02/99)

An XQ101 error is generated for the third argument (i.e., count) of a %COPY if this argument is a function return of a user-defined integer (single/double) function with no parameters (DR120271, 08/02/05).

The programmer may use %COPY in a way that bypasses the type checking protection provided by the HAL/S language. See the HIGHOPT option in section 5.1 for a detailed explanation.

Warning: Programmers should be aware that when using the %COPY macro to clear a memory range (e.g. a buffer), since the MVH instruction moves data from last to first, the highest indexed items involved should be initialized. And due to a microcode bug, the

source block should reside at least 2 HWs higher than the destination block and so the last 2 HWs of the source block should be initialized.

Reference the AP-101S Principles of Operation for further details about the Move Halfword instruction.

%NAMEADD. The %NAMEADD statement allows address arithmetic to be performed on NAME variables. The allowable forms are:

```
%NAMEADD(N1,V,int);
%NAMEADD(N1,N2,int);
%NAMEADD(RN1,RV,int);
%NAMEADD(RN1,RN2,int);
%NAMEADD(RN1,V,int);
%NAMEADD(RN1,N2,int);
```

where:

N1,N2	is any NAME variable.
RN1,RN2	is any NAME REMOTE variable.
V	is any HAL/S variable which is legal in the context NAME(V).
RV	is any HAL/S REMOTE variable which is legal in the context NAME(RV).
Int	is either an integer literal or variable which specifies the amount to be added (which may be negative) to the second operand (Note: literals must not be signed).

The source cannot be REMOTE if the destination is non-REMOTE.

The effect is the same as if the user specified the (quite illegal) HAL/S statements:

```
NAME(N1) = NAME(V) + int;
     or
NAME(N1) = NAME(N2) + int;
```

Operation	Code
%NAMEADD(N1,V,5);	LA R,V+5 STH R,N1
%NAMEADD(N1,V,I);	LA R,V AH R,I STH R,N1
%NAMEADD(N1,N2,8);	LH R,N2 LA R,8(R) STH R,N1
%NAMEADD(RN1,RV,8);	L R,ZCON -> RV RN2 R,8 ST R,RN1
%NAMEADD(RN1,RN2,8);	L R,RN2 AHI R,8 ST R,RN1

```
%NAMEADD(RN1,V,5);        LA   R,V+5
                          OHI  R,x'8000' (PASS Only)
                          IAL  R,x'0800' OR '0000'
                               (linker fills in sector number)
                          ST   R,RN1

%NAMEADD(RN1,N2,8);       LH   R,N2
                          LA   R,8(R)
                          IAL  R,x'0800'
                               (non-aggregate variables only)
                          SRA  R,1
                          SRR  R,31
                          OHI  R,x'8000'  (PASS only)
                          ST   R,RN1
```

User Notes for %NAMEADD

It is possible to mix NAME variables of different types as the operands of %NAMEADD. For example: a NAME SCALAR can be mixed with NAME CHARACTER. This feature can be used to "convert" a NAME variable from one type to another. Mixing NAME types in a %NAMEADD should be used with great care. The type of the resulting NAME variable remains its declared type rather than the type of the data it now points to.

For example:

```
DECLARE CH CHARACTER;
DECLARE NSC NAME SCALAR;
DECLARE NCH NAME CHARACTER INITIAL(NAME(CH));
%NAMEADD(NSC,NCH,0);
```

Here, NSC will point to the CHARACTER, CH, even though it is a NAME SCALAR. However, NSC should still be thought of as a NAME SCALAR in subsequent uses.

An XQ101 error is generated for the third argument (i.e., count) of a %NAMEADD if this argument is a function return of a user-defined integer (single/double) function with no parameters. (DR120271, 08/02/05).

The programmer may use %NAMEADD in a way that bypasses the type checking protection provided by the HAL/S language. See the HIGHOPT option in section 5.1 for a detailed explanation.

%NAMEBIAS. By convention, the compiler uses an address for aggregate data that is offset a certain number of halfwords (depending on data type) before the actual beginning of the data (see Section 7.8). The %NAMEBIAS statement performs this zeroth element calculation by determining the positive value needed to point to the first element of the data. It consists of two operands as seen below:

%NAMEBIAS(X,Y)

where:

X is a destination variable of integer type

Y is a source vairable of any data type (unsubscripted)

The result of the %NAMEBIAS statement is placed into the variable X. Please note that the offset of a NAME variable is the offset of the variable to which it points.

Operation	Code	
%NAMEBIAS(X,Y)	LFXI/LHI/L	Ry,zeroth element offset of Y
where X is a non-NAME variable	STH/ST	Ry,X
%NAMEBIAS(X,Y)	LFXI/LHI/L	Ry,zeroth element offset of Y
where X is a NAME variable	LH	Rx,X
	STH/ST	Ry,disp(Rx)
%NAMEBIAS(X,Y)	LFXI/LHI/L	Ry,zeroth element offset of Y
where X is REMOTE	STH@#/ST@#	Ry,X
%NAMEBIAS(X,Y)	ZH/ZH@#	Rx
where Y has zero offset		

User Notes for %NAMEBIAS

The following examples illustrate the use of %NAMEBIAS with %NAMEADD. Note that subscripting the source of the %NAMEADD statement down to the element level eliminates the need for zeroth element calculations since the starting address is already specified. Due to this, most assignments where the source and destination are of the same type can be performed with a NAME assignment with subscripts instead of a %NAMEADD statement. Also, %NAMEBIAS works strictly on data type and arrayness, so STT2.B22 below will give the same offset as AB22.

```
STRUCTURE T RIGID:
 1 B2 ARRAY(2) BIT(16),
 1 B22 ARRAY(2,2) BIT(16);

STRUCTURE X RIGID:
 1 B BIT(16),
 1 B3 ARRAY(3) BIT(16);
DECLARE OFF1 INTEGER;
DECLARE OFF2 INTEGER;
DECLARE NB NAME BIT(16);
DECLARE NI NAME INTEGER;
DECLARE AB22 ARRAY(2,2) BIT(16) INITIAL(BIN'110001',BIN'110010',BIN'110011',
    BIN '110100');
DECLARE NAB22 NAME ARRAY(2,2) BIT(16) INITIAL(NAME(AB22));
DECLARE STT2 T-STRUCTURE(2) INITIAL(BIN'0001',BIN'0010',BIN'0011',BIN'0100',
    BIN'0101',BIN'0110',BIN'0111',BIN'1000',BIN'1001',BIN'1010',BIN'1011',BIN'1100');
```

```
DECLARE NSTX2 NAME X-STRUCTURE(2);

%NAMEBIAS(OFF1,NAB22);                  ← offset of 3 is saved into OFF1
%NAMEADD(NI,NAB22,OFF1);                ← NI=49 (1st element of AB22)
%NAMEADD(NI,NAB22$(1,1:),OFF1);         ← NI=52 (3 halfwords added to address of 1st
                                          element)
%NAMEADD(NI,NAB22$(1,1:),0);            ← NI=49 (since NAB22 was subscripted
                                          down to the element level, no offset was
                                          required for NI to point to the 1st element of
                                          NAB22)

%NAMEBIAS(OFF1,STT2);                   ← offset of 6 is saved into OFF1
%NAMEADD(NB,STT2,OFF1);                 ← NB='0001' (1st element in structure)

%NAMEBIAS(OFF1,STT2);                   ← offset of 6 is saved into OFF1
%NAMEADD(NB,STT2$(1;),OFF1);            ← NB='0111' (1st element in 2nd copy)

%NAMEBIAS(OFF1,STT2.B22);               ← offset of 3 is saved into OFF1
%NAMEADD(NB,STT2.B22$(1;),OFF1);        ← NB='0011' (1st element of STT2.B22)

/*IF THE DESTINATION NAME VARIABLE POINTS TO A VARIABLE WITH AN     */
/*OFFSET, THEN SUBTRACT THAT OFFSET FROM THE STARTING ADDRESS */
%NAMEBIAS(OFF1,AB22);                   ← offset of 3 is saved into OFF1
OFF1 = -OFF1;
%NAMEADD(NAB22,STT2.B2$(1;2:),OFF1);    ← NAB22 contains 4 BIT(16) values
                                          starting with the 2nd element of
                                          STT2.B2='0010', '0011', '0100', '0101'

/*IF THE SOURCE ALSO HAS AN OFFSET, THEN THAT OFFSET MUST BE         */
/*ADDED AND THE OFFSET OF THE DESTINATION SUBTRACTED.                */
%NAMEBIAS(OFF1,STT2);                   ← offset of 6 is saved into OFF1
%NAMEBIAS(OFF2,NSTX2);                  ← offset of 4 is saved into OFF2
OFF1 = OFF1 - OFF2;
%NAMEADD(NSTX2,STT2,OFF1);              ← NSTX2 now points to the start of STT2 so
                                          NSTX2.B$(1;)='0001' and
                                          NSTX2.B$(2;)='0101'

/*IF THE SOURCE AND DESTINATION ARE OF THE SAME TYPE AND ARRAYNESS, */
/*THEN THE OFFSETS CANCEL.  A NAME ASSIGNMENT SHOULD BE USED         */
/*INSTEAD - NAME(NAB22)=NAME(STT2.B22$(1;)).                         */
%NAMEADD(NAB22,STT2.B22$(1;),0);        ← NAB22='0011', '0100', '0101', '0110'
```

8.8 Arrayed Addressing 0th Element and % NAMEADD

The HAL/S-FC compiler arranges data in memory such that the least number of base registers need be dedicated in addressing. Storage assignments are made with the required base-displacement combinations being generated to properly access the data. The storage addresses assigned refer to the actual data beginning, but for arrayed data types, the base-displacement address includes a negative offset value. This negative

offset value is commonly referred to as an imaginary *0th element*. This value can be computed with %NAMEBIAS.

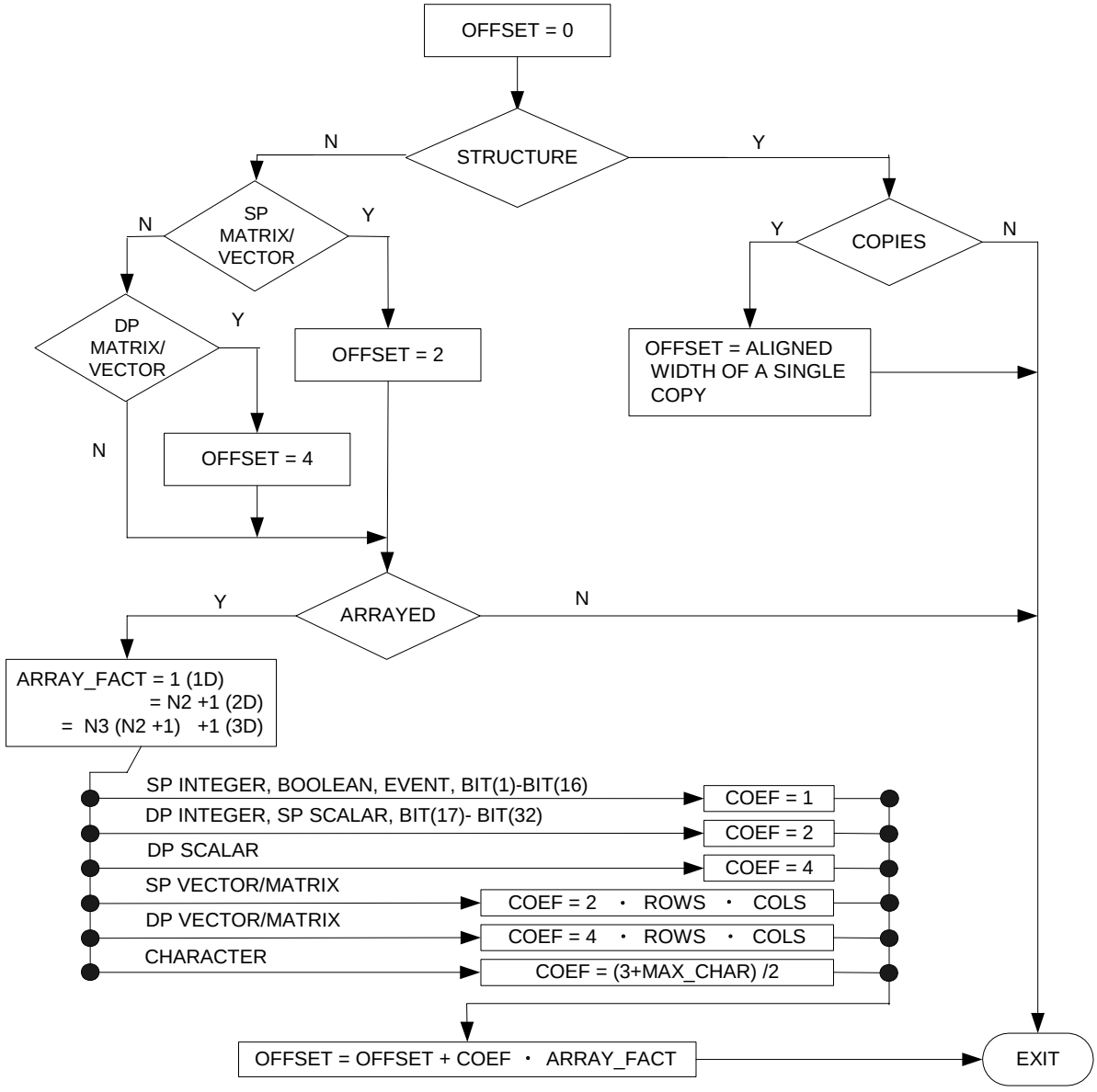

Figure 8-1 Algorithm for Calculating the 0th Element Offset

Example:

```
DECLARE  A SCALAR,
         B INTEGER,
         C CHARACTER(7),
         D ARRAY(5) DOUBLE;
DECLARE  E ARRAY(5),
```

```
          F ARRAY(3,3) VECTOR,
          G MATRIX;
DECLARE   H DOUBLE,
          I ARRAY(5,5) INTEGER;
```

Alignment	Name	Location	Base	Displacement	(In Decimal) 0th element
Halfword	B	00002	1	0002	0
Fullword	A	00004	1	0004	0
Doubleword	H	00008	1	0008	0
Halfword	C	0000C	1	000C	0
Halfword	I	00011	1	000B	-6
Fullword	E	0002A	1	0028	-2
Doubleword	D	00034	1	0030	-4
Fullword	G	00048	1	0046	-2
Fullword	F	0005A	1	0040	-26

Ordinarily, the user does not need to be concerned with the addressing mechanism, or the 0^{th} element, as the compiler handles it automatically. One exception is in the use of %NAMEADD. If the user uses the %NAMEADD macro to manually create an address for an arrayed type NAME variable, the 0^{th} element calculation must be taken into consideration. In the following example, the desire is to treat the three integers in the structure as an array by making a NAME variable point to them.

```
STRUCTURE T RIGID:
1 I1 INTEGER,
1 I2 INTEGER,
1 I3 INTEGER,
1 A ARRAY(3) INTEGER;
DECLARE ST T-STRUCTURE INITIAL(1,2,3,44,55,66);
DECLARE NA NAME ARRAY(3) INTEGER;
DECLARE ZEROTH INTEGER;
ZEROTH = -1;
%NAMEADD(NA, ST, ZEROTH);
```

NA can now be referenced as an ARRAY pointer, pointing to an ARRAY containing (1,2,3). If the %NAMEADD were to ignore the 0^{th} element and point directly to the data, NA would point to one element past where it should be, and the 1^{st} element could not be referenced at all - instead the ARRAY pointed to would have the values (2,3,44).

8.9 CSECT Naming Conventions

Each successful HAL/S compilation produces several named control sections (CSECTs). The CSECT names are derived as follows. Only the first six characters of a compilation unit name are used; furthermore, any occurrence of the underscore character (_) in the

first six characters of a compilation unit name is eliminated. The resulting characters are joined together to produce the characteristic name of the compilation unit (e.g. A_B_C becomes ABC). The name is then padded or truncated to six characters where necessary. Care should be exercised in naming compilation units to ensure that only unique names will be generated for a given program complex.

Additional characters are placed on the front of the characteristic name to form the final name for each of the types of CSECTS generated by the compilation. All CSECT names therefore take the form:

```
ccNNNNNN
```

where the value of cc for individual cases is:

```
PROGRAM            :    $0
TASKs              :    $a a=(0-9, A-Z)
COMSUBs            :    #C
Internal procs     :    an a=(A-Z),n=(0-9)
Library            :    aa a=(A-Z)
STACK              :    @a where a=(0-9,A-Z)
DECLAREd data      :    #D
COMPOOL            :    #P
Process            :    #E
directory
entry(PDE)
Remote data        :    #R
ZCON for COMSUBs   :    #Z
ZCON for Library   :    #Q
EXCLUSIVE flag     :    #X
Data for Library   :    #L
```

If the SDL option is not specified, a separate CSECT is generated for each compilation which is a PROGRAM. Its contents in AP-101 Assembler language are as follows (the name of the HAL/S program in this example is HALPROG):

```
START CSECT
      EXTRN     $0HALPRO
      BAL       4,$0HALPRO
      END       START
```

Linkage editor STACK control cards are issued to force static stack allocation for stand-alone operation.

In addition to control section names, the characteristic name of a compilation unit may appear in external contexts preceded by the following characters:

@@ - the member name of the template created for a compilation unit

- the member name of the simulation data file created for a compilation unit

See Sections 3.4 and 3.5.

8.10 Character Code Conventions

The HAL/S language defines a set of standard, printable characters which are used to create HAL/S programs. In particular, these characters are used to create the HAL/S primitives known as "character literals".

The AP-101, however, includes in its character set several special characters which can be used for display and control purposes but which do not form part of the standard HAL/S character set. Such special characters are furthermore, in general, not available on the 360, where HAL/S-FC programs are written and compiled.

The HAL/S language, as described in the Language Specification (Section 2.5), permits the use of a special "escape" character (¢) to indicate within a character literal that some special character is required. The following two tables show how this feature is implemented in the HAL/S-FC system.

Table 8-1 shows the various characters acceptable within character literals in HAL/S source code.

For each character (or pair of characters, if an escape character is used) the actual internal hex representation of that character is shown. Thus for instance if the pair '¢E' appears in source code to be compiled by the FC compiler, its internal representation will be the hex numeral 14.

Table 8-1 should be used in conjunction with Table 8-2, which contains the actual character set available on the AP-101 and the internal representations associated with these characters. Here the hexadecimal numeral 14 turns out to stand for the character epsilon. The result is that '¢E' appearing in a character literal within a HAL/S program would be interpreted by the FC compiler as the AP-101 character epsilon. WRITE(6) '¢E<0', for example, would, if executed on the AP-101, cause "ε<0" to be displayed on the appropriate output device.

First HEX Digit

Second HEX Digit	0	1	2	3	4	5	6	7
0	¢0	¢A	blank	0	¢G	P	¢Y	p
1	]	¢B	!	1	A	Q	a	q
2	[	¢R	¢Q	2	B	R	b	r
3	¢7	¢W	#	3	C	S	c	s
4	¢8	¢E	¢a	4	D	T	d	t
5	¢9	¢O	%	5	E	U	e	u
6	¢V	__	&	6	F	V	f	v
7	¢.	¢I	‘	7	G	W	g	w
8	¢__	¢J	(	8	H	X	h	x
9	¢/	¢K	)	9	I	Y	i	y
A	¢4	¢M	*	:	J	Z	j	z
B	¢>	¢N	+	;	K	¢S	k	¢C
C	¢<	¢1	,	<	L	¢T	l	¢U
D	¢	¢2	-	=	M	¢Z	m	¢X
E	¢5	¢3	.	>	N	¢P	n	¢L
F	¢6	¢4	/	?	O	¢F	o	¢D

Table 8-1 HAL/S-FC CONVENTION FOR CHARACTER INPUT

First HEX Digit

Second HEX Digit	0	1	2	3	4	5	6	7
0	NULL	α	SPACE	⊙	ϒ	P	ψ	p
1	]	β	!	1	A	Q	a	q
2	[	ρ	~	2	B	R	b	r
3	SELF TEST	ω	#	3	C	S	c	s
4	•	ε	√	4	D	T	d	t
5	••	Ω	%	5	E	U	e	u
6	∇	_	&	6	F	V	f	v
7	•	—	‘	7	G	W	g	w
8	BACKSPACE	|	(	8	H	X	h	x
9	÷	◊	)	9	I	Y	i	y
A	⇧	□	*	:	J	Z	j	z
B	⊳	•	+	;	K	Σ	k	σ
C	⊲	↑	,	<	L	θ	l	|
D	CARRIAGE RETURN	↓	-	=	M	‾ OVERSCORE	m	UNDERSCORE
E	⇦	→	.	>	N	π	n	λ
F	⇗	←	/	?	O	ϕ	o	Δ

Table 8-2 DEU Character Set

8.11 Remote Data

REMOTE data may be specified three ways:

1) `D INCLUDE TEMPLATE CPL REMOTE`

which causes all data in the COMPOOL template CPL to be addressed via ZCON's, a less efficient method than non-REMOTE addressing. If all compilation units including CPL specify REMOTE, then it is not necessary for that COMPOOL to be placed in the standard data area (either sector 0 or 1), since no direct addressing will be generated for accessing CPL data.

Specification of a single-value REMOTE data item will take up space for the data plus a two halfword ZCON plus a long format (RX) AP-101 instruction for each data access. See section 5.2, item 2 for more information on the INCLUDE directive.

2) `DECLARE A ARRAY(5) REMOTE`

which causes the array A to be located in a REMOTE (#R) CSECT, and can be placed in a sector other than 0 or 1. As above, the data will use ZCON addressing. This type of REMOTE specification is restricted when the SDL option is on (except on NAME variable and formal parameter declarations, in which case the REMOTE attribute signifies that the data pointed to or passed in is REMOTE).

3) `D DATA_REMOTE`

which allows all locally declared data in a PROGRAM or COMSUB to be placed in a sector other than 0 or 1. The addressing performed for this method is more efficient than ZCON addressing. See section 5.2, item 10 for more information on the DATA_REMOTE directive.

All HAL/S data types except EVENT can be specified as REMOTE. An attempt to specify REMOTE EVENT data will generate a DA9 error.

Note that the construction:

```
DECLARE N NAME ARRAY (5) REMOTE;
```

does not cause the name variable N to be placed in the REMOTE CSECT. Instead, N becomes a 32-bit ZCON rather than the standard 16-bit YCON. It is possible to initialize N with the NAME of any REMOTE addressable data item either via:

```
INITIAL (NAME (A));
      or
NAME (N) = NAME (A);
```

A NAME REMOTE (32-bit) variable cannot be assigned to a local (16-bit) NAME variable.

The following sections contain information on known restrictions in the use of REMOTE data that are not documented elsewhere.

8.11.1 NAME Variable Initialization Restriction

A regular (16-bit) NAME variable cannot be initialized at declaration time with a REMOTE address or a NAME REMOTE (32-bit) variable. This includes the REMOTE addresses of variables located in REMOTEly included compools. Any attempt to do this will result in the DI21 or DI107 error message.

8.11.2 Other Restrictions

An FT108 error is emitted for all REMOTE aggregate data passed to an unverified RTL routine.

An FT112 error is emitted when a NAME variable is passed inside a NAME() pseudo-function as an ASSIGN parameter and the NAME variable lives REMOTE. The address of the NAME variable must be passed to the procedure, and there is no way in HAL/S to

specify in the formal parameter declaration that the NAME variable lives REMOTE (the REMOTE keyword specifies where it points). Therefore this must be a restriction.

The REMOTE keyword cannot be used on templates for external procedures, external functions, or external programs. Any attempt to do so will result in a PS13 or X15 error message.

The REMOTE keyword cannot be used on a 'D INCLUDE SDF' directive. Any attempt to do so will result in an X14 error message.

8.12 DENSE attribute

The DENSE attribute only affects the data storage packing density of non-NAME, non-arrayed BIT variables in structure templates. The DENSE attribute for all other variable types is ignored.

Any error checking done for DENSE attribute matching or the restriction of the DENSE attribute will only be relevant for non-NAME, non-arrayed BIT variables in structure templates.

The Symbol And Cross Reference Table Listing in the compiler output listing for Phase 1 and the Symbol Data section in the output listing for Phase 4 will only report the DENSE attribute if specified for a structure template, minor node, or a non-Name, non-arrayed BIT variable in a structure template.

8.13 User Note's

This section contains user notes that were required to close Discrepancy Reports (DRs) against the HAL/S-FC compiler. Each user note is followed by the DR number to which it applies and the date that the user note was written.

1. Structure Stack Terminals Not Flagged as 'Stack'

A structure stack variable, such as results from declaring a temporary structure variable is flagged as 'stack' in the SDF. The terminals for the structure are not flagged as stack variables because they are listed as part of the structure template, not the structure variable. Programs (such as HALSTAT) that perform processing on the SDFs need to accommodate this distinction between structure variables and structure templates. The SDF reflects the storage of structures in the symbol table; it has an entry for the structure template and its terminal names and attributes. There are no additional entries for the structure terminals to accompany the structure variable entry. Instead, the structure variable points back to the structure template to access terminal information (DR 50917, 5/6/88).

2. SUBBIT Incorrectly Expected to Return to 32 Bits

The compiler implementation of the SUBBIT pseudo-variable incorrectly assumes that an unsubscripted SUBBIT(X) is always a 32-bit quantity regardless of the type of X. For example, the following SUBBIT statement gets the error EB1 of severity 1 because the compiler assumes that the result of the catenation is 64 bits:

```
DECLARE INT1  INTEGER INITIAL(1);
DECLARE INT2  INTEGER INITIAL(2);
DECLARE BIT32 BIT(32);
BIT32 = SUBBIT(INT1) CAT SUBBIT(INT2);
```

One work around to this problem is to subscript the SUBBIT pseudo-variable as follows:

```
BIT32 = SUBBIT$(1 T0 16)(INT1) CAT SUBBIT$(1 TO 16)(INT2);
```

Another work around is to break the statement up so that the unsubscripted SUBBIT pseudo-variables are not used with the CAT operator:

```
(BIT1 and BIT2 are declared as BIT(16))
BIT1 = SUBBIT(INT1);
BIT2 = SUBBIT(INT2);
BIT32 = BIT1 CAT BIT2;
```

(DR 58320, 5/6/88).

3. Misspelled INITIAL in a Declare Statement Causes Infinite Loop

Phase 1 of the compiler goes into an infinite loop when the keyword 'INITIAL' is misspelled more than one time in a program. The cross-reference table is printed over and over again until the printed page limit is reached, resulting in a 600 DEC abend (DR 58323, 5/6/88).

4. Statement Too Long Error

When a declaration statement of a multi-copy structure with a long initialization list is compiled, the compiler may issue the optimizer error BI309, "Statement too long. Compilation abandoned." The test case that failed compilation with a BI309 error contained a structure declaration with 159 or more copies and an initialization list for each structure copy. The structure template consisted of a name variable, a minor node with 3 bit strings totaling 16 bits, 2 16-bit strings, and 2 integers. The initialization elements for the name variables are what caused the statement to become "too long," as explained in the paragraph below.

Certain types of initialization elements can cause a declaration to grow longer than the optimizer's capacity for a single statement. The problem occurs when the optimizer expands the HALMAT (the HAL/S intermediate language) of the initialization list with NO-OPs to reserve room for optimization, finds that no optimizations are possible, and then finds that the expanded HALMAT is too long to fit in the HALMAT double block. All of the HALMAT for a single statement is required to fit in one double block. In the test case described above, 10 NO-OPs were produced for each name variable initialization element that was itself a subscripted multi-copy structure, such as

```
NAME(STRUCT_NAME.STRUCT_ELEMENT$(3;))
```

(the 3 is for example only, any valid subscript would do). No NO-OPs were produced for simple name variable initialization elements of the form.

```
NAME(VARIABLE).
```

There were enough initialization elements of the first form in the test code that when combined with the remaining initialization elements the declaration statement become too long to fit in a double block.

Two possible work arounds to this problem can be implemented at the source code level. Either rewrite the code to split up the structure into two structures, or use fewer initialization elements that are themselves subscripted multi-copy structures (DR 63453, 5/6/88).

5. Template Cannot Handle All Nested Includes

The following scenario involving TEMPLATEs fails to compile successfully:

-- A compool includes an external program and initializes a name variable to this program name.
-- A second program includes the compool.

During compilation of the second program the compiler abandons compilation after Phase 1 because it cannot resolve the reference to the external program in the compool's included template.

This scenario works with SDFs (DR 101033, 5/6/88).

6. Invalid Error Message DU5 When Including a Template

An invalid error message may be emitted by the compiler when nested compools define and use structure templates. In the set of compilation units described below, the compiler erroneously emits error DU5 ("Reference to undeclared structure template...") when compiling Module A.

Compool C	--	defines a structure template
Compool B	--	includes compool C
	--	contains declarations that refer to the structure template in C (i.e. Name variables or structure variable declarations)
Module A	--	includes only compool B
	--	may or may not refer to any of the variables in compools B and C

One work around to this problem is to add compool C to the include list of module A so that A includes both B and C (DR 101648, 5/6/88).

7. BI504 Error -- Loop in Operand Stack Pointers

The compiler may emit the severity 2 error BI504 ("Loop in Operand Stack Pointers") during Phase 2 of a compilation for certain combinations of statements that use subscripted references to the same multi-copy structure. As an example, the following program gets error BI504:

```
1: DR57989: PROGRAM;
2:   DECLARE I1 INTEGER INITIAL(1);
3:   DECLARE V1 VECTOR(3) SCALAR INITIAL(1,2,3);
4:   STRUCTURE STRUCT2:
4:   1 STR2_V VECTOR(3),
4:   1 STR2_I INTEGER;
5:   DECLARE STRUCT2 STRUCT2-STRUCTURE(3) INITIAL(12#1);
6:   V1= STR2_V$ (I1;);
7:   I1 = STR2_I$(I1;);
8: CLOSE DR57989;
```

The code getting the BI504 error has the following characteristics:

1. The same variable subscript is used for both structure references (stmts 6 and 7 in code segment).
2. Both structure references are on the right-hand side of the expression.
3. The first structure reference is a vector or matrix terminal and the second structure reference is an integer or scalar terminal.
4. The two structure references may or may not have statements between them (certain intervening expressions, such as an integer assignment other than the structure terminal, will prevent the error condition).

Error BI504 indicates an internal compiler failure, not a source code error. Some source code work arounds for this problem are: use different variable subscripts for the two structure references, move the integer or scalar assignment before the vector or matrix assignment, or try separating the two structure references to see if the intervening statements will prevent the error condition (DR 57989, 6/13/88).

8. BS122 Compilation Error on UPDATE PRIORITY Statement

The compiler emits the severity 2 error BS122 ("Indirect Stack Usage Conflict") during Phase 2 of a compilation for an UPDATE PRIORITY statement that does not contain a label. This error indicates an internal compiler failure. The error is not emitted if the UPDATE PRIORITY statement contains a label. For example, the statement below gets error BS122:

```
UPDATE PRIORITY TO 10;
```

whereas the following statement compiles successfully:

```
UPDATE PRIORITY PROCESS_NAME TO l0;
```

(DR 51017, 6/23/88).

9. Limited Range of Input Values Valid for FLOOR Function

The range of input values to the single precision FLOOR function is limited to -32768 ≤ x ≤ 32767.996. Values outside this range will give erroneous results (DR102944, 10/19/88).

10. ZO3 Compilation Error - Loop Invariant Messages

An extraneous ZO3 warning message may sometimes appear when the optimizer pulls some loop invariant HALMAT from an IF-THEN block to outside of a loop. The compiler uses HALMAT to generate AP-101 code. The warning message was added under DR 103032 for possible GPC errors due to the loop invariant code being executed when the IF conditions were false. The ZO3 message could appear even when the loop invariant HALMAT which is moved does not change the subsequent AP-101 code. For this case, a GPC error is not possible. Note also that multiple ZO3 warning messages may be obtained for one line of HAL/S source code since the ZO3 message is due to changes in HALMAT, not changes in AP-101 code (DR105051, 3/92 & 3/93).

11. Underflow/Overflow Restrictions

The range of 'safe' double precision numbers that can be used with a comparison in HAL/S has been determined to be:

```
H 'F1FFFFFFFFFFFFFF' (-1.004336 E+59) to H '8D10000000000001' (-2.430865E-63)
```

and

```
H '0D10000000000001' (1.004336 E+59) to H '71FFFFFFFFFFFFFF' (2.430865E-63)
```

If either or both of the numbers being compared are in the specified range then no underflow or overflow is possible. If both are outside of the 'safe' range then the potential for overflow or underflow exists.

Floating point exponent overflow and underflow are caused by the SED instruction in the case of a comparison operation using SCALAR DOUBLE values. The compiler will generate a LED (Load Double) followed by a SED (Subtract Double) instruction (DR103772, 9/93).

12. BI002 SEVERITY 3 Space Management System Error

A BI002 severity 3 error message is emitted at the end of the compilation listing when compiling a HAL/S source program which generates any of the following valid error messages:

```
BI501: SEVERITY 2, ?? BITS UNACCOUNTED FOR
BS102: SEVERITY 3, SUBPROGRAM STACK OVERFLOW
BS120: SEVERITY 3, DATA STORAGE CAPACITY EXCEEDED
BX100: SEVERITY 2, ?? NOT ADDRESSABLE
BX200: SEVERITY 2, SYT_SCOPE>MAX_SCOPE#
PE100  (only PASS compilers): SEVERITY 2, ILLEGAL NONHAL FUNCTION TYPE
PR6    (only BFS compilers): SEVERITY 2, THE NAME ?? DOES NOT SATISFY HAL/S
NAME UNIQUENESS CRITERIA FOR TASKS;  NAMES OF ALL TASKS AND PROGRAMS MUST BE
UNIQUE TO SEVEN CHARACTERS
```

The BI002 error message states "BI002 SEVERITY 3 BUG IN SPACE MANAGEMENT SYSTEM ->->IN RECORD_FREE, NOT ALLOCATED: ..". The BI002 error message is emitted when the records that are being freed have never been allocated. The records do not get allocated because a valid error message is emitted (a severity 3 error message causes a branch to the end of the compilation). Therefore, the code to allocate the records is never executed and the BI002 error message is emitted (DR108642, 9/93).

13. Unexpected BS117 Error

When a TEMPORARY variable is declared within a DO statement, the compiler attempts to store and subsequently access this data on the stack. Since the first eighteen halfwords of the stack are reserved, the first available entry for storage of TEMPORARY variables is at the nineteenth halfword. For a construct which exhibits arrayness, including multi-copy structures, the compiler attempts to generate a stack location for its zeroth element (i.e., it subtracts the length of one element from the starting location on the stack). When the available stack location minus the length of the zeroth element is less than zero, the compiler emits a BS117 severity 2 error, as there can be no negative stack addressability. The only construct which can have a zeroth element of length greater than the lowest starting address on the stack (nineteen), is a multi-copy structure with a template length of more than eighteen halfwords (the zeroth element of a multi-copy structure is a single copy of the structure). Therefore, when a TEMPORARY multi-copy structure of single-copy length greater than eighteen is declared such that the next available stack location minus the zeroth element length is less than zero, a BS117 error is emitted. This same structure, however, would not receive an error and would produce correct object code if there were enough stack entries assigned prior to it such that the zeroth element would not reside at a negative stack address. Also, the stack available for TEMPORARY variable storage is refreshed upon reaching the end of the DO loop, so the problem can manifest itself over and over again with each subsequent DO statement.

To avoid the BS117 error, the user can either (1) make sure no TEMPORARY multi-copy structures which have a template length greater than eighteen halfwords are used or (2) utilize dummy declarations of TEMPORARY variables just prior to the problem declare such that enough "pad" is established to avoid the BS117 error (DR106639, 9/93).

14. Exceeded Bounds Assignments

The compiler can allow an assignment to exceed the bounds of the receiving data. There are two methods of doing this that could cause problems, and thus should be avoided:

a) A %COPY statement with a variable halfword count field larger than the size of the destination when the destination is local data (this may also go beyond the bounds of a CSECT). The compiler performs bound checking when a literal count is provided but cannot perform these checks when a variable count is used. Local data can be rearranged whenever a compilation unit is modified.
b) Overindexing an arrayed variable (subscript is greater than the receiving data's declared arrayness).

The compiler assumes that arrayed assignments and %COPY are not used in this manner and does not update registers that may have been modified as a result of a violation of these rules (DR109030, 03/95).

15. "Referenced but not Assigned" Error (BI105) Is Not Produced (DR103765, 2/15/95)

It is not always possible to determine at compile time if a variable has been initialized before being referenced. Each situation described below is accompanied by a related code fragment.

Variables that are referenced after being assigned will not produce an error message since branching makes execution order of code very difficult for the compiler to determine.

```
DECLARE SCALAR, S1, S2;
S1 = S2;
S2 = 3;
```

NAME variables pointing to uninitialized NAME variables within templates will not produce error messages as it is impossible to determine what a NAME variable will point to at compile time.

```
STRUCTURE S2:
          1  N1 NAME SCALAR,
          1  S  SCALAR;
DECLARE S2 S2-STRUCTURE;
DECLARE NS2 NAME S2-STRUCTURE INITIAL(NAME(S2));
DECLARE NSCAL NAME SCALAR;
NAME(NSCAL) = NAME(NS2.N1);
```

The compiler also assumes that all compool variables have been initialized, and will accordingly not flag as an error compool variables being referenced in programs but not assigned.

```
CM103765 : COMPOOL;
      DECLARE DUMMY SCALAR INITIAL(0);
      DECLARE B SCALAR;
CLOSE CM103765;
D INCLUDE TEMPLATE CM103765
DR103765: PROGRAM;
DECLARE X SCALAR;
X = B;
```

All parameters passed into a function via parameter list or ASSIGN construct are assumed initialized/assigned prior to the function call if they are referenced in the function before assigned. The calling program also assumes that parameters passed using the ASSIGN construct will be modified within the function.

```
DECLARE SCALAR,A,B,C;
QUICK:  PROCEDURE(X,Y) ASSIGN(Z);
         DECLARE SCALAR,X,Y,Z,T;
         T = X + Y + Z;
CLOSE QUICK;
A = 1;
B = 2;
CALL QUICK(A,B) ASSIGN(C);
```

16. Error Message Is on Wrong Statement (DR109042 -- 6/4/97, DR111369 -- 3/12/01, DR120262 --9/29/03)

There are three known problems with the compiler and where it prints error messages

1) The compiler may report an error on the wrong statement, usually one statement too early for two known cases. This problem occurs when the compiler looks ahead on the next executable statement for an ELSE clause after an "IF <condition> THEN <true part>" statement or a "DO CASE <variable>" statement but instead finds an error. When the compiler finishes scanning the current statement (<true part> of the IF … THEN or the DO CASE statement), it needs to properly terminate the statement so it looks ahead to the first token of the next statement looking for an ELSE clause but finds an error. Since the compiler is not completely finished with the current statement, the error is incorrectly attached to the current statement, not the statement where the token was read from.

   Note: There may be more language constructs for which the parser performs a look-ahead which are not known at this time.

2) The compiler emits an error one statement too late in a structure template or on a factored DECLARE statement where the last item declared receives an error and there is a 'C' card (comment card) or a 'D' card (compiler directive) between the last item declared and the semi-colon. The compiler internally attaches the error message to the correct statement but the 'C' card and/or the 'D' card breaks the continuity of parsing of the HAL/S source code and forces a flush of the print stack before the error is detected. Since the print stack is prematurely flushed, the compiler does not print the error until the next statement.

   Note: There may be more language constructs for which the compiler emits an error one statement too late which are not known at this time.

3) The compiler generates the M1 error (ILLEGAL CARD TYPE - CHANGED TO COMMENT) with the wrong statement. If there is a D-card or a C-card line before the statement with the invalid card-type, then the error will not be printed until the compiler prints the next M-card line it processes. This occurs because errors are only printed when the compiler prints an M-card line. If there are M-card lines before the statement with an invalid card type then the error will be printed one or two statements before the invalid card-type line. This happens because the compiler retrieves the next group of E/M/S lines before it completes processing an M-card line. When the compiler retrieves a line, it checks for invalid card types and

generates the M1 error when it finds one. Since it has not completed processing and printing the M-card line that precedes the line with the invalid card-type, the compiler prints the M1 error when it prints the M-card line it is processing. :

Examples for problem 1 are shown in the following HAL/S listing compiled with PASS 28.0 / BFS 12.0:

```
0   1 M| E9042C:                                                                    |E9042C
    1 M| PROGRAM;                                                                   |E9042C
    2 M|   DECLARE SUCCESS BOOLEAN INITIAL(TRUE);                                  |E9042C
    3 M|   DECLARE BOL1 BOOLEAN,                                                    |E9042C
    3 M|           BOL2 BOOLEAN;                                                    |E9042C
    4 M|   DECLARE INT1 INTEGER,                                                    |E9042C
0     E|                                          .                                |
    4 M|           NAM_BOL1 NAME BOOLEAN INITIAL(NAME(BOL1)),                       |E9042C
0     E|                                          .                                |
    4 M|           NAM_BOL2 NAME BOOLEAN INITIAL(NAME(BOL2));                       |E9042C
    5 M|   DECLARE ARRAY(3),                                                        |E9042C
    5 M|             A, B, C ARRAY(5)                                               |E9042C
0     C|                                                                            |E9042C
      C|  THIS SET OF COMMENT CARDS ARE NECESSARY TO CREATE THE USER                |E9042C
      C|  NOTE SCENARIO #2. DO NOT REMOVE THEM!!!!! THE D6 ERROR IS INCORRECTLY     |E9042C
      C|  REPORTED ON THE 'IF SUCCESS THEN' STATEMENT.                              |E9042C
      C|                                                                            |E9042C
    5 M|             ;                                                              |E9042C
0     C|                                                                            |E9042C
      C|  THIS IS SCENARIO #1 -- THE COMPILER IS IN A LOOK-AHEAD STATE WHEN IT      |E9042C
      C|  FINDS THE %NAMEADDS TOKEN AND REPORTS THE XM1 ERROR ON THE 'INT1 = 2;'    |E9042C
      C|  STATEMENT, WHICH IS INCORRECT.                                            |E9042C
      C|                                                                            |E9042C
0     E|         .                                                                  |
    6 M|   IF SUCCESS THEN                                                          |E9042C
0***** D6       ERROR #1 OF SEVERITY 2. *****
 ***** THE DECLARATION OF C HAS BOTH FACTORED AND NON-FACTORED ARRAY SPECIFICATIONS;
 ***** THE NON-FACTORED SPECIFICATION WILL BE USED
    7 M|     INT1 = 2;                                                            |E9042C
0***** XM1      ERROR #2 OF SEVERITY 2. *****
 ***** %NAMEADDS IS A NONEXISTENT %MACRO
 ***** LAST ERROR WAS DETECTED AT STATEMENT 6. *****
0     E|               .         .                                                |
    8 M|   %NAMEADDS(NAM_BOL1, NAM_BOL2, 0);                                       |E9042C
0     C|                                                                            |E9042C
0     E|               .         .                                                |
    9 M|   %NAMEADDS(NAM_BOL1, NAM_BOL2, 0);                                       |E9042C
0***** XM1      ERROR #3 OF SEVERITY 2. *****
 ***** %NAMEADDS IS A NONEXISTENT %MACRO
 ***** LAST ERROR WAS DETECTED AT STATEMENT 7. *****
0     C|                                                                            |E9042C
      C|  THIS IS SCENARIO #1 -- THE COMPILER IS IN A LOOK-AHEAD STATE WHEN IT      |E9042C
      C|  FINDS THE NAMES TOKEN AND REPORTS THE DU1 ERROR ON THE 'DO CASE INT1;'    |E9042C
0     C|  STATEMENT, WHICH IS INCORRECT.                                            |E9042C
      C|                                                                            |E9042C
   10 M|   DO CASE INT1;                                                            |E9042C
0***** DU1      ERROR #4 OF SEVERITY 2. *****
 ***** UNDECLARED IDENTIFIER NAMES
 ***** LAST ERROR WAS DETECTED AT STATEMENT 9. *****
   11 M| 1   NAMES (NAM_BOL1) = NAME(BOL1);                                        |CASE1
0***** P8       ERROR #5 OF SEVERITY 2. *****
 ***** THE FOLLOWING SYMBOL IS SYNTACTICALLY ILLEGAL IN THE CONTEXT USED: (
 ***** ERROR RECOVERY MAY CAUSE SUBSEQUENT SPURIOUS ERRORS
 ***** LAST ERROR WAS DETECTED AT STATEMENT 10. *****
   12 M| 1   WRITE(6) 'CASE #2.';                                                  |CASE2
   13 M| 1   WRITE(6) 'CASE #3.';                                                  |CASE3
   14 M|   END; |ST#10                                                            |E9042C
0     C|   %SVCI(0);                                                             |E9042C
   15 M| CLOSE
```

17. Constant Double Scalar Converted to a Character as Single Precision (DR 109083, 02/14/97, DR120223, 7/7/04)

When a scalar arithmetic expression that contains only literals and non-aggregate constants is converted to a character string via a runtime library routine (ETOC or DTOC), the calculation and conversion will be in single precision. To force the result to be in double precision use the @DOUBLE function.

18. Long Statement with Subscripts or Exponents Is Not Printed Correctly (DR109081, 7/16/98), (DR111326, 1/10/00)

Extremely long statements may not be printed correctly if the end of each 100 character line is part of a subscript or exponent. Either a BS6 error message will be emitted and the remaining subscript or exponent will not be printed correctly, or the line will be interrupted with no error issued. These problems may be eliminated by inserting a comment line. The effects in each case are visual only and do not affect program execution after the error is downgraded.

19. Comparison of Structures Containing Arrayed Character Strings Fails (DR101047, 5/22/90)

The unverified runtime library routine, CPRA (it compares remote arrays of character strings when the arrays are located in structures for '=' or '^='), may cause other object code to be incorrect besides the compare operation.

20. Comparison of Structures Containing Arrayed Character Strings Fails (DR101047, 5/22/90)

The unverified runtime library routine, CPRA (it compares remote arrays of character strings when the arrays are located in structures for '=' or '^='), may cause other object code to be incorrect besides the compare operation.

21. Statement Split in Output Listing after REPLACE Macro (DR111340, 09/17/99)

If a non-expanded REPLACE macro contains the last token of a statement (e.g. a semi-colon or 'THEN') and is followed by a Comment or Directive (C or D card), the statement may be split in the output listing, and the REPLACE macro will be printed after the C or D card. The revision level information of the macro may also be lost from the Current Scope field.

22. Long Factored DECLARE Indented Incorrectly (DR111320 - Incorrectly Printed Exponent Line in Output Listing, 1/7/00)

Long factored DECLAREs may not be indented correctly in the output listing. If a long statement cannot fit on the statement stack, the first part is printed to make room on the stack. At this point the type of statement has not yet been identified as a factored DECLARE so it is not indented properly. Once enough of the statement has been added to the stack, the statement is identified, and the remaining lines of the statement are correctly indented. This usually occurs when each identifier has a separate INITIAL clause rather than sharing a common clause.

23. REPLACE Macro Statements Incorrectly Printed (DR111320 - Incorrectly Printed

Exponent Line in Output Listing, 1/7/00)

Statements containing REPLACE macros may be incorrectly printed in the output listing. For example, if a factored or compound DECLARE contains a REPLACE macro, the statement may not be indented correctly. The identifiers may be printed on the same line as the attributes or on a line separate from the rest of the statement. These types of statements are printed based on the location of certain attributes in the statement stack. If a REPLACE macro token occupies one of these slots, the statement may not be properly identified and hence not indented correctly.

```
0   1 M| TEST:                                                    |TEST
    1 M| PROGRAM;                                                 |TEST
    2 M|   REPLACE ATTR BY "ARRAY(2) INTEGER INITIAL (1,2)";     |TEST
0     C|  FACTORED DECLARE                                       |TEST
    3 M|   DECLARE ATTR, F1,F2,F3;                               |TEST
+                 ____
0     C|  COMPOUND DECLARE                                       |TEST
    4 M|   DECLARE C1 ATTR,                                      |TEST
+                     ____
    4 M|           C2                                            |TEST
    4 M|           ATTR,                                         |TEST
+                   ____
    4 M|           C3 ATTR;                                      |TEST
+                      ____
    5 M| CLOSE;                                                  |TEST
```

24. BI512 Error for Large Repetition Factor in Shaping Function (DR111371 -- 03/02/01)

A BI512 error (DISPLACEMENT > 2047 IS INVALID FOR INDEXED RS TYPE INSTRUCTION) may be emitted if an INTEGER or SCALAR shaping function is used to create a multidimensional array or a MATRIX shaping function is used to create a matrix and the array or matrix is over 2034 halfwords in size (a single precision integer is 1 halfword and a single precision scalar is 1 fullword) and a repetition factor is used. Depending on the stack usage, shaping functions smaller than 2034 halfwords can also cause the problem.

25. No Compile Time Errors for Invalid CHARACTER Subscripts (DR111 376 - No SR3 Error Generated for CHARACTER Shaping Function -- 05/16/01)

A. The compiler will not generate errors during compilation for an invalid CHARACTER subscript due to the use of the '#' in a partition subscript (TO - or AT - partition). Since the maximum length of a CHARACTER variable is the current length of the variable that may be varied during the execution, there is no way to check for the validity of a CHARACTER partition subscript that uses the "#" (to indicate the last element) during compilation.

As examples, the following subscripts will not receive errors during compilation (run-time errors will be generated):

```
DECLARE C3 CHARACTER(3);
DECLARE C4 CHARACTER(4) INITIAL('123')

C3=C4$(2 AT #+1);
C3=CHARACTER$(4 AT #-2)(C4);
```

B. For the same reason that the length of a CHARACTER variable may be dynamic, the following subscript will receive a run-time error for referencing an 'out-of-range' element:

```
C3=C4$4
```

C. The compiler will not check for the validity of the subscript of a CHARACTER shaping function that has an INTEGER or SCALAR argument since the length of the INTEGER/SCALAR cannot be determined at compiler time. For example:

```
DECLARE S SCALAR INITIAL(1);
DECLARE C3 CHARACTER(3);

C3=CHARACTER$3(S);
```

26. BI223 Error for CHARACTER(*) Parameter (DR111386-BIX Loop Incorrect for CHARACTER(*)--03/29/02)

Since the size of a CHARACTER(*) variable is not known until run time, the compiler could not determine the amount of space to save for CHARACTER(*) variables that were passed as an argument to a Procedure or Function, an INTEGER/SCALAR Shaping Function, or a Built-in Function. Instead of assigning the maximun amount of space needed for the variable (enough space for a CHARACTER string of length 255), code was added to emit an error when an unknown size has been requested (BI223 Severity 2-NEGATIVE STORAGE SIZE). BS105 or BS102 errors may also be emitted. These problems will only occur if the variable was attempting to be copied to the run-time stack. Possible workarounds include assigning the CHARACTER(*) to a local variable first or only passing one string as the argument.

27. BS108 Error for Large Negative Literal Value (DR120268 - Incorrect Error Near Negative Limit -- 1/12/05)

Non-integer literal values less than -2,147,483,647.5 but greater than (or equal to) -2,147,483,648.5 incorrectly receive a BS108 error (LITERAL PROCESSING FAILURE) when converted to an Integer (e.g. when these literals are used to initialize, assign to, or pass to an Integer). To avoid this error, assign the literal to a double Scalar or use an Integer literal value instead.

Appendix A PROTOTYPE CATALOGUED PROCEDURES

This Appendix contains the two catalogued procedures discussed in Chapter 2 which are needed to compile and link edit a HAL/S program. The two procedures perform varying degrees of processing. The procedure names and their functions are:

HALFC - Compiles a HAL/S program

HALFCL - Compiles and links a HAL/S program

The following comments describe each of the processes involved in detail. Line references are to the most complex procedure, HALFCL; the other procedure simply consists of the first job step invoked in HALFCL.

Line 1000 - This PROC statement names the procedure and defines the symbolic parameters. The OPTION parameter is the means of supplying optional information to the compiler.

Step One (HAL): Compilation

Lines 20000-30000 - The name of the compilation step is HAL. The name of the actual program to be executed is MONITOR. MONITOR handles all compiler/OS interfaces and also performs the actual loading and overlaying of the three phases of the compiler. The compiler requires a 350K region, although a larger region may be specified. The compiler will always use all the memory it is given. A larger region will generally result in smaller compilation times. A default time limit of 1 minute is supplied. This is sufficient for most average size HAL/S programs (approximately 300 HAL/S statements). The PARM field contains the compile-time options. The OPTION field receives any user-specified options.

Line 40000 - The STEPLIB DD card specifies the location of the load module library containing the module MONITOR needed to run the compiler. This card may define any direct access library which contains the proper module or may be deleted at installations where the module has been made part of the system library (SYS1.LINKLIB).

Line 50000 - The PROGRAM DD card defines the compiler phases that are to be used. This dataset has a DCB of the form:

DCB= (RECFM=F, LRECL=7200, BLKSIZE=7200)

and may reside on direct access or magnetic type. It is recommended, however, that direct access be used.

Line 60000 - The SYSPRINT DD card defines the primary listing dataset. This is generally assigned to a system output class, but may be associated with any sequential dataset with the proper characteristics. The user may supply DCB attributes; the record format must be FBA with a logical record length of 133 and any appropriate block size.

Line 70000 - The LISTING2 DD card defines the secondary listing dataset controlled by the LISTING2 compiler option. It may define a system output class or any sequential dataset. The DCB requirements are the same as for SYSPRINT.

Lines 80000-100000 - The OUTPUT3 DD card defines the dataset which is to receive object code which is produced by the compiler. In the prototype procedure, this dataset is given a temporary name (&&HALOBJ) and is passed (DISP=(MOD,PASS)) to subsequent steps. Since this dataset contains card images, it must have a logical record length of 80 and a record format of F or FB. The blocking factor may be any legal multiple of 80. This DD card may specify a direct access, magnetic tape, or unit record device.

Line 110000 - The OUTPUT4 DD card defines the location to which a duplicate of the OUTPUT4 file will be sent under control of the DECK compiler option. Its DCB characteristics are identical to OUTPUT3.

Lines 120000-130000 - The OUTPUT5 DD card defines the dataset which will receive the Simulation Data File (SDF).

The OUTPUT5 DD card must define a partitioned dataset. The SPACE parameters used in the prototype procedure are more than adequate to contain the mapping file of any one HAL/S program.

One member of the partitioned dataset is created for a compilation unit. The name of the member is the characteristic name of the unit being compiled, padded with blanks if necessary and preceded by the characters ##.

If a member with the desired name does not exist at compile-time, one is created. If a member with the desired name already exists, it is replaced by the new member.

The use of a permanent partitioned dataset makes it possible to maintain a "library" of SDFs, with the member names uniquely specifying the HAL/S compilations to which they correspond (see Section 3.5).

Lines 140000-150000 - The OUTPUT6 DD card defines the partitioned dataset onto which templates for compilation units may be placed. Refer to Section 3.4 for details on template generation. The DCB attributes for the statement should be compatible with those used for primary compiler input (SYSIN) and secondary compiler input (INCLUDE).

Line 160000 - The OUTPUT7 DD card specified as "Dummy" is included to cover an option which is not used for normal compilations.

Line 170000 - The ERROR DD card defines the partitioned dataset which contains the error message texts used by the compiler. This dataset is supplied with the compiler and, being a partitioned dataset, must reside on a direct access volume.

Lines 180000-230000 - The FILE1 through FILE6 DD cards specify work files. These files are used for interphase communication. The device may be either direct access or magnetic tape. Space equivalent to approximately 60 tracks on a 2314 should be available for each DD card. The DCB is internally specified and should not be altered by JCL.

Step Two (LKED): Link Editing

Lines 240000-340000 - These lines define the link edit step, processing the object module output of the HAL/S compilation and producing a load module for the flight computer.

```
010000    //HALFC    PROC OPTION=,LEVEL=HALS101
020000    //HAL       EXEC PGM=MONITOR,REGION=350K,TIME=1,
030000    //             PARM='NOZCON,&OPTION'
040000    //STEPLIB   DD DISP=SHR,DSN=&LEVEL..MONITOR
050000    //PROGRAM   DD DISP=SHR,DSN=&LEVEL..COMPILER
060000    //SYSPRINT  DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
070000    //LISTING2  DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
080000    //OUTPUT3   DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
090000    //             DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
100000    //             DSN=&&HALOBJ
110000    //OUTPUT4  DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400,FUNC=I)
120000    //OUTPUT5  DD DISP=(MOD,PASS),DSN=&&HALSDF,SPACE=(TRK,(2,2,1)),
130000    //             DCB=(RECFM=F,LRECL=1680,BLKSIZE=1680),UNIT=SYSDA
140000    //OUTPUT6  DD DISP=(MOD,PASS),DSN=&&TEMPLIB,SPACE=(TRK,(2,2,1)),
150000    //             DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),UNIT=SYSDA
160000    //OUTPUT7  DD DUMMY,DCB=(RECFM=FM,LRECL=121,BLKSIZE=121)
170000    //ERROR    DD DISP=SHR,DSN=&LEVEL..ERRORLIB
180000    //FILE1    DD UNIT=SYSDA,SPACE=(CYL,3)
```

```
190000    //FILE2    DD UNIT=SYSDA,SPACE=(CYL,3)
200000    //FILE3    DD UNIT=SYSDA,SPACE=(CYL,3)
210000    //FILE4    DD UNIT=SYSDA,SPACE=(CYL,3)
220000    //FILE5    DD UNIT=SYSDA,SPACE=(CYL,3)
230000    //FILE6    DD UNIT=SYSDA,SPACE=(CYL,3)
```

Figure A-1 HALFC

```
010000 //HALFCL    PROC OPTION=,LEVEL=HALS101
020000 //HAL       EXEC PGM=MONITOR,REGION=350K,TIME=1,
030000 //              PARM='NOZCON,&OPTION'
040000 //STEPLIB   DD DISP=SHR,DSN=&LEVEL..MONITOR
050000 //PROGRAM   DD DISP=SHR,DSN=&LEVEL..COMPILER
060000 //SYSPRINT  DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
070000 //LISTING2  DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
080000 //OUTPUT3       DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
090000 //              DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
100000 //              DSN=&&HALOBJ
110000 //OUTPUT4   DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400,FUNC=I)
120000 //OUTPUT5       DD DISP=(MOD,PASS),DSN=&&HALSDF,SPACE=(TRK,(2,2,1)),
130000 //          DCB=(RECFM=F,LRECL=1680,BLKSIZE=1680),UNIT=SYSDA
140000 //OUTPUT6       DD
                   DISP=(MOD,PASS),DSN=&&TEMPLIB,SPACE=(TRK,(2,2,1)),
150000 //          DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),UNIT=SYSDA
160000 //OUTPUT7   DD DUMMY,DCB=(RECFM=FM,LRECL=121,BLKSIZE=121)
170000 //ERROR     DD DISP=SHR,DSN=&LEVEL..ERRORLIB
180000 //FILE1     DD UNIT=SYSDA,SPACE=(CYL,3)
190000 //FILE2     DD UNIT=SYSDA,SPACE-(CYL,3)
200000 //FILE3     DD UNIT=SYSDA,SPACE=(CYL,3)
210000 //FILE4     DD UNIT=SYSDA,SPACE=(CYL,3)
220000 //FILE5     DD UNIT=SYSDA,SPACE=(CYL,3)
230000 //FILE6     DD UNIT=SYSDA,SPACE=(CYL,3)
240000 //LKED      EXEC
                   PGM=LNK101,TIME=1,PARM=(LIST,MAP,NOXREF,NOAUTO,NOTABLE)
                   ,
250000 //              COND=(0,LT,HAL)
260000 //STEPLIB   DD DISP=SHR,DSN=&LEVEL..SUPPORT
270000 //SYSPRINT  DD SYSOUT=A,DCB=BLKSIZE=121
280000 //SYSLIB    DD DISP=SHR,DSN=&LEVEL..RUNLIB
285000 //          DD DISP=SHR,DSN=&LEVEL..ZCONLIB
290000 //SYSLIN    DD DISP=SHR,DSN=&LEVEL..NUCLEUS
300000 //          DD DISP=(OLD,DELETE),DSN=&&HALOBJ
310000 //          DD DDNAME=SYSIN
320000 //SYSLMOD   DD DSN=&&LOADMOD(GO),DISP=(,PASS),UNIT=SYSDA,
330000 //              SPACE=(CYL,(1,1,1))
340000 //SYSUT1    DD SPACE=(CYL,(1,1)),UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD))
```

Figure A-2 HALFCL

This page intentionally left blank.

Appendix B COMPILE-TIME ERROR BEHAVIOR

The text of all the possible compile-time error messages is presented here, along with a series of explanations giving more detail about the error encountered, its probable cause, and often a specific reference in the Language Specification for more information. See section 4.3 for a discussion of the format of error messages in the compiler listing.

The first table gives the mnemonic naming scheme used to identify the class-subclass structure of the error messages. The complete list of messages and associated explanations is presented next. The occurrence of double question marks (??) in the text of the messages listed here indicate positions at which text specific to each actual error will be inserted (e.g., a variable name may be inserted to make a clear identification of the error source).

Errors marked as "Internal compiler failure" should not occur; if they do, a compiler representative should be informed. Note: "b" denotes a blank.

CLASS A: ASSIGNMENT STATEMENTS

```
A  ARRAY ASSIGNMENT
V  COMPLEX VARIABLE ASSIGNMENT
b MISCELLANEOUS ASSIGNMENT
```

CLASS B: COMPILER TERMINATION

```
B  HALMAT BLOCK SIZE
I  INTERNAL ERRORS
N  NAME SCOPE NESTING
S  STACK SIZE LIMITATIONS
T  TABLE SIZE LIMITATIONS
X  COMPILER ERRORS
b MISCELLANEOUS
```

CLASS C: COMPARISONS

```
b GENERAL COMPARISONS
```

CLASS D: DECLARATION ERRORS

```
A  ATTRIBUTE LIST
C  STORAGE CLASS ATTRIBUTE
D  DIMENSION
I  INITIALIZATION
L  LOCKING ATTRIBUTE
N  NAME
Q  STRUCTURE TEMPLATE TREE ORGANIZATION
S  FACTORED/UNFACTORED SPECIFICATION
T  TYPE SPECIFICATION
U  UNDECLARED DATA
b MISCELLANEOUS
```

CLASS E: EXPRESSIONS

A ARRAYNESS
B BIT STRINGS EXPRESSIONS
C CROSS PRODUCT
D DOT PRODUCT
L LIST EXPRESSIONS
M MATRIX EXPRESSIONS
N NAME
O OUTER PRODUCT
V VECTOR EXPRESSIONS
b MISCELLANEOUS EXPRESSIONS

CLASS F: FORMAL PARAMETERS & ARGUMENTS

D DIMENSION AGREEMENT
N NUMBER OF ARGUMENTS
S SUBBIT ARGUMENTS
T TYPE AGREEMENT
b MISCELLANEOUS

CLASS G: STATEMENT GROUPINGS (DO GROUPS)

B BIT TYPE CONTROL EXPRESSION
C CONTROL EXPRESSION
E EXIT/REPEAT STATEMENTS
L END LABEL
V CONTROL VARIABLE

CLASS I: IDENTIFIERS

L LENGTH
R REPLACED IDENTIFIERS
S QUALIFIED STRUCTURE NAMES

CLASS L: LITERALS

B BIT STRING
C CONVERSION TO INTERNAL FORMS
F FORMAT OF ARITHMETIC LITERALS
S CHARACTER STRING

CLASS M: MULTILINE FORMAT

C OVERPUNCH CONTEXT
E E-LINE
O OVERPUNCH USE
S S-LINE
b CONTENTS

CLASS P: PROGRAM CONTROL & INTERNAL CONSISTENCE

A ACCESS CONTROL
C COMPOOL BLOCKS
D DATA DEFINITION
E EXTERNAL TEMPLATES
F FUNCTION RETURN EXPRESSIONS
L LABELS
M MULTIPLE DEFINITIONS
P BLOCK DEFINITION

R ON ERROR/SVCI MACRO (ONLY EMITTED BY BFS COMPILER)
S PROCEDURE/FUNCTION TEMPLATES
T TASK DEFINITIONS
U CLASS FROM UPDATE BLOCKS
b MISCELLANEOUS

CLASS Q: SHAPING FUNCTIONS

A ARRAYNESS
D DIMENSION INFORMATION
S SUBSCRIPTS
X ARGUMENT TYPE

CLASS R: REAL TIME STATEMENTS

E ON/SEND ERROR STATEMENTS
T TIMING EXPRESSIONS
U UPDATE BLOCKS

CLASS S: SUBSCRIPT USAGE

C SUBSCRIPT COUNT
P PUNCTUATION
Q PRECISION QUALIFIER
R RANGE OF SUBSCRIPT VALUES
S USAGE ASTERISKS
T SUBSCRIPT TYPE
V VALIDITY OF USAGE

CLASS T: I/O STATEMENTS

C CONTROL
D DEVICE NUMBER
b MISCELLANEOUS

CLASS U: UPDATE BLOCKS

I IDENTIFIER USAGE
P PROGRAM BLOCKS
T I/O

CLASS V: COMPILE-TIME EVALUATIONS

A ARITHMETIC OPERATIONS
C CATENATION OPERATIONS
E UNCOMPUTABLE EXPRESSIONS
F FUNCTION EVALUATION

CLASS X: IMPLEMENTATION DEPENDENT FEATURES

A PROGRAM ID DIRECTIVE
D DEVICE DIRECTIVE
I INCLUDE DIRECTIVE
R DATA_REMOTE DIRECTIVE
U UNKNOWN OR INVALID DIRECTIVE

CLASS Y: ADVISORY MESSAGES

A ASSIGNMENTS
C COMPARISONS
D DECLARES
E EXPRESSIONS

```
F  FORMAL PARAMETERS & ARGUMENTS
```

CLASS Z: PRODUCE "TRAP" MESSAGES FROM THE COMPILER

```
B  BIT INSTRUCTION
C  %COPY
O  OPTIMIZER
P  REGISTER PRESSURE
R  REMOTE
S  SUBBIT EXPRESSION
```

ERROR MESSAGES FOR MAJOR CLASSIFICATION A

CLASSIFICATION "A" ERRORS ARE RELATED TO ASSIGNMENT STATEMENTS

AA1 - SEVERITY 2

ARRAYNESS OF LEFT HAND SIDE OF ASSIGNMENT DOES NOT MATCH THAT OF RIGHT HAND SIDE

IN "L=R", IF R IS ARRAYED, THEN L MUST POSSESS IDENTICAL ARRAYNESS. SPEC-SECTION 7.3

AA2 - SEVERITY 2

ARRAYNESS OF ?? IS INCONSISTENT WITH THAT OF OTHER LEFT HAND SIDE VARIABLES IN MULTIPLE ASSIGNMENTS SUCH AS "L,M,N=R", ALL LEFT-HAND VARIABLES MUST POSSESS IDENTICAL ARRAYNESS.

SPEC-SECTION 7.3

AA3 - SEVERITY 2

ARRAYNESS OF ?? DISAGREES WITH ARRAYNESS OF ITS SUBSCRIPTING

IF AN ARRAYED SUBSCRIPT APPEARS WITHIN AN ARRAYED EXPRESSION, THE ARRAYNESS MUST MATCH.

AV0 - SEVERITY 2

ARGUMENTS ON EITHER SIDE OF NAME ASSIGNMENT ARE INCOMPATIBLE

IN "NAME(L)=NAME(R)", L MUST BE A NAME VARIABLE DECLARED TO BE OF THE SAME TYPE, ETC, AS THE ORDINARY OR NAME VARIABLE, "R". SPEC-SECTION 11.4.1

AV1 - SEVERITY 2

TYPE OF ?? IS ILLEGAL FOR ASSIGNMENT FROM GIVEN RIGHT-HAND SIDE

IN "L=R", L AND R MUST MATCH IN TYPE, WITH THE FOLLOWING EXCEPTIONS: INTEGER TO SCALAR, SCALAR TO INTEGER, ZERO TO VECTOR/MATRIX, AND INTEGER/SCALAR TO CHARACTER ASSIGNMENTS ARE ALL POSSIBLE.

AV2 - SEVERITY 2

MATRIX DIMENSIONS DISAGREE ACROSS ASSIGNMENT

MAKE SURE THE DIMENSIONS OF THE MATRIX REFERRED TO AND THE MATRIX ASSIGNED INTO ARE ALL THE SAME.

AV3 - SEVERITY 2

VECTOR LENGTHS DISAGREE ACROSS ASSIGNMENT

MAKE SURE THE LENGTH OF THE VECTOR REFERRED TO AND THE VECTOR ASSIGNED INTO ARE ALL THE SAME.

AV4 - SEVERITY 2

TREE ORGANIZATIONS DO NOT MATCH ACROSS ASSIGNMENT

IN "S1=S2", BOTH STRUCTURES MUST HAVE IDENTICAL ORGANIZATION. SPEC-SECTION 7.3

AV5 - SEVERITY 2

ONLY ONE OPERAND IN ASSIGNMENT IS A NAME PSEUDO-FUNCTION OR NULL.

THE NAME PSEUDO-FUNCTION MUST OCCUR ON BOTH SIDES OF THE ASSIGNMENT: "NAME(L)=R" AND "L=NAME(R)" ARE BOTH ILLEGAL. ONLY NAME(L)=NULL IS LEGAL.

A1 - SEVERITY 2

ILLEGAL ASSIGNMENT TO CONSTANT OR PARAMETER ??

DATA ITEMS DECLARED AS CONSTANTS MAY NOT BE CHANGED (ASSIGNED INTO); NEITHER MAY INPUT PARAMETERS IN A PROCEDURE. SPEC-SECTION 7.3

A2 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO POSSESSES SUBSCRIPTS ILLEGAL IN ASSIGNMENT CONTEXT

IN "NAME(L)=NAME(R)", THE NAME VARIABLE 'L' MAY NOT POSSESS ARRAYNESS OR COMPONENT SUBSCRIPTING.

SPEC-SECTION 11.4.11

A3 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO DOES NOT POSSESS THE NAME ATTRIBUTE: IT IS THEREFORE ILLEGAL IN ASSIGNMENT CONTEXT

IN "NAME(L)=NAME(R)", THE VARIABLE L MUST BE A DECLARED NAME VARIABLE. SPEC-SECTION 11.4.6

ERROR MESSAGES FOR MAJOR CLASSIFICATION B

CLASSIFICATION "B" ERRORS RESULT FROM ABORTIVE COMPILER FAILURES

NOTE: MANY ERRORS IN THIS SECTION ARISE BECAUSE PROGRAM ORGANIZATION LIMITS INTERNAL TO THE COMPILER HAVE BEEN REACHED. MOST OFTEN THIS IS DUE TO THE COMPILER'S ATTEMPT TO INTERPRET HAL/S STATEMENTS OF EXTREMELY HIGH COMPLEXITY: E.G., WITH MANY NESTED FUNCTION INVOCATIONS IN WHICH PARAMETERS ARE ARRANGED IN SEVERAL DIMENSIONS, ETC. IF THIS OCCURS, THE USER SHOULD TRY TO SIMPLIFY THE SOURCE PROGRAM AS MUCH AS IS POSSIBLE, BREAKING DOWN COMPLICATED STATEMENTS INTO A SERIES OF SIMPLER CONSTITUENTS.

BB1 - SEVERITY 3

INTERMEDIATE CODE STORAGE OVERFLOW: ERROR SCAN CONTINUING

SIMPLIFY: TOO MANY LITERALS, INITIAL VALUES, ETC., SPECIFIED AT ONCE. BREAK UNIT INTO SEPARATE COMPILATIONS.

BI000 - SEVERITY 3

NOTICE FROM DESCRIPTOR_MONITOR: UNABLE TO ALLOCATE ADDITIONAL DESCRIPTORS. JOB ABANDONED.

CAUSES A USER 4000 ABEND.

BI001 - SEVERITY 3

INVALID CALL TO RETURN DESCRIPTORS

CAUSES A USER 4000 ABEND.

BI002 - SEVERITY 3

BUG IN SPACE MANAGEMENT SYSTEM __ ??

CAUSES A USER 4000 ABEND.

BI003 - SEVERITY 3

BUG IN SPACE MANAGEMENT SYSTEM

CAUSES A USER 4000 ABEND.

BI004 - SEVERITY 3

TOO MANY SPACE MANAGEMENT ERRORS -- ABORTING

CAUSES A USER 4000 ABEND.

BI005 - SEVERITY 3

NOTICE FROM COMPACTIFY: INSUFFICIENT SORT AREA. JOB ABANDONED.

CAUSES A USER 4000 ABEND. TRY INCREASING REGION SIZE AND/OR THE FREE PARAMETER FOR THE COMPILATION.

BI006 - SEVERITY 3

BUG IN COMPACTIFY *** ABORTING

CAUSES A USER 4000 ABEND.

BI007 - SEVERITY 3

NOT ENOUGH SPACE AVAILABLE -- TRY LARGER REGION.

CAUSES A USER 4000 ABEND.

BI008 - SEVERITY 3

NOTICE FROM COMPACTIFY: INSUFFICIENT MEMORY FOR STRING STORAGE. JOB ABANDONED.

CAUSES A USER 4000 ABEND.

BI009 - SEVERITY 3
NOT ENOUGH SPACE FOR INCREASED ALLOCATION, GIVING UP.
CAUSES A USER 4000 ABEND.

BI010 - SEVERITY 3
SPACE MANAGEMENT YOYOING -- TRY LARGER REGION.
CAUSES A USER 4000 ABEND.

BI011 - SEVERITY 3
NOT ENOUGH FREE STRING AVAILABLE, RERUN WITH LARGER REGION
CAUSES A USER 4000 ABEND.

BI100 - SEVERITY 2
ERROR LOOKUP FAILURE: ??

BI101 - SEVERITY 2
WARNING *** TOO MANY ERROR MESSAGES GENERATED FOR THIS STATEMENT; NOT ALL ERRORS ARE LISTED FULLY

BI102 - SEVERITY 2
XREF LIST WAS FILLED, AND IS THEREFORE INCOMPLETE. ONE OR MORE SPURIOUS ADMONITIONS MAY APPEAR ABOVE.

BI103 - SEVERITY 2
INVALID MFID CHARACTER ??

BI104 - SEVERITY 2
ERROR *** ONE OR MORE DUPLICATE BLOCK NAMES FOUND.

BI105 - SEVERITY 2
ERROR ONE OR MORE VARIABLES REFERENCED BUT NOT ASSIGNED.

BI106 - SEVERITY 4
TOO MANY ERRORS; COMPILATION ABORTED

BI107 - SEVERITY 2
ERROR CLASS NOT FOUND FOR DOWNGRADE DIRECTIVE, PLEASE CONFIRM ERROR CLASS AND RECOMPILE.

BI108 - SEVERITY 2
NO ERROR NUMBER TO DOWNGRADE, DETERMINE ERROR NUMBER TO DOWNGRADE AND RECOMPILE.

BI109 - SEVERITY 2
YOU HAVE EXCEEDED THE NUMBER OF ALLOWABLE DOWNGRADES, NEED TO REMOVE UNNEEDED DOWNGRADES AND RECOMPILE.
THE LIMIT ON DOWNGRADES IS DOCUMENTED IN APPENDIX B OF THE FC SPEC.

BI200 - SEVERITY 2
SYT VPTR NOT FOUND FOR ??

BI201 - SEVERITY 2
MALFORMED TEMPLATE, STRUCTURE WALK INHIBITED

BI202 - SEVERITY 2
STRUCTURE SUBSCRIPT AT ?? NOT PREPROCESSED

BI203 - SEVERITY 2
SF START NOT FOUND

BI204 - SEVERITY 2
HALMAT OP ?? NOT PREPROCESSED

BI205 - SEVERITY 2
NON-INTEGER SUBSCRIPT

BI206 - SEVERITY 2
INIT REPEAT LIST OPERATOR MISMATCH

BI207 - SEVERITY 2
STRUCT_COPIES NOT EQUAL TO IDLP COUNT

BI208 - SEVERITY 2
SUBSCRIPT NOT PREPROCESSED ??

BI209 - SEVERITY 2
P/F CALL NOT PREPROCESSED ??

BI210 - SEVERITY 2
STRUCT SUBSCRIPT NOT PREPROCESSED ??

BI211 - SEVERITY 2
XXST NOT FOUND ??

BI213 - SEVERITY 2
NEGATIVE NEST LEVEL

BI214 - SEVERITY 2
TINT REF NOT PREPROCESSED ??

BI215 - SEVERITY 2
VAR REF NOT PREPROCESSED: ??

BI216 - SEVERITY 2
GET_STMT_VARS FOUND: ??

BI217 - SEVERITY 2
ILLEGAL INIT LIST OP

BI218 - SEVERITY 5
END OF STMT DATA: ??

BI219 - SEVERITY 5
STACK OVERFLOW WHILE FORMATTING VMEM CELLS

BI220 - SEVERITY 5
SAVE_VACS STACK OVERFLOW

BI221 - SEVERITY 5
WORD STACK OVERFLOW

BI222 - SEVERITY 5
END OF STMT DATA IN DECL: ??

BI223 - SEVERITY 2
NEGATIVE STORAGE SIZE

BI300 - SEVERITY 2
COMPILE-TIME INT/SCA ADDITION FAILURE

BI301 - SEVERITY 2
COMPILE-TIME INT/SCA SUBTRACTION FAILURE

BI302 - SEVERITY 2
COMPILE-TIME INT/SCA MULTIPLICATION FAILURE

BI303 - SEVERITY 2
COMPILE-TIME INT/SCA DIVISION FAILURE

BI304 - SEVERITY 2
LITERAL COLLAPSING ABANDONED

BI305 - SEVERITY 4
LITERAL TABLE OVERFLOW

BI306 - SEVERITY 4
STRUCTURE TEMPLATE USED WITHOUT XREF INDICATION

BI307 - SEVERITY 4
TEMPLATE STACK OVERFLOW

BI308 - SEVERITY 4
CSE TABLE OVERFLOW

BI309 - SEVERITY 4
(PREVIOUS) STATEMENT TOO LONG. COMPILATION ABANDONED

BI310 - SEVERITY 4
STATEMENT TOO LONG. COMPILATION ABANDONED

BI311 - SEVERITY 4
DO LEVEL EXCEEDED

BI400 - SEVERITY 4
SYT_REF_POOL OVERFLOW

BI401 - SEVERITY 4
VAC_REF_POOL OVERFLOW

BI402 - SEVERITY 4
STACK UNDERFLOW

BI403 - SEVERITY 4
ATTEMPT TO DECODE RAND ?? AS RATOR

BI404 - SEVERITY 4
ATTEMPT TO DECODE RATOR ?? AS RAND

BI405 - SEVERITY 4
STACK FRAME TYPE COMPILER ERROR

BI406 - SEVERITY 4
BLOCK CLOSE AND BLOCK OPEN LABELS ON STACK DIFFER

BI407 - SEVERITY 4
ATTEMPT TO DECODE RATOR AS RAND WHILE GENERATING TARGET INFORMATION

BI408 - SEVERITY 4
ATTEMPT TO DECODE RAND AS RATOR WHILE GENERATING TARGET INFORMATION

BI409 - SEVERITY 4
ATTEMPT TO TARGET ?? OPERATOR

BI410 - SEVERITY 4
ATTEMPT TO DECODE RATOR AS RAND WHILE TESTING STOP CONDITIONS FOR TARGETING

BI411 - SEVERITY 4
ATTEMPT TO DECODE RAND AS RATOR WHILE TESTING STOP CONDITIONS FOR TARGETING

BI412 - SEVERITY 4
COMPILATION ENDED IN AUXMATTER PHASE DUE TO DEBUG TOGGLE 134 BEING SET

BI500 - SEVERITY 2
HALMAT/FILE6 PHASING ERROR

BI501 - SEVERITY 2
?? BITS UNACCOUNTED FOR

BI504 - SEVERITY 2
LOOP IN OPERAND STACK POINTERS

BI505 - SEVERITY 2
INTERNAL COMPILER ERROR IN BRANCH_CONDENSE

BI506 - SEVERITY 2
EXTRA CODE RECORD, LHS = ??

BI507 - SEVERITY 2
SPLBL LOC NOT EQUAL TO 0 FOR CSECT ??

BI509 - SEVERITY 2
LABEL VERIFICATION ERROR FOR LBL# ??

BI510 - SEVERITY 5
HALMAT/FILE6 ERROR FOUND IN EMIT_ADDRS

BI511 - SEVERITY 4
THE COMPILER GENERATED AN INSTRUCTION WITH A NEGATIVE ?? REGISTER

BI512 - SEVERITY 4
DISPLACEMENT > 2047 IS INVALID FOR INDEXED RS TYPE INSTRUCTION

BI513 - SEVERITY 4
THE COMPILER ATTEMPTED TO GENERATE INCORRECT OBJECT CODE DUE TO TRUNCATION OF AN INDEX DURING ADJUSTMENT FOR AUTOMATIC INDEX ALIGNMENT

BI514 - SEVERITY 4
DISPLACEMENT > 65535 IS INVALID FOR EXTENDED RS TYPE INSTRUCTION

BI515 - SEVERITY 4
DISPLACEMENT > = 56 IS INVALID FOR ?? TYPE INSTRUCTION

BI516 - SEVERITY 4
COMPILER GENERATED AN INVALID BASE REGISTER FOR ?? INSTRUCTION

BI517 - SEVERITY 4
COMPILER IS INCORRECTLY UTILIZING A BASE REGISTER IN A STORE INSTRUCTION

BI518 - SEVERITY 4
CANNOT FIND A VALID BASE REGISTER TO LOAD VARIABLE.

BI700 - SEVERITY 5
VIRTUAL MEMORY PAGING AREA DEPLETED

BI701 - SEVERITY 5
BAD POINTER DETECTED BY PTR_LOCATE = ??

BI702 - SEVERITY 5
VIRTUAL MEMORY CAPACITY EXHAUSTED

BI703 - SEVERITY 5
REQUESTED VIRTUAL MEMORY CELL SIZE TOO LARGE

BN1 - SEVERITY 4
MAX NAME SCOPE NESTING DEPTH EXCEEDED
CODE BLOCKS (AND HENCE NAME SCOPES) MAY NOT BE NESTED TO A DEPTH OF MORE THAN 16 LEVELS.

BS1 - SEVERITY 4
MAXIMUM DEPTH OF DO...END GROUP NESTING EXCEEDED
NO MORE THAN 24 LEVELS OF DO...END GROUP NESTINGS ARE PERMITTED

BS100 - SEVERITY 2
DO GROUPS NESTED TOO DEEP
NO MORE THAN 16 LEVELS OF DO...END GROUP NESTINGS ARE PERMITTED.

BS101 - SEVERITY 2
FUNCTION CALLS NESTED TOO DEEP
NO MORE THAN 20 LEVELS OF FUNCTION INVOCATION ARE PERMITTED.

BS102 - SEVERITY 3
SUBPROGRAM STACK OVERFLOW
MORE THAN 255 CSECTS WERE GENERATED BY THE COMPILATION (THIS INCLUDES DATA, CODE, TEMPLATE CSECTS, BUT NOT EXTERNAL REFERENCES, E.G. LIBRARIES).

BS103 - SEVERITY 2
EXCEEDED ARGUMENT STACK SIZE
FUNCTION AND PROCEDURE INVOCATIONS MAY NOT INVOLVE MORE THAN 100 ARGUMENTS. THIS FIGURE INCLUDES ALL ARGUMENTS TO FUNCTIONS WHICH ARE THEMSELVES USED AS ARGUMENTS IN THE SUBROUTINE INVOCATION. THUS, CALL PROC1(A,B,FUNC(X,Y)) COUNTS AS HAVING 4 ARGUMENTS.

BS105 - SEVERITY 2
DATA STORAGE CAPACITY EXCEEDED
A TEMPORARY DATA ITEM WAS DECLARED THAT WOULD OCCUPY MORE THAN 2 MILLION BYTES (360) OR HALFWORDS (FC) STORAGE.

BS106 - SEVERITY 3
INDIRECT STACK OVERFLOW
SIMPLIFY. MAY POSSIBLY BE COMPILER ERROR. LOOK FOR "LOOP IN OPERAND STACK

POINTERS" MESSAGE AT END OF PHASE II LISTING.

BS107 - SEVERITY 3

CHARACTER LITERAL BUFFER OVERFLOW

INCREASE LITSTRINGS COMPILER OPTION AND RESUBMIT.

BS108 - SEVERITY 2

LITERAL PROCESSING FAILURE

ATTEMPT WAS MADE TO EMPLOY, AS AN INTEGER, A NUMBER WHICH IS NOT INTEGERIZABLE. INTEGERS MAY NOT TAKE VALUES GREATER THAN 2**32.

BS109 - SEVERITY 3

CONSTANT TABLE OVERFLOW

THE MAXIMUM NUMBER OF UNIQUE CONSTANTS WHICH MAY BE DEFINED IN A COMPILATION IS 2000.

BS110 - SEVERITY 2

TOO MANY EXTERNAL NAMES

NO MORE THAN 400 EXTERNAL NAMES ARE PERMITTED IN ONE COMPILATION.

BS111 - SEVERITY 2

EXCEEDED ON ERROR STACK SIZE

NO MORE THAN 100 ON ERROR STATEMENTS MAY BE POTENTIALLY ACTIVE AT ONE TIME.

BS112 - SEVERITY 2

STORAGE DESCRIPTOR STACK OVERFLOW

NO MORE THAN 100 TEMPORARIES MAY BE EMPLOYED AT ONE TIME. CAUTION: THE LIMIT FOR USER-DEFINED TEMPORARY VARIABLES WILL IN GENERAL BE LESS THAN THIS, BECAUSE THE COMPILER GENERATES ITS OWN TEMPORARIES AS WELL, AND THESE COUNT TOWARDS THE LIMIT OF 100.

BS113 - SEVERITY 2

EXCEEDED TEMPORARY SPACE

SIMPLIFY: DON'T DECLARE TOO MANY TEMPORARY VARIABLES, DECLARE SOME AS PERMANENT.

BS115 - SEVERITY 2

ARRAYNESS PUSHED TOO MANY LEVELS

SIMPLIFY: AN ARRAYED EXPRESSION INVOLVING NESTED FUNCTION CALLS ? WITH ARRAYED PARAMETERS IS NESTED MORE THAN 5 LEVELS. BREAK IT UP INTO TWO OR MORE STATEMENTS.

BS116 - SEVERITY 2

SIMPLIFY: EXCEEDED ARRAY TEMPORARY SIZE STACK.

TOO MANY DIFFERENT ARRAYNESSES AT WORK IN ONE STATEMENT. BREAK IT UP INTO TWO OR MORE DIFFERENT STATEMENTS.

BS117 - SEVERITY 2

TEMPORARY ?? NOT ADDRESSABLE

AN ADDRESSING DIFFICULTY. SIMPLIFYING OR EVEN CHANGING THE ORDER IN WHICH TEMPORARY VARIABLES ARE DEFINED MAY HELP. REFER TO SECTION 8.13 USER NOTES FOR FURTHER INFORMATION.

BS118 - SEVERITY 2

EXCEEDED SHAPING FUNCTION DIMENSION STACK

NO MORE THAN 20 DIFFERENT DIMENSIONS FOR INTEGER/SCALAR SHAPING FUNCTIONS MAY BE MENTIONED IN ONE STATEMENT. THE SOLUTION HERE IS TO BREAK UP THE STATEMENT.

BS119 - SEVERITY 2

EXCEEDED ARRAYNESS STACK SIZE

SIMPLIFY: TOO MANY DIFFERENT ARRAYNESSES AT WORK IN ONE STATEMENT. BREAK IT UP INTO TWO OR MORE STATEMENTS.

BS120 - SEVERITY 3

DATA STORAGE CAPACITY EXCEEDED

A DATA ITEM WAS DECLARED THAT WOULD OCCUPY MORE THAN 2 MILLION BYTES (360) OR HALFWORDS (FC) OF STORAGE.

BS122 - SEVERITY 2

INDIRECT STACK USAGE CONFLICT

INTERNAL COMPILER FAILURE.

BS123 - SEVERITY 1

UNSUPPORTED REMOTE ADDRESSING FORM (??) ENCOUNTERED.

THE BS123 ERROR MAY BE EMITTED UNDER THE FOLLOWING 3 CONDITIONS TO INDICATE AN UNSUPPORTED REMOTE ADDRESSING FORM:

1) THE VARIABLE IS REMOTE (EITHER AGGREGATE OR NON-AGGREGATE) AND HAS INSTRUCTION MODIFIER CODE ASSOCIATED WITH IT.
2) THE VARIABLE IS A REMOTE AGGREGATE WITHOUT INDEXING.
3) THE VARIABLE IS A REMOTE NON-AGGREGATE WITH INDEXING.

THE BS123 ERROR MESSAGE HAS BEEN UPDATED TO INDICATE WHICH OF THE ABOVE CASES (1, 2, OR 3) HAS BEEN ENCOUNTERED.

BS130 - SEVERITY 2

OFF PAGE REFERENCE NOT FOUND

SAME AS ABOVE.

BS131 - SEVERITY 2

TOO MANY OFF PAGE ENTRIES

SAME AS ABOVE.

BS2 - SEVERITY 3

INDIRECT PARSE STACK SIZE EXCEEDED

SIMPLIFY: E.G., TOO MANY PARENTHESIZED OPERATIONS.

BS3 - SEVERITY 4

PARSE STACK OVERFLOW

SIMPLIFY.

BS4 - SEVERITY 3

CURRENT ARRAYNESS STACK SIZE EXCEEDED

SIMPLIFY: TOO MANY DIFFERENT ARRAYNESSES AT WORK IN ONE STATEMENT. BREAK INTO TWO OR MORE STATEMENTS.

BS5 - SEVERITY 2

MAXIMUM FUNCTION NESTING DEPTH EXCEEDED

FUNCTIONS MAY NOT BE NESTED TO A DEPTH OF MORE THAN 10 LEVELS.

BS6 – SEVERITY 1

DUE TO INTERNAL COMPILER LIMITATIONS, SUBSCRIPTS OR EXPONENTS ARE INCORRECTLY PRINTED, INTERLEAVING COMMENT LINES MAY AVIOD THIS PROBLEM

SUBSCRIPTS OR EXPONENTS WILL NOT BE PRINTED CORRECTLY FOR STATEMENTS WITH OVER 250 TOKENS WHEN SUBSCRIPTS OR EXPONENTS FALL AT THE END OF 100 CHARACTER OUTPUT LINES AND IN TOKEN POSITION 248 DUE TO SIZE RESTRICTIONS OF THE STATEMENT STACK (SEE SECTION 8.13)

BS7 – SEVERITY 3

MAXIMUM SUBSCRIPT OR EXPONENT NESTING DEPTH EXCEEDED

SUBSCRIPTS OR EXPONENTS MAY NOT EXCEED 11 LEVELS

BT2 - SEVERITY 2

STATEMENT NODE STACK OVERFLOW IN CURRENT STATEMENT: SIMULATION TABLES EMITTED BY PHASE III WILL LACK INFORMATION CONCERNING LABELS ATTACHED TO STATEMENT, OR VARIABLES CHANGED BY IT.

THE SIMULATION DATA FILE TABLES PRODUCED BY PHASE III OF THE COMPILER WILL BE INCOMPLETE.

BT3 - SEVERITY 4

LITERAL TABLE DATA OVERFLOW

NO MORE THAN 32,767 UNIQUE LITERALS OF ANY TYPE MAY BE REFERRED TO WITHIN ONE COMPILATION UNIT.

BT4 - SEVERITY 4

LITERAL TABLE STRING OVERFLOW

INCREASE "LITSTRINGS" COMPILER OPTION AND RECOMPILE

BT7 - SEVERITY 3

INITIAL LIST STORAGE CAPACITY EXCEEDED

NO MORE THAN 32,767 ITEMS MAY BE REFERRED TO IN AN INITIAL LIST.

BX1 - SEVERITY 3

SYT_CLASS = 0 FOR ??

EITHER A SPURIOUS ERROR CAUSED BY PREVIOUS ERROR RECOVERY (PARTICULARLY FROM A SYNTAX ERROR ENCOUNTERED WITHIN A DECLARE STATEMENT), OR ELSE AN INTERNAL COMPILER FAILURE.

BX100 - SEVERITY 2

?? NOT ADDRESSABLE

SIMPLIFY. TOO MUCH DATA DECLARED IN COMPOOLS.

BX101 - SEVERITY 2

LEVEL MISMATCH ON PROC/FUNC CALL

THIS AND THE FOLLOWING ERRORS MAY APPEAR AS SPURIOUS BY-PRODUCTS OF ATTEMPTS AT RECOVERY FROM PREVIOUS ERRORS, AND IN THAT CASE SHOULD BE IGNORED. IN THE ABSENCE OF OTHER ERRORS, HOWEVER, THEY REPRESENT INTERNAL COMPILER FAILURES.

BX102 - SEVERITY 2
ARRAY LEVEL MISMATCH
SEE BX101

BX103 - SEVERITY 2
INDEX STACK USAGE INCONSISTENT
SEE BX101

BX104 - SEVERITY 2
BASE STACK USAGE INCONSISTENT
SEE BX101

BX105 - SEVERITY 2
UNMATCHED DO CASE ENDING
SEE BX101

BX106 - SEVERITY 2
UNMATCHED DO CASE LABEL
SEE BX101

BX107 - SEVERITY 2
UNMATCHED DO WHILE ENDING
SEE BX101

BX108 - SEVERITY 2
UNMATCHED DO FOR ENDING
SEE BX101.

BX109 - SEVERITY 2
DO FOR ENDING INCOMPLETE DUE TO PREVIOUS ERROR RECOVERY
SEE BX101.

BX110 - SEVERITY 2
ARRAY ENDING MISMATCH
SEE BX101.

BX111 - SEVERITY 2
LEVEL MISMATCH ON PROC/FUNC/IO ARGUMENT
SEE BX101.

BX112 - SEVERITY 2
MISMATCH CLOSING
SEE BX101.

BX113 - SEVERITY 2
ERROR FUNCTION LOOKUP FAILED -- ERRORCODE ??
MOST LIKELY THIS IS THE RESULT OF A FAILURE TO EMPLOY AN UP TO DATE ERROR LIBRARY.
SEE BX101.

BX150 - SEVERITY 1
NOT ALL DATA MOVED VIA MVH ALGORITHM
COMPILER GENERATED LOAD, STORE SEQUENCES TO MOVE DATA.

BX2 - SEVERITY 3
FUNC_TOKEN = 0

BX200 - SEVERITY 2

SYT_SCOPE> MAX_SCOPE#

INTERNAL COMPILER FAILURE

BX5 - SEVERITY 3

EXT_ARRAY OVERFLOW

SIMPLIFY: TOO MANY ARRAYED VARIABLES DECLARED WITH COMPLEX AND DIFFERING DIMENSIONS. THE TOTAL NUMBER OF DIFFERENTLY-SHAPED ARRAYS DEFINED, PLUS THE TOTAL NUMBER OF DIMENSIONS DEFINED FOR THESE ARRAYS MUST BE LESS THAN 300.

B1 - SEVERITY 4

INSUFFICIENT CORE AVAILABLE

INCREASE REGION SIZE AND RE-RUN

B100 - SEVERITY 2

STATEMENT CONTAINS PHASE I ERROR

B102 - SEVERITY 2

UNIMPLEMENTED FEATURE OF HAL/S CALLED FOR

B2 - SEVERITY 4

INLINE FUNCTION MAY NOT BE IN A SUBSCRIPT OR EXPONENT. THIS ERROR IS IRRECOVERABLE.

SEE SPEC-SECTION 11.2.1

B3 - SEVERITY 4

PRIMARY INPUT DATASET EMPTY

MAKE SURE THAT THE CORRECT DATA SET (CONTAINING THE HAL/S SOURCE) IS DEFINED ON THE SYSIN DD CARD.

ERROR MESSAGES FOR MAJOR CLASSIFICATION C

CLASSIFICATION "C" ERRORS DEAL WITH COMPARISONS

C0 - SEVERITY 2

ARGUMENTS IN NAME COMPARISON ARE INCOMPATIBLE.

NAME VARIABLES MAY ONLY BE COMPARED IF THEY MATCH PRECISELY AS TO TYPE, ETC.
SPEC-SECTION 11.4.8

C1 - SEVERITY 2

?? COMPARISONS MAY ONLY BE = OR NOT=

COMPARISONS BETWEEN ENTIRE ARRAYS, MATRICES, VECTORS AND BITS MAY ONLY BE IN TERMS OF EQUALITY OR INEQUALITY

C2 - SEVERITY 2

ARRAYED COMPARISONS ARE RESTRICTED TO = OR NOT=

LESS THAN, GREATER THAN, ETC. WOULD GIVE NO UNAMBIGUOUS RESULT HERE.

C3 - SEVERITY 2

TREE ORGANIZATIONS OF STRUCTURES COMPARED DO NOT MATCH

STRUCTURES CAN ONLY BE COMPARED IF THEY POSSESS IDENTICAL ORGANIZATIONS.

C4 - SEVERITY 2

ONLY ONE OPERAND OF COMPARISON IS A NAME PSEUDO-FUNCTION OR NULL.

THE NAME PSEUDO-FUNCTION MUST OCCUR ON BOTH SIDES OF THE COMPARISON: "NAME(L)=R" AND "L=NAME(R)" ARE BOTH ILLEGAL, UNLESS L OR R REPRESENT THE NULL NAME VALUE.
SPEC-SECTION 11.4.8

C5 - SEVERITY 1

BROKEN DOUBLE COMPARE INSTRUCTION REPLACED WITH SINGLE COMPARE.

THE COMPILER NEEDS TO GENERATE A DOUBLE PRECISION COMPARE INSTRUCTION (CED OR CEDR) BUT INSTEAD GENERATES A SINGLE PRECISION COMPARE (CE OR CER) BECAUSE THE DOUBLE PRECISION COMPARE INSTRUCTIONS ARE CURRENTLY BROKEN ON THE AP101S.

ERROR MESSAGES FOR MAJOR CLASSIFICATION D

CLASSIFICATION "D" ERRORS ARE RELATED TO DATA DECLARATIONS

DA0 - SEVERITY 2

CONFLICTING ATTRIBUTE SPECIFIED WITH THE LATCHED ATTRIBUTE

THE LATCHED ATTRIBUTE APPLIES ONLY TO EVENT VARIABLES; THE REMOTE ATTRIBUTE AND PRECISION SPECIFICATIONS ARE ILLEGAL FOR EVENT VARIABLES. SPEC-SECTION 4.5

DA1 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR BIT OR BOOLEAN DATA TYPE

THE LATCHED ATTRIBUTE IS ILLEGAL FOR BIT AND BOOLEAN DATA TYPES. PRECISION SPECIFICATIONS ARE AS WELL. SPEC-SECTION 4.5

DA10 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR STRUCTuRE DATA TYPE

THE ATTRIBUTES, ALIGNED AND RIGID, ARE ILLEGAL FOR MAJOR STRUCTURES AS ARE PRECISION SPECIFICATIONS.

SPEC-SECTION 4.5

DA11 - SEVERITY 2

CONFLICT BETWEEN ATTRIBUTES AND FUNCTION OR LABEL TYPE SPECIFICATION - ATTRIBUTES IGNORED

PROCEDURE, TASK, OR FUNCTION DECLARATIONS MAY NOT POSSESS ANY ATTRIBUTES. SPEC-SECTION 4.5

DA12 - SEVERITY 2

RIGID ATTRIBUTE MAY ONLY APPEAR IN A STRUCTURE TEMPLATE OR COMPOOL BLOCK DEFINITION

DA13 - SEVERITY 2

?? MAY NOT BE THE NAME OF A FUNCTION

NAME VARIABLES MAY NOT BE DECLARED TO BE THE NAME OF FUNCTIONS. SPEC-SECTION 11.4.1

DA14 - SEVERITY 2

?? MAY NOT BE THE NAME OF A PROCEDURE

NAME VARIABLES MAY NOT BE DECLARED TO BE THE NAME OF PROCEDURES. SPEC-SECTION 11.4.1

DA15 – SEVERITY 2

DATA CANNOT BE DECLARED WITH BOTH THE AUTOMATIC AND REMOTE ATTRIBUTES

NON-NAME DATA DECLARED IN A REENTRANT PROCEDURE WITH THE AUTOMATIC ATTRIBUTE CAN NOT ALSO BE REMOTE, BECAUSE IT MUST BE ALLOCATED ON THE RUNTIME STACK.

DA2 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR CHARACTER DATA TYPE

PRECISION SPECIFICATION AND THE LATCHED ATTRIBUTE ARE ILLEGAL FOR CHARACTER DATA TYPES. SPEC-SECTION 4.5

DA20 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR MINOR STRUCTURE ??

THE REMOTE, STATIC/AUTOMATIC, ACCESS, LOCKED, OR LATCHED ATTRIBUTES ARE ALL ILLEGAL FOR MINOR-STRUCTURES.

DA21 - SEVERITY 2

AN ARRAY SPECIFICATION IS NOT ALLOWED FOR THE MINOR STRUCTURE ??

SEE SPEC-SECTION 4.5

DA22 - SEVERITY 2

ILLEGAL ATTRIBUTE SUPPLIED ON TEMPLATE NAME ??

DENSE, ALIGNED, AND RIGID ARE THE ONLY ATTRIBUTES PERMITTED IN STRUCTURE TEMPLATES. SPEC-SECTION 4.3

DA23 - SEVERITY 2

ILLEGAL ATTRIBUTE FOR THE STRUCTURE TERMINAL ??

DENSE, ALIGNED, RIGID, PRECISION AND ARRAYNESS ARE THE ONLY ATTRIBUTES PERMITTED FOR STRUCTURED TERMINALS.

SPEC-SECTION 4.5

DA24 - SEVERITY 2

FACTORED AND NON-FACTORED ATTRIBUTE SPECIFICATIONS FOR ?? DISAGREE; THE NON-FACTORED ATTRIBUTES WILL BE GIVEN PRECEDENCE.

FOR EXAMPLE, "DECLARE SCALAR AUTOMATIC INITIAL(5), S,X,M STATIC" IS AMBIGUOUS AS TO M'S INITIALIZATION AND ATTRIBUTES. IN THIS CASE, STATIC WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED ATTRIBUTES.

DA25 - SEVERITY 2

CONTRADICTORY PAIR OF ATTRIBUTES SUPPLIED - FIRST APPEARING ATTRIBUTE WILL BE USED

STATIC AND AUTOMATIC, DENSE AND ALIGNED, SINGLE AND DOUBLE ARE ALL MUTUALLY EXCLUSIVE PAIRS OF ATTRIBUTES. USE, AT MOST, ONE FROM EACH PAIR.

DA26 - SEVERITY 2

DECLARATIONS OF THE FORM: <TEMPLATE NAME>-STRUCTURE MAY NOT HAVE AN ARRAY SPECIFICATION

USE THE COPINESS ATTRIBUTE INSTEAD.

DA3 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR MATRIX DATA TYPE

LATCHED IS ILLEGAL FOR MATRICES.

DA4 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR VECTOR DATA TYPE

LATCHED IS ILLEGAL FOR VECTORS.

DA5 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR SCALAR DATA TYPE

LATCHED IS ILLEGAL FOR SCALARS.

DA6 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR INTEGER DATA TYPE

LATCHED IS ILLEGAL FOR INTEGERS.

DA9 - SEVERITY 2

ILLEGAL ATTRIBUTE SPECIFIED FOR EVENT DATA TYPE

REMOTE, DENSE, ALIGNED, AND RIGID ARE ALL ILLEGAL FOR EVENTS. ALSO, UNLATCHED EVENTS CANNOT BE INITIALIZED.

DC1 - SEVERITY 2

DECLARATION CONTAINS BOTH LABEL TYPE AND DATA TYPE SPECIFICATION - LABEL TYPE IGNORED.

AN ITEM DEFINED TO BE A PROCEDURE OR TASK MAY NOT ALSO BE DEFINED AS AN INTEGER OR VECTOR. "DECLARE P PROCEDURE BOOLEAN", FOR INSTANCE, IS ILLEGAL. SPEC-SECTION 4.5

DC2 - SEVERITY 2

THE ATTRIBUTES STATIC AND AUTOMATIC MAY NOT BE SPECIFIED IN COMPOOL DECLARATIONS.

DC4 - SEVERITY 2

FACTORED AND NON-FACTORED TYPE SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED TYPE SPECIFICATION WILL BE USED

FOR EXAMPLE, "DECLARE SCALAR, S, X, I INTEGER" IS AMBIGUOUS AS TO I'S TYPE. IN THIS CASE, INTEGER WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTOR TYPE SPECIFICATIONS.

DC5 - SEVERITY 2

ACCESS ATTRIBUTES MAY ONLY BE QUALIFIED IN COMPOOL DECLARATIONS.

SEE SPEC-SECTION 4.5

DD1 - SEVERITY 2

ILLEGAL ARRAY DIMENSION SPECIFICATION

ARRAYNESS MUST BE * OR 1 < ARRAYSIZE <= 32767 AND THE TOTAL NUMBER OF ELEMENTS MAY NOT BE GREATER THAN 32767.
SPEC-SECTION 4.5

DD10 - SEVERITY 2

* ARRAY SIZE IS ILLEGAL FOR ?? - ARRAY SIZE OF 2 ASSUMED.

THE (*) SPECIFICATION IS ONLY PERMITTED IN THE DECLARATION OF FORMAL PARAMETERS FOR FUNCTIONS AND PROCEDURES.

DD11 - SEVERITY 2

ILLEGAL FORM OF STRUCTURE DIMENSION SPECIFICATION

THE CORRECT FORM FOR SPECIFYING STRUCTURE COPIES, IS: "DECLARE ZIP Q-STRUCTURE(50);" COPYNESS MUST BE: * OR 1 < COPY <= 32767 AND THE TOTAL NUMBER OF ELEMENTS MULTIPLIED BY THE NUMBER OF COPIES MAY NOT BE GREATER THAN 32767.

DD12 - SEVERITY 2

ILLEGAL CONTEXT FOR SPECIFICATION OF STRUCTURE COPIES

STRUCTURE COPIES SHOULD BE SPECIFIED IN A DECLARE STATEMENT, NOT IN THE STRUCTURE TEMPLATE. IN ADDITION, FUNCTIONS OF STRUCTURE TYPE MUST NOT HAVE COPYNESS.

DD3 - SEVERITY 3

TOO MANY DIMENSIONS IN ARRAY

ARRAYS MAY HAVE NO MORE THAN THREE DIMENSIONS.

DD4 - SEVERITY 2

INVALID MATRIX DIMENSION SPECIFICATION; A DIMENSION OF 3 IS ASSUMED

THE DIMENSIONS SPECIFIED FOR A MATRIX MUST EACH BE AN UNARRAYED INTEGER OF SCALAR EXPRESSION COMPUTABLE AT COMPILE TIME, AND RESULTING IN DIMENSIONS BETWEEN 2 AND 64 AFTER CONVERSION TO AN INTEGER.

DD5 - SEVERITY 2

INVALID VECTOR LENGTH SPECIFICATION; A 3-VECTOR IS ASSUMED

THE LENGTH SPECIFIED FOR A VECTOR MUST BE AN UNARRAYED INTEGER OR SCALAR EXPRESSION COMPUTABLE AT COMPILE TIME, AND RESULTING (AFTER CONVERSION TO AN INTEGER) IN A LENGTH BETWEEN 2 AND 64.

DD6 - SEVERITY 2

ONLY SINGLE DIMENSION ARRAYS MAY USE THE * TO DENOTE UNKNOWN LENGTH

DD7 - SEVERITY 2

A * MAY NOT BE USED TO SPECIFY VECTOR LENGTH; A 3-VECTOR IS ASSUMED

DD8 - SEVERITY 2

* STRUCTURE COPY NOTATION IS ILLEGAL FOR ?? - STRUCTURE COPY SIZE OF 2 ASSUMED.

THE (*) COPY SPECIFICATION IS ONLY PERMITTED ON THE DECLARATION OF FORMAL PARAMETERS FOR FUNCTIONS AND PROCEDURES. SPEC-SECTION 4.7

DD9 - SEVERITY 2

A * MAY NOT BE USED TO SPECIFY A MATRIX DIMENSION; A DIMENSION OF 3 IS ASSUMED

DI1 - SEVERITY 2

REPEAT FACTOR IN INITIALIZATION HAS NO LEGAL VALUE COMPUTABLE AT COMPILE TIME

"INITIAL (N K)" IS LEGAL ONLY IF N IS COMPUTABLE AT COMPILE TIME. SPEC-SECTION 4.8

DI10 - SEVERITY 2

TOO MANY ELEMENTS SUPPLIED IN INITIAL LIST FOR ??

THE SIZE OF THE INITIAL LIST SHOULD NOT BE GREATER THAN THE DECLARED SIZE OF THE VARIABLE.

DI100 - SEVERITY 2

INITIALIZATION DATA TYPE MISMATCH

THE DECLARED VARIABLE DATA TYPE MUST MATCH THE TYPE OF DATA SUPPLIED IN THE INITIAL LIST, OR AT LEAST AN IMPLICIT CONVERSION MUST BE POSSIBLE.

DI101 - SEVERITY 2

NULL ONLY LEGAL NAME CONSTANT

NAME VARIABLES MAY NOT BE INITIALIZED TO ARITHMETIC LITERALS, ALTHOUGH THE INITIALIZATION TO A NAME PSEUDO-FUNCTION IS LEGAL IF THE ADDRESS OF ITS ARGUMENT IS KNOWN AT COMPILE TIME.

DI102 - SEVERITY 1

INITIAL STRING TOO LONG

IF A CHARACTER OR BIT STRING IN THE INITIAL LIST IS LONGER THAN THE DECLARED LENGTH OF THE CHARACTER OR BIT VARIABLE, TRUNCATION WILL OCCUR.

DI103 - SEVERITY 2

INITIAL VALUE TOO LARGE

CANNOT INITIALIZE AN INTEGER WITH A VALUE GREATER THAN 16 BITS. NOTE THAT THIS ERROR DOES NOT APPLY TO NON-AGGREGATE CONTANTS (SEE LANGUAGE SPECIFICATION SECTION 4.8).

DI104 - SEVERITY 2

STATIC INITIALIZATION OF DYNAMIC NAME

NAME VARIABLES, IF STATIC, MAY BE INITIALIZED ONLY TO ADDRESSES WHICH ARE COMPUTABLE AT COMPILE TIME.

DI105 - SEVERITY 2

INITIAL VALUE MUST BE LITERAL VALUE

DI106 - SEVERITY 2

ATTEMPT TO EQUATE TO DYNAMICALLY ADDRESSABLE SYMBOL

THIS OCCURS WHEN EQUATING AN EXTERNAL SYMBOL TO AN ADDRESS UNCOMPUTABLE AT COMPILE TIME.

DI107 - SEVERITY 1

ATTEMPT TO INITIALIZE A NON-REMOTE NAME WITH A REMOTE VARIABLE. 16 BIT NAME VARIABLES MUST POINT TO NON-REMOTE DATA.

DI108 - SEVERITY 2

THE TYPE OF THE INITIAL LIST NAME PSEUDO-FUNCTION MUST MATCH EXACTLY THE TYPE OF THE VARIABLE BEING DECLARED.

DI109 - SEVERITY 2

THE ATTRIBUTE OF THE INITIAL LIST NAME PSEUDO-FUNCTION MUST MATCH EXACTLY THE TYPE OF THE VARIABLE BEING DECLARED.

DI110 - SEVERITY 2

ARRAYNESS/MULTI-COPINESS CONFLICT BETWEEN THE TYPE OF THE VARIABLE BEING DECLARED AND THE VARIABLE IN THE INITIAL STATEMENT.

DI11 - SEVERITY 2

THE VARIABLE ?? USED IN A COMPILE-TIME EXPRESSION OR AS THE TARGET OF AN EQUATE STATEMENT HAS NOT BEEN PREVIOUSLY DEFINED

THE VALUE OF A VARIABLE USED IN AN INITIALIZING EXPRESSION MUST BE AVAILABLE AT COMPILE TIME, AND IN FACT, THE VARIABLE MUST HAVE BEEN DECLARED AND INITIALIZED (USING THE CONSTANT FORM) PRIOR TO ITS USE.

DI12 - SEVERITY 2

FORMAL PARAMETER ?? POSSESSES ILLEGAL ATTRIBUTES CONCERNING INITIALIZATION - ATTRIBUTES IGNORED.

FORMAL PARAMETERS IN FUNCTIONS AND PROCEDURES MAY NOT BE INITIALIZED. SPEC-SECTION-4.5

DI13 - SEVERITY 2

LABEL OR FUNCTION ?? POSSESSES ILLEGAL ATTRIBUTES CONCERNING INITIALIZATION - ATTRIBUTES IGNORED.

FUNCTIONS MAY NOT BE INITIALIZED. SPEC-SECTION-4.5

DI14 - SEVERITY 2

IN AN INITIAL LIST, THE ARGUMENT OF A NAME PSEUDO-FUNCTION MAY NOT POSSESS A PRECISION QUALIFIER.

PROGRAMMER'S GUIDE-SECTION 28

DI15 - SEVERITY 2

IN AN INITIAL LIST, THE ARGUMENT OF A NAME PSEUDO-FUNCTION MAY NOT POSSESS ARRAYNESS.

DI16 - SEVERITY 2

IN AN INITIAL LIST, THE ARGUMENT OF A NAME PSEUDO-FUNCTION MAY NOT POSSESS THE NAME ATTRIBUTE.

PROGRAMMER'S GUIDE-SECTION 28

DI17 - SEVERITY 2

INITIAL VALUE TOO LARGE

CANNOT INITIALIZE AN INTEGER CONSTANT WITH A VALUE GREATER THAN 32 BITS. (THIS ERROR ONLY APPLIES TO NON-AGGREGATE CONSTANTS).

DI18 - SEVERITY 1

IN THE DECLARATION OF ?? , THE SIZE OF THE CHARACTER CONSTANT IS LARGER THAN THE DECLARED SIZE. THE CONSTANT WILL BE TRUNCATED TO THE DECLARED SIZE.

DI2 - SEVERITY 3

IMPLIED NUMBER OF ELEMENTS IN INITIAL LIST EXCEEDS COMPILER LIMIT.

NO MORE THAN 32767 DATA ITEMS ARE ALLOWED IN AN INITIAL LIST.

DI21 - SEVERITY 1

A REMOTE COMPOOL VARIABLE IS THE OBJECT OF A NAME INITIAL STATEMENT. MAKE SURE THE NAME VARIABLE IS DECLARED AS NAME REMOTE.

DI3 - SEVERITY 2

EXPRESSION IN INITIAL LIST IS NOT COMPUTABLE AT COMPILE TIME. ALTHOUGH BIT CONSTANTS ARE COMPUTABLE AT COMPILE TIME, THEY ARE NOT AVAILABLE IN THIS COMPILER IMPLEMENTATION.

SEE SPEC, APPENDIX F.

DI4 - SEVERITY 2

INITIALIZATION OF ?? HAS ILLEGAL TERMINATING * : NUMBER OF INITIAL VALUES MATCHES TOTAL NUMBER OF ELEMENTS

THE NUMBER OF VALUES SPECIFIED IN AN INITIAL LIST ENDING IN (*) MUST BE STRICTLY LESS THAN THE NUMBER OF VALUES IMPLIED IN THE DECLARE STATEMENT. SPEC-SECTION 4.8

DI5 - SEVERITY 2

TOO FEW ELEMENTS SUPPLIED IN INITIAL LIST FOR ??

IF PARTIAL INITIALIZATION IS DESIRED, THE LIST SHOULD END WITH A (*).

DI6 - SEVERITY 2

ILLEGALLY-TYPED INITIAL VALUE--INITIALIZATION OF ?? EXPECTS A VALUE OF CHARACTER TYPE

SEE SPEC-SECTION 4.8

DI7 - SEVERITY 2

ILLEGALLY-TYPED INITIAL VALUE--INITIALIZATION OF ?? EXPECTS A VALUE OF BIT TYPE

SEE SPEC-SECTION 4.8

DI8 - SEVERITY 2

ILLEGALLY-TYPED INITIAL VALUE--INITIALIZATION OF ?? EXPECTS A VALUE OF INTEGER OR SCALAR TYPE

SEE SPEC-SECTION 4.8

DI9 - SEVERITY 2

THE DECLARATION OF ?? HAS BOTH FACTORED AND UNFACTORED INITIAL/CONSTANT ATTRIBUTES; THE UNFACTORED ATTRIBUTES WILL BE USED

FOR EXAMPLE, "DECLARE INTEGER INITIAL (2), I,J,K INITIAL (0)" IS AMBIGUOUS AS TO WHAT THE

INITIAL VALUE OF K IS, IN THIS CASE, INITIAL (0) WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED INITIALIZATIONS.

DL1 - SEVERITY 2

?? DOES NOT APPEAR IN A COMPOOL OR PROGRAM DECLARATION, AND IS NOT AN ASSIGN PARAMETER: THE LOCKING ATTRIBUTE SPECIFIED IS THEREFORE ILLEGAL.

THE CASES MENTIONED ARE THE ONLY ONES IN WHICH LOCKS MAY BE USED.

DL2 - SEVERITY 2

THE LOCKED ATTRIBUTE MAY NOT BE USED IN CONJUNCTION WITH THE CONSTANT ATTRIBUTE.

DL3 - SEVERITY 2

ILLEGAL LOCK GROUP NUMBER SPECIFIED

LOCK NUMBERS MUST BE IN THE RANGE FROM 1 TO 15.

DN1 - SEVERITY 2

PROGRAM OR PROCEDURE DECLARATION FOR ?? IS INVALID.

PROGRAMS AND PROCEDURES MAY NOT BE DECLARED. HOWEVER, NAME VARIABLES THAT POINT TO PROGRAMS MAY BE. FOR EXAMPLE, BOTH "DECLARE PROG PROGRAM" AND "DECLARE PROC PROCEDURE" ARE ILLEGAL, BUT "DECLARE NAME PROG PROGRAM" IS LEGAL.

DN2 - SEVERITY 2

THE NAME ATTRIBUTE MAY NOT BE USED IN THE DECLARATION OF TEMPORARIES -- ATTRIBUTE IGNORED.

SEE SPEC-SECTION 11.4.3

DQ1 - SEVERITY 2

FIRST NODE DECLARED IN TEMPLATE MUST BE AT LEVEL 1

SEE PROGRAMMER'S GUIDE-SECTION 19 FOR INFORMATION ON HOW TO FORM STRUCTURE TEMPLATES.

DQ10 - SEVERITY 2

STRUCTURE ?? MAY NOT BE A TEMPORARY SINCE ITS TEMPLATE CONTAINS AT LEAST ONE TERMINAL NODE WITH THE NAME ATTRIBUTE.

SEE SPEC-SECTION 11.4.3

DQ100 - SEVERITY 2

MALFORMED TEMPLATE, WALK INHIBITED

SEE PROGRAMMER'S GUIDE-SECTION 19 FOR INFORMATION ON HOW TO FORM STRUCTURE TEMPLATES. THIS ERROR PROBABLY IS A SPURIOUS ONE, RESULTING FROM PREVIOUS ERROR RECOVERY ACTIONS.

DQ101 - SEVERITY 2

TREE ORGANIZATIONS OF STRUCTURES ARE NOT IDENTICAL

STRUCTURE TREE ORGANIZATIONS MUST MATCH. SEE SPEC-SECTIONS 4.3, 6.4, 7.4, 11.4.1

DQ102 - SEVERITY 2

STRUCTURE NODE SIZE CONFLICT

EITHER A SPURIOUS ERROR RESULTING FROM AN EARLIER ERROR RECOVERY, OR ELSE AN INTERNAL COMPILER FAILURE.

DQ2 - SEVERITY 2

ILLEGAL SEQUENCE OF LEVEL NUMBERS IN TEMPLATE

NUMBERING OF LEVELS MUST FOLLOW CORRECT CONVENTIONS. ALL NODES MUST BE NUMBERED, NO LEVELS MAY BE SKIPPED, ETC. PROGRAMMER'S GUIDE-SECTION 19

DQ3 - SEVERITY 2

NAME OF STRUCTURE TEMPLATE CAUSES UNQUALIFICATION IN AN ILLEGAL CONTEXT

FOR INSTANCE: STRUCTURES DEFINED AS DATA ITEMS WITHIN A STRUCTURE TEMPLATE MAY NOT BE UNQUALIFIED; UNQUALIFIED STRUCTURES MAY NOT HAVE STRUCTURE TERMINALS WHOSE NAMES ARE NOT UNIQUE WITHIN THE COMPILATION. PROGRAMMER'S GUIDE-SECTION 19

DQ4 - SEVERITY 2

STRUCTURE ?? CANNOT BE UNQUALIFIED - STRUCTURE TEMPLATE IS ALREADY USED BY AN UNQUALIFIED STRUCTURE

ONLY ONE UNQUALIFIED STRUCTURE MAY BE DEFINED FOR EACH STRUCTURE TEMPLATE. PROGRAMMER'S GUIDE-SECTION 19

DQ5 - SEVERITY 2

STRUCTURE ?? CANNOT BE UNQUALIFIED - STRUCTURE TEMPLATE IS NOT IN SAME NAME SCOPE

UNQUALIFIED STRUCTURES MAY ONLY BE DECLARED WITHIN THE SAME NAME SCOPE AS THE STRUCTURE TEMPLATE THEY REFERENCE. PROGRAMMER'S GUIDE-SECTION 19

DQ6 - SEVERITY 2

STRUCTURE ?? CANNOT BE UNQUALIFIED - STRUCTURE TEMPLATE CONTAINS A REFERENCE TO ANOTHER STRUCTURE TEMPLATE.

NO STRUCTURES ARE PERMITTED TO BE NESTED WITHIN UNQUALIFIED STRUCTURES: I.E. NO NODE OF THE UNQUALIFIED STRUCTURE MAY ITSELF BE A STRUCTURE. PROGRAMMER'S GUIDE-SECTION 19

DQ7 - SEVERITY 2

THE DECLARED NAME ?? DUPLICATES THE NAME OF A NODE OF THE TEMPLATE OF AN UNQUALIFIED STRUCTURE PREVIOUSLY DECLARED IN THE SAME NAME SCOPE: THIS CAUSES THE UNQUALIFICATION TO BECOME ILLEGAL.

PROGRAMMER'S GUIDE-SECTION 19

DQ8 - SEVERITY 2

STRUCTURE ?? CANNOT BE UNQUALIFIED - STRUCTURE TEMPLATE CONTAINS AT LEAST ONE NAME NOT UNIQUE TO THE NAME SCOPE

PROGRAMMER'S GUIDE-SECTION 19

DQ9 - SEVERITY 2

THE DECLARED NODE ?? DUPLICATES A PREVIOUSLY DECLARED NODE NAME - CAUSING QUALIFIED REFERENCES TO THE TWO NODES TO BE INDISTINGUISHABLE

TWO NODES IN A SINGLE STRUCTURE HAVE THE SAME NAME. PROGRAMMER'S GUIDE-SECTION 19

DS1 - SEVERITY 2

INVALID BIT-LENGTH SPECIFICATION

THE EXPRESSION SPECIFYING BIT-LENGTH MUST BE AN ARITHMETIC EXPRESSION COMPUTABLE AT COMPILE TIME, CREATING A BIT STRING WITH A POSITIVE LENGTH NO GREATER THAN 32.

DS10 - SEVERITY 2

FACTORED AND NON-FACTORED STRUCTURE TEMPLATE REFERENCES DISAGREE: NON-FACTORED REFERENCE WILL BE USED

FOR EXAMPLE, "DECLARE Q-STRUCTURE,ZIP,ZAP,ZOOP T-STRUCTURE;" IS AMBIGUOUS AS TO ZOOP'S TREE ORGANIZATION. IN THIS CASE, TEMPLATE T WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED TEMPLATE REFERENCES.

DS11 - SEVERITY 2

INPUT/ASSIGN PARAMETERS OF CHARACTER TYPE CAN ONLY BE GIVEN A * LENGTH SPECIFICATION

SPEC-SECTION 4.7

DS2 - SEVERITY 2

INVALID CHAR-LENGTH SPECIFICATION

THE EXPRESSION SPECIFYING CHARACTER LENGTH MUST BE AN ARITHMETIC EXPRESSION COMPUTABLE AT COMPILE-TIME, CREATING A CHARACTER STRING WITH A LENGTH NO GREATER THAN 255.

DS3 - SEVERITY 2

A * IS AN ILLEGAL CHARACTER LENGTH SPECIFICATION; A LENGTH OF 8 IS ASSUMED

(*) MAY ONLY BE USED IN DECLARING CHARACTER STRINGS AS FORMAL PARAMETERS IN PROCEDURES AND FUNCTIONS, OR IF THE VARIABLE IS UNARRAYED, IN DEFINING A NAME VARIABLE. SPEC-SECTIONS 4.7, 11.4.1

DS4 - SEVERITY 2

A * IS AN ILLEGAL BIT LENGTH SPECIFICATION; A LENGTH OF 1 IS ASSUMED

(*) MAY NEVER BE USED IN SPECIFYING BIT LENGTH.

DS5 - SEVERITY 2

FACTORED AND NON-FACTORED BIT SIZE SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED SPECIFICATION WILL BE USED

FOR EXAMPLE, "DECLARE BIT(8), B, C, FLAGS BIT(4);" WOULD BE AMBIGUOUS AS TO THE LENGTH OF THE BIT-STRING FLAGS. IN THIS CASE, A LENGTH OF 4 WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED LENGTH SPECIFICATIONS.

DS6 - SEVERITY 2

FACTORED AND NON-FACTORED CHARACTER LENGTH SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED SPECIFICATION WILL BE USED.

FOR EXAMPLE, "DECLARE CHARACTER(5), WORD1,WORD2,WORD3,BLANK CHARACTER(1)" WOULD BE AMBIGUOUS AS TO THE LENGTH OF THE CHARACTER STRING, BLANK. IN THIS CASE, A LENGTH OF 1 WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED LENGTH SPECIFICATIONS.

DS7 - SEVERITY 2

FACTORED AND NON-FACTORED MATRIX DIMENSION SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED SPECIFICATION WILL BE USED.

FOR EXAMPLE, "DECLARE MATRIX(4,4), SQUARE1, SQUARE2, NOTSQUARE MATRIX(3,2);" WOULD BE AMBIGUOUS AS TO THE SHAPE OF THE MATRIX, NOTSQUARE. IN THIS CASE A 3X2 MATRIX WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED DIMENSION SPECIFICATIONS.

DS8 - SEVERITY 2

FACTORED AND NON-FACTORED VECTOR DIMENSION SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED SPECIFICATION WILL BE USED

FOR EXAMPLE, "DECLARE VECTOR (3), POSITION, VELOCITY, TEST-TIMES VECTOR (5)" WOULD BE AMBIGUOUS AS TO THE LENGTH OF THE VECTOR, TEST-TIMES. IN THIS CASE, A LENGTH OF 5 WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED DIMENSION SPECIFICATIONS.

DS9 - SEVERITY 2

THE FACTORED AND NON-FACTORED STRUCTURE DIMENSION SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED SPECIFICATION WILL BE USED

FOR EXAMPLE, "DECLARE Q-STRUCTURE(50), Z1,Z2,Z3, Q-STRUCTURE(100)" WOULD BE AMBIGUOUS AS TO THE NUMBER OF COPIES OF THE STRUCTURE, Z3. IN THIS CASE, 100 COPIES WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED COPY SPECIFICATIONS.

DT1 - SEVERITY 2

CONFLICTING TYPE SPECIFICATIONS FOR ??

ONLY ONE TYPE SPECIFICATION MAY BE GIVEN FOR EACH DATA ITEM. "DECLARE M INTEGER MATRIX," FOR INSTANCE, IS NOT LEGAL. IF A MATRIX OF INTEGERS MUST BE CREATED, A TWO DIMENSIONAL ARRAY SHOULD PROBABLY BE EMPLOYED INSTEAD: "DECLARE MA MATRIX" CREATES A MATRIX OF SCALARS.

DT3 - SEVERITY 2

LABEL TYPE CONFLICT FOR ??

IF AN ITEM IS DECLARED AS A PROCEDURE, FOR INSTANCE, IT SHOULD NOT LATER BE DEFINED AS A FUNCTION OR TASK. SIMILARLY, IF AN ITEM IS CALLED, IT MUST BE A PROCEDURE; IF IT IS SCHEDULED, IT MUST BE A PROGRAM OR TASK. THIS MAY ALSO BE THE RESULT OF AN ATTEMPTED 'GO TO BLOCK-LABEL'.

DT4 - SEVERITY 2

ILLEGAL CHARACTER; HEX REPRESENTATION IS ??

A CHARACTER NOT VALID IN THE HAL/S CHARACTER SET HAS BEEN USED. SEE SPEC-SECTION 2.2

DT5 - SEVERITY 2

A TYPE SPECIFICATION MAY NOT BE USED ON THE MINOR STRUCTURE ??

A MINOR STRUCTURE NAME IS NOT THE NAME OF ANY ORDINARY DATA ITEM AND HENCE SHOULD GET NO TYPE SPECIFICATION (SCALAR, VECTOR, ETC.). IN EFFECT, IT IS THE NAME OF THE STRUCTURE CONSISTING OF ALL THE ELEMENTS BELOW IT ON THE STRUCTURE 'TREE'.

DT6 - SEVERITY 2

A STRUCTURE TEMPLATE MAY NOT CONTAIN A REFERENCE TO ITSELF

THE RESULT OF SUCH RECURSIVE SELF-REFERENCE WOULD BE A TEMPLATE OF INFINITE SIZE IN HAL/S.

DT7 - SEVERITY 2

ILLEGAL TYPE FOR THE STRUCTURE TERMINAL ??

A STRUCTURE TERMINAL MAY NOT ITSELF BE A MAJOR STRUCTURE (I.E. ANOTHER STRUCTURE TEMPLATE) OR EVENT. IT MAY, HOWEVER, BE A STRUCTURE DATA TYPE, AND REFER TO ANOTHER TEMPLATE NAME. SPEC-SECTION 4.5

DT8 - SEVERITY 2

?? IS EVENT TYPE AND MAY NOT THEREFORE BE AN INPUT PARAMETER

SEE SPEC-SECTION 4.7

DU1 - SEVERITY 2

UNDECLARED IDENTIFIER ??

ALL VARIABLES IN HAL/S MUST BE DECLARED BEFORE THEY'RE REFERENCED. CHECK TO MAKE SURE THAT THE VARIABLE HAS BEEN DECLARED AND THAT THE VARIABLE NAME HAS NOT BEEN TYPED INCORRECTLY.

DU10 - SEVERITY 2

ILLEGAL PRECISION QUALIFIER ON VARIABLE ?? IN AN EQUATE STATEMENT.

SEE SPEC-SECTION 11.5.1

DU100 - SEVERITY 2

REFERENCE TO UNDEFINED PROCEDURE OR FUNCTION

DU11 - SEVERITY 2

TARGET OF THE EQUATE SYMBOL ?? MAY NOT BE A SUBBIT PSEUDO-FUNCTION

SEE SPEC-SECTION 11.5.1

DU12 - SEVERITY 2

THE SYMBOL ?? MAY NOT BE USED AS THE TARGET OF AN EQUATE STATEMENT SINCE IT HAS A CONSTANT VALUE WHICH IS TREATED AS A LITERAL

SEE SPEC-SECTION 11.5.1

DU13 - SEVERITY 2

THE VARIABLE ?? HAS THE AUTOMATIC ATTRIBUTE AND IS DEFINED WITHIN A REENTRANT PROCEDURE/FUNCTION. IT MAY NOT BE REFERENCED IN AN EQUATE STATEMENT.

SEE SPEC-SECTION 11.5.1

DU14 - SEVERITY 2

TARGET OF THE EQUATE SYMBOL ?? MAY NOT BE A NAME PSEUDO-FUNCTION

SEE SPEC-SECTION 11.5.1

DU2 - SEVERITY 2

UNDECLARED PARAMETER ??

FORMAL PARAMETERS OF PROCEDURES AND FUNCTIONS MUST BE DECLARED IN THEIR DECLARE GROUP. MAKE SURE THIS HAS NOT BEEN INADVERTENTLY OMITTED.

DU3 - SEVERITY 0

?? IS NOT A PARAMETER AND CANNOT BE DECLARED IN A TEMPLATE

ONLY FORMAL PARAMETERS (WHICH WILL BE PASSED INTO AND OUT OF A COMSUB) SHOULD BE DECLARED IN PROCEDURE OR FUNCTION TEMPLATES. ONLY A WARNING: THE COMPILER WILL IGNORE THIS DECLARATION, AND EXECUTION WILL NOT BE INHIBITED.

DU4 - SEVERITY 2

NO TYPE INFORMATION AVAILABLE FOR ??
A DEFAULT TYPE HAS BEEN ASSIGNED

A FORMAL PARAMETER HAS BEEN EMPLOYED AFTER WHICH NO DECLARE STATEMENT EXISTS. THE COMPILER WILL ASSUME THAT IT IS SCALAR FOR THE PURPOSE OF FURTHER ERROR-SCANNING (WHICH MAY RESULT IN FURTHER SPURIOUS ERRORS), BUT EXECUTION WILL BE INHIBITED. THIS ERROR MAY ALSO BE THE RESULT OF PREVIOUS ERROR RECOVERY ACTIONS.

DU5 - SEVERITY 2

REFERENCE TO UNDECLARED STRUCTURE TEMPLATE ??

FOR EXAMPLE, "DECLARE ZAP Q-STRUCTURE;" CAN ONLY OCCUR AFTER A TEMPLATE "STRUCTURE Q" HAS BEEN DEFINED.

DU6 - SEVERITY 2

MULTIPLY DEFINED EQUATE NAME: ??

SEE SPEC-SECTION 11.5.1

DU7 - SEVERITY 2

THE VARIABLE ?? REFERENCED IN AN EQUATE STATEMENT IS DEFINED OUTSIDE THE CURRENT BLOCK

SEE SPEC-SECTION 11.5.1

DU8 - SEVERITY 2

THE SYMBOL ?? HAS AN ILLEGAL TYPE FOR REFERENCE IN AN EQUATE STATEMENT

SEE SPEC-SECTION 11.5.1

DU9 - SEVERITY 2

THE SYMBOL ?? HAS ATTRIBUTES WHICH ARE ILLEGAL FOR THE OBJECT OF AN EQUATE STATEMENT

SEE SPEC-SECTION 11.5.1

D1 - SEVERITY 2

THE TYPE SPECIFICATION FOR THE PARAMETER ?? IS ILLEGAL

D10 - SEVERITY 2

TEMPORARY STATEMENTS MAY NOT APPEAR AT THE OPENING OF A DO CASE...END GROUP

D100 - SEVERITY 2

STAR SIZE ARRAY TEMPORARY NOT ALLOWED

TEMPORARY ARRAYS MAY NOT BE DECLARED WITH A (*) LENGTH SPECIFICATION. REMOTE ARRAY PARAMETERS ARE TREATED LIKE TEMPORARY ARRAYS WHEN USED IN A SUM, MAX, MIN OR PROD BUILT-IN FUNCTION UNLESS A FINITE SIZE ARRAY IS ALSO PART OF THE ARGUMENT.

D11 - SEVERITY 2

TYPE SPECIFICATION OF ?? RENDERS NONHAL ATTRIBUTE ILLEGAL
ATTRIBUTE IGNORED.

ONLY PROCEDURES AND FUNCTIONS MAY BE DEFINED TO BE NON-HAL.

D12 - SEVERITY 2

DECLARATION OF ?? SPECIFIES ATTRIBUTES INCOMPATIBLE WITH THE NAME ATTRIBUTE.

THE ACCESS OR NONHAL ATTRIBUTE MAY NOT APPLY TO THE NAME VARIABLES.

D13 - SEVERITY 2

DECLARATION OF ?? HAS CONFLICTING FACTORED AND UNFACTORED NONHAL ATTRIBUTES - UNFACTORED ATTRIBUTE WILL BE USED.

FOR EXAMPLE, "DECLARE NONHAL SINGLE, F1 FUNCTION, F2 FUNCTION NONHAL(1);" WOULD BE AMBIGUOUS AS TO THE NONHAL TYPE OF F2. IN THIS CASE, NONHAL(1) WOULD BE ASSUMED.

D15 - SEVERITY 2

ORDER OF DECLARATIONS WILL CAUSE LOCAL DATA DECLARATIONS TO APPEAR IN ANY PROCEDURE OR FUNCTION TEMPLATE GENERATED BY THIS COMPILATION

NOT A FATAL ERROR: BUT IT'S BETTER TO DECLARE FORMAL PARAMETERS PRIOR TO ANY LOCAL DATA DECLARATION. DU3 ERRORS WILL RESULT WHEN INCLUDING THIS TEMPLATE.

D2 - SEVERITY 2

THE PROCEDURE OR FUNCTION ?? MAY NOT BE DECLARED IN A COMPOOL

COMPOOLS CONTAIN ONLY DATA TO BE SHARED. FUNCTIONS AND PROCEDURES TO BE SHARED SHOULD BE DECLARED IN A COMSUB (EXTERNAL FUNCTION OR EXTERNAL PROCEDURE). SPEC-SECTION 3.5

D5 - SEVERITY 2

?? HAS BOTH FACTORED AND NON-FACTORED LOCK GROUP SPECIFICATION - NON-FACTORED SPECIFICATION WILL BE USED.

FOR EXAMPLE, "DECLARE INTEGER LOCK(1), I,J,K LOCK(2);" WOULD BE AMBIGUOUS AS TO K'S LOCK GROUP. IN THIS CASE, LOCK(2) WOULD BE ASSUMED.

D6 - SEVERITY 2

THE DECLARATION OF ?? HAS BOTH FACTORED AND NON-FACTORED ARRAY SPECIFICATIONS; THE NON-FACTORED SPECIFICATION WILL BE USED

FOR EXAMPLE: "DECLARE ARRAY(5) INTEGER, I,J,K ARRAY(3)" WOULD BE AMBIGUOUS AS TO THE LENGTH OF K. IN THIS CASE, A LENGTH OF 3 WOULD BE ASSUMED. USE CARE IN EMPLOYING FACTORED SPECIFICATIONS.

D7 - SEVERITY 2

TEMPORARY STATEMENT DOES NOT APPEAR BEFORE THE FIRST EXECUTABLE STATEMENT OF THE ENCLOSING DO...END GROUP

ALL TEMPORARY DECLARATIONS MUST PRECEDE THE FIRST EXECUTABLE STATEMENT OF THE DO GROUP IN WHICH THEY APPEAR. SPEC-SECTION 11.3.1

D8 - SEVERITY 2

DECLARATION OF ?? SPECIFIES TYPE OR ATTRIBUTES NOT LEGAL FOR TEMPORARIES

TEMPORARY VARIABLES MAY NOT BE INITIALIZED OR DECLARED AS EVENT TYPE. THEY MAY NOT POSSESS ANY MINOR ATTRIBUTES AT ALL (DENSE/ALIGNED, ACCESS, LOCK, RIGID, LATCHED, STATIC/AUTOMATIC, NON-HAL, ETC.) SPEC-SECTION 11.3.1

ERROR MESSAGES FOR MAJOR CLASSIFICATION E

CLASSIFICATION "E" ERRORS DEAL WITH EXPRESSIONS

EA1 - SEVERITY 2

ARRAYNESS OF ?? IS INCONSISTENT WITH CURRENT ARRAYNESS OF EXPRESSION

EA100 - SEVERITY 2

ARRAYNESS CONFLICT

THE DEGREE OF ARRAYNESS MUST BE COMPATIBLE WITH THAT SPECIFIED IN THE STATEMENT. EXAMPLE: VAR(5) IS DECLARED BUT OPERATIONS ARE PERFORMED ON VAR(4:10), ARRAY ELEMENTS WHICH ARE OUTSIDE THE DECLARED RANGE.

EA101 - SEVERITY 2

ARRAYNESS INCONSISTENT WITH STATEMENT

EA102 - SEVERITY 2

COPYNESS CONFLICT

SAME AS ABOVE, BUT PERTAINING TO COPYNESS.

EA103 - SEVERITY 2

NON-ARRAYED ARGUMENT TO ARRAY FUNCTION

MAX, MIN, PROD, SIZE, AND SUM ALL REQUIRE THEIR ARGUMENTS TO BE ARRAYED. SPEC-APPENDIX C

EB1 - SEVERITY 1

RESULT OF BIT CATENATION WILL BE LEFT TRUNCATED TO MAXIMUM BIT LENGTH

BIT EXPRESSIONS OF LENGTH GREATER THAN THE IMPLEMENTATION MAXIMUM (LESS THAN OR EQUAL TO 32) ARE NOT PERMITTED; CARE THEREFORE SHOULD BE TAKEN WHEN CATENATING BIT VARIABLES. THE RESULTING BIT EXPRESSION WILL BE LEFT TRUNCATED.

EC1 - SEVERITY 2

CROSS PRODUCT MUST BE BETWEEN THREE DIMENSIONAL VECTORS

SEE SPEC-SECTION 6.1.1

EC2 - SEVERITY 2

CROSS PRODUCT * USED WITHOUT A VECTOR AFTER IT

SEE SPEC-SECTION 6.1.1 A 3-ARRAY WILL NOT WORK HERE.

EC3 - SEVERITY 2

CROSS PRODUCT * USED WITHOUT A VECTOR BEFORE IT

SEE SPEC-SECTION 6.1.1 A 3-DIMENSIONAL ARRAY WILL NOT WORK HERE.

ED1 - SEVERITY 2

DOT PRODUCT S USED WITHOUT A VECTOR AFTER IT

SEE SPEC-SECTION 6.1.1 AN ARRAY WILL NOT WORK HERE.

ED2 - SEVERITY 2

DOT PRODUCT S USED WITHOUT A VECTOR BEFORE IT

SEE SPEC-SECTION 6.1.1 AN ARRAY WILL NOT WORK HERE.

EL1 - SEVERITY 2

ONLY ARITHMETIC CONVERSION FUNCTIONS MAY POSSESS ARGUMENTS WITH REPEAT FACTORS

IN ALL OTHER FUNCTIONS WITH MULTIPLE PARAMETERS, EACH PARAMETER MUST BE SEPARATELY SPECIFIED.

EL2 - SEVERITY 2

REPETITION FACTOR OF EXPRESSION MUST BE AN UNARRAYED INTEGER OR SCALAR EXPRESSION COMPUTABLE AT COMPILE TIME

SEE SPEC-SECTION 6.5.1. THUS, FOR EXAMPLE, IF N IS A VARIABLE (NOT PREVIOUSLY DECLARED AS CONSTANT), THE EXPRESSION N 6 IS ILLEGAL AS AN ARGUMENT IN A CONVERSION FUNCTION. THE REPETITION FACTOR MUST BE IN THE RANGE: 1 < FACTOR <= 32767

EM1 - SEVERITY 2

DIMENSIONS OF MATRIX OPERANDS IN EXPRESSION DISAGREE

IN OPERATIONS INVOLVING MATRICES, THE SIZES OF THE OPERANDS MUST BE COMPATIBLE WITH THE OPERATION INVOLVED, IN THE USUAL MATHEMATICAL SENSE. SPEC-SECTION 6.1.1

EM2 - SEVERITY 2

MATRIX ARITHMETIC TYPE CANNOT BE CONVERTED TO ? A CHARACTER STRING

SEE SPEC-SECTION 6.5.3

EM3 - SEVERITY 2

MATRIX-MATRIX MULTIPLICATION DIMENSION DISAGREEMENT

IF M AND N ARE MATRICES, MN IS ONLY LEGAL IF THE SECOND DIMENSION OF M AND THE FIRST DIMENSION OF N ARE EQUAL.

EM4 - SEVERITY 2

INVERSE OF NON-SQUARE MATRIX ATTEMPTED

ONLY SQUARE MATRICES CAN BE INVERTED, I.E., BOTH DIMENSIONS MUST BE THE SAME.

EN1 - SEVERITY 2

AN ARGUMENT OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO MAY NOT BE A NAME PSEUDO-FUNCTION.

NAME(NAME(K)) IS ALWAYS ILLEGAL.

EN10 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO MAY NOT BE A TEMPORARY

SEE SPEC-SECTION 11.4.3

EN11 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO MAY NOT BE A CONSTANT OR AN INPUT PARAMETER

SEE SPEC-SECTION 11.4.5

EN12 - SEVERITY 2

ACCESS RIGHTS HAVE BEEN DENIED TO ?? - ITS USE AS AN ARGUMENT OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO IS THEREFORE ILLEGAL

SEE SPEC-SECTION 11.4.5

EN13 - SEVERITY 2

?? IS NOT A MAJOR STRUCTURE BUT POSSESSES STRUCTURE COPIES - IT IS THEREFORE ILLEGAL AS AN ARGUMENT OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO

EN14 - SEVERITY 2

AN ARGUMENT OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO MAY NOT POSSESS A PRECISION QUALIFIER.

EN2 - SEVERITY 2

THE ARGUMENT OF A NAME PSEUDO-FUNCTION MAY NOT BE A SUBBIT PSEUDO-VARIABLE.

EN4 - SEVERITY 2

THE TYPE OF THE LABEL ?? IS ILLEGAL AS THE ARGUMENT OF A NAME PSEUDO-FUNCTION.

THE ONLY ALLOWABLE LABEL TYPES ARE: PROGRAM AND TASK. ALL OTHER LABELS ARE INVALID.

EN5 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO MAY NOT BE A NONHAL PROCEDURE OR FUNCTION

EN6 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO IS A PROCEDURE OR FUNCTION WHICH IS NOT EXTERNAL

YOU ARE NOT ALLOWED TO USE PROCEDURE NAMES FOR VARIABLES, OR CREATE NAMES OF PROCEDURE BLOCKS.

EN7 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO POSSESSES ILLEGAL SUBSCRIPTING

BIT AND CHARACTER TYPES MAY NOT POSSESS SUBSCRIPTING IN NAME PSEUDO-FUNCTIONS; AND OTHER TYPES MUST BE SUBSCRIPTED SO THAT A SINGLE ELEMENT IS CHOSEN. SPEC-SECTION 11.4.5

EN8 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO MAY NOT POSSESS THE DENSE ATTRIBUTE

SEE SPEC-SECTION 11.4.5

EN9 - SEVERITY 2

ARGUMENT ?? OF THE NAME PSEUDO-FUNCTION OR SPECIFIED %MACRO MAY NOT BE A MINOR NODE OF A STRUCTURE

SEE SPEC-SECTION 11.4.5

EO1 - SEVERITY 2

ILLEGAL PRODUCT: OUTER PRODUCT TIMES A VECTOR

EO2 - SEVERITY 2

A PRODUCT INVOLVING BOTH CROSS AND OUTER PRODUCTS IS INDICATED. USE MORE PARENTHESES.

"V1 V2*V3" IS ILLEGAL, SINCE IT'S INTERPRETED AS "(V1 V2)*V3". WRITE "V1(V2*V3)" INSTEAD. SEE SPEC-SECTION 6.1.1

EO3 - SEVERITY 2

A PRODUCT INVOLVING BOTH DOT AND OUTER PRODUCTS IS INDICATED. USE MORE PARENTHESES.

"V1 V2SV3" IS ILLEGAL, SINCE IT'S INTERPRETED AS (V1 V2)SV3). USE "V1(V2SV3)" INSTEAD. SEE SPEC-SECTION 6.1.1

EV1 - SEVERITY 2

LENGTHS OF VECTOR OPERANDS IN EXPRESSION DISAGREE

ALL VECTORS EMPLOYED IN AN EXPRESSION MUST HAVE THE SAME LENGTH. SPEC-SECTION 6.1.1

EV2 - SEVERITY 2

MATRIX-VECTOR MULTIPLICATION DIMENSION DISAGREE

IN "M V", THE LENGTH OF V MUST EQUAL THE SECOND DIMENSION OF M.

EV3 - SEVERITY 2

VECTOR-MATRIX MULTIPLICATION DIMENSION DISAGREEMENT

IN "V M", THE LENGTH OF V MUST EQUAL THE SECOND DIMENSION OF M.

EV4 - SEVERITY 2

VECTOR MAY NOT HAVE AN EXPONENT

SEE SPEC-SECTION 6.1.1

EV5 - SEVERITY 2

VECTOR ARITHMETIC TYPE CANNOT BE CONVERTED TO A CHARACTER STRING

SEE SPEC-SECTION 6.1.3

E1 - SEVERITY 2

DIVISORS MAY ONLY BE OF INTEGER OR SCALAR TYPE

DIVISION BY A MATRIX OR VECTOR IS NOT PERMITTED. SPEC-SECTION 6.1.1

E100 - SEVERITY 2

EVENT EXPRESSION TOO LONG

THE FOLLOWING LIMITATIONS PERTAIN TO EVENT EXPRESSIONS: IN THE 360 ENVIRONMENT, NO EXPRESSION CAN HAVE MORE THAN 50 OPERATORS AND 10 OPERANDS, AND ANY COMBINATION THEREOF CANNOT EXCEED 50. IN THE AP101 ENVIRONMENT, NO EXPRESSION MAY CONTAIN MORE THAN 5 OPERANDS, WITH ANY COMBINATION THEREOF NOT EXCEEDING 15.

E101 - SEVERITY 2

TOO MANY UNIQUE OPERANDS IN EVENT EXPRESSION

E102 - SEVERITY 2

INVALID EVENT EXPRESSION

VALID EVENT EXPRESSIONS ARE LIMITED TO THE FOLLOWING TYPES: A SINGLE EVENT VARIABLE WITH OR WITHOUT THE 'NOT' OPERATOR, MORE THAN ONE SINGLE EVENT VARIABLE CONNECTED BY ALL 'OR' OPERATORS, AND MORE THAN ONE SINGLE EVENT VARIABLE CONNECTED BY ALL 'AND' OPERATORS.

E2 - SEVERITY 2

MATRIX MUST HAVE AN EXPONENT OF INTEGER TYPE KNOWN AT COMPILE TIME

SEE SPEC-SECTION 6.1.1 EXPONENTIATION OF A MATRIX BY A VARIABLE (NOT DECLARED AS A CONSTANT) IS THEREFORE ILLEGAL.

E3 - SEVERITY 2

EXPONENT MUST BE A SINGLE VALUED QUANTITY

EXPONENTIATION BY A VECTOR OR MATRIX IS NOT PERMITTED. SEE SPEC-SECTION 6.1.1

E4 - SEVERITY 2

DOT OR CROSS PRODUCT SYMBOL (S OR *) USED IN A PRODUCT NOT INVOLVING VECTORS

CHECK THAT THE OPERATIONS ARE NOT BEING PERFORMED IN THE WRONG ORDER BECAUSE OF PRECEDENCE RULES OR IMPROPER PLACEMENT. RECALL ALSO THAT * IS THE SYMBOL FOR VECTOR CROSS-PRODUCT; MULTIPLICATION IS REPRESENTED BY ADJACENCY. SEE SPEC-SECTION 6.1.1

E6 - SEVERITY 2

INCOMPATIBLE ARITHMETIC OPERAND TYPES IN EXPRESSION

SEE SPEC-SECTION 6.1.1 FOR A SUMMARY OF ALL LEGAL OPERAND TYPES.

ERROR MESSAGES FOR MAJOR CLASSIFICATION F

CLASSIFICATION "F" ERRORS DEAL WITH FORMAL PARAMETERS AND ARGUMENTS

FD100 - SEVERITY 2

ARRAYNESS CONFLICT ON PARAMETER #??

THE NUMBER OF DIMENSIONS OF ARRAYNESS OF THE ARGUMENT AND ITS CORRESPONDING FORMAL PARAMETER DO NOT AGREE. SEE SPEC SECTIONS 6.4, 7.4.

FD101 - SEVERITY 2

SIZE CONFLICT ON VECTOR/MATRIX PARAMETER #??

ALL VECTOR/MATRIX ARGUMENTS MUST AGREE IN SIZE WITH THE FORMAL PARAMETERS THEY REPLACE. SEE SPEC-SECTIONS 6.4, 7.4

FD102 - SEVERITY 2

ARRAY SIZE CONFLICT ON PARAMETER #??

THE SIZE OF AN ARRAYNESS DIMENSION OF THE ARGUMENT AND ITS CORRESPONDING FORMAL PARAMETER DO NOT AGREE. SEE SPEC SECTIONS 6.4, 7.4.

FD103 - SEVERITY 2

STRUCTURE COPYNESS CONFLICT ON PARAMETER #??

SAME AS FD101, BUT PERTAINING TO COPYNESS

FD104 - SEVERITY 2

STRUCTURE COPY SIZE CONFLICT ON PARAMETER #??

SAME AS FD102, BUT PERTAINING TO COPYNESS

FD4 - SEVERITY 2

ARRAYNESS OF FUNCTION ARGUMENT DOES NOT MATCH CURRENT ARRAYNESS OF EXPRESSION CONTAINING THE INVOCATION

IF AN ARRAY IS PASSED AS AN ARGUMENT TO A FUNCTION WHICH HAS A SINGLE VALUED PARAMETER, THE FUNCTION WILL BE REPETITIVELY INVOKED FOR EACH ARRAY ELEMENT TO CREATE A SIMILARLY ARRAYED RETURN VALUE. SUCH AN ARRAY MUST BE LEGAL WITHIN THE EXPRESSION IN WHICH THE FUNCTION CALL IS FOUND. SEE SPEC-SECTION 6.4

FD6 - SEVERITY 2

ARGUMENT OF ?? FUNCTION IS NOT A SQUARE MATRIX

DET, INVERSE, AND TRACE MAY ONLY BE USED ON SQUARE MATRICES. SPEC-APPENDIX C

FD7 - SEVERITY 2

A NAME PSEUDO-FUNCTION MAY NOT POSSESS MULTIPLE COPIES IF AN ARGUMENT OF A PROCEDURE CALL.

FN100 - SEVERITY 2

WRONG NUMBER OF PERCENT MACRO ARGUMENTS

SVC EXPECTS 1 ARGUMENT, NAMECOPY 2 ARGUMENTS, COPY 2 OR 3, NAMEBIAS 2 ARGUMENTS, AND NAMEADD 3 ARGUMENTS. SEE USER'S GUIDE-SECTION 8.7

FN102 - SEVERITY 2

INCORRECT NUMBER OF ARGUMENTS TO ??

EXAMPLE:

```
CALL  ABC(X,Y,Z);
  ABC:PROCEDURE(X,Y);
```

INVOKING ABC SPECIFIES 3 PARAMETERS, BUT THE PROCEDURE DECLARATION CONTAINS 2 ARGUMENTS.

FN103 - SEVERITY 2

UNEXPECTED PERCENT MACRO ARGUMENT

FN105 - SEVERITY 1

?? OF %COPY INVOLVES A NAME VARIABLE DEREFERENCE. NOT CHECKED.

FN106 - SEVERITY 2

ELEMENT BOUNDARY EXCEEDED FOR ?? OF %COPY

FN107 - SEVERITY 1

?? OF %COPY INVOLVES RUNTIME ADDRESSING. NOT CHECKED.

FN108 - SEVERITY 1

?? OF %COPY INVOLVES A PASS-BY-REFERENCE PARAMETER DEREFERENCE. NOT CHECKED.

FN3 - SEVERITY 3

?? USED MORE THAN ONCE AS A PARAMETER

EACH FORMAL PARAMETER MUST BE UNIQUELY NAMED.

FN4 - SEVERITY 2

?? FUNCTION HAS INCORRECT NUMBER OF ARGUMENTS

THIS IS THE RESULT OF CALLING A HAL/S BUILT-IN FUNCTION WITH AN INCORRECT NUMBER OF ARGUMENTS.

FS1 - SEVERITY 2

AN ASSIGN ARGUMENT OF A PROCEDURE CALL MAY NOT BE A SUBBIT PSEUDO VARIABLE

SEE SPEC-SECTION 6.5.4

FS2 - SEVERITY 2

THE STRUCTURE COPIES OF ASSIGN ARGUMENT ?? MUST BE SUBSCRIPTED AWAY

SEE SPEC-SECTION 7.4

FT10 - SEVERITY 2

A NAME PSEUDO-FUNCTION MAY NOT BE THE ARGUMENT OF A BUILT-IN OR SHAPING/CONVERSION FUNCTION.

FT100 - SEVERITY 2

ILLEGAL ARGUMENT TO SIZE FUNCTION

THE ARGUMENT TO THE BUILT-IN SIZE FUNCTION MUST BE AN UNSUBSCRIPTED DATA ITEM WITH A ONE DIMENSIONAL ARRAY SPECIFICATION OR A MULTIPLE COPY SPECIFICATION, OR ELSE AN UNSUBSCRIPTED BUT (ONE DIMENSIONALLY) ARRAYED STRUCTURE TERMINAL. SPEC-APPENDIX C

FT101 - SEVERITY 2

DATA TYPE CONFLICT ON PARAMETER #??

DATA TYPE IN PARAMETER AND ARGUMENT MUST BE COMPATIBLE. SEE SPEC-SECTIONS 6.4, 7.4, 11.4.9

FT102 - SEVERITY 2

ASSIGN PARAMETER ATTRIBUTES MUST MATCH

THE ATTRIBUTES OF A PROCEDURE'S ASSIGN PARAMETERS MUST MATCH PRECISELY THE ATTRIBUTES OF THE ARGUMENTS EMPLOYED WHEN THE PROCEDURE IS CALLED: THIS IS TRUE OF TYPE, PRECISION, DIMENSION, ARRAYNESS, AND STRUCTURE TREE ORGANIZATION. THE REMOTE ATTRIBUTE MUST ALSO MATCH IF THE ASSIGN PARAMETER IS NON-REMOTE OR IS A NAME VARIABLE. CHARACTER LENGTHS NEED NOT MATCH PRECISELY, HOWEVER. SPEC-SECTION 7.4

FT103 - SEVERITY 2

LOCK NUMBER CONFLICT ON PARAMETER #??

IF AN ASSIGN ARGUMENT IN A PROCEDURE CALL IS OF LOCK GROUP N, THEN THE CORRESPONDING ASSIGN PARAMETER MUST ALSO BE OF LOCK GROUP N, OR ELSE OF (*). IF THE ASSIGN ARGUMENT IS OF GROUP (*), THE ASSIGN PARAMETER MUST BE TOO. SPEC-SECTION 7.4

FT104 - SEVERITY 2

ILLEGAL NAME PARAMETER #??

THE ARGUMENT OF A NAME PSEUDO-FUNCTION WHICH IS A PARAMETER MUST BE EITHER NULL OR A VARIABLE. EXPRESSIONS ARE NOT ALLOWED.

FT105 - SEVERITY 2

NOT ASSIGN PARAMETER

ALL ARGUMENTS IN A PROCEDURE CALL WHICH ARE USED AS ASSIGN ARGUMENTS MUST CORRESPOND TO A DEFINED ASSIGN PARAMETER IN THE PROCEDURE DEFINITION. FOR EXAMPLE:

```
PROC1:PROCEDURE ASSIGN(A,B,C);
CALL PROC1(A,B,C);
  OR
PROC1:PROCEDURE(A,B) ASSIGN(C);
CALL PROC1(A) ASSIGN(B,C);
```

FT106 - SEVERITY 2

NOT NAME PARAMETER

IF AN ARGUMENT OF A PROCEDURE OR FUNCTION IS A NAME VALUE, THE CORRESPONDING FORMAL PARAMETER OF THE PROCEDURE OR FUNCTION MUST BE DECLARED AS HAVING THE NAME ATTRIBUTE AS WELL.

FT107 - SEVERITY 2

INVALID ARGUMENT TO ARRAY FUNCTION

MAX, MIN, PROD, AND SUM ALL REQUIRE THEIR ARGUMENTS TO BE ARRAYS OF INTEGERS OR SCALARS. SEE SPEC-APPENDIX C

FT108 - SEVERITY 2

ATTEMPT TO PASS REMOTE PARAMETER BY REFERENCE TO RTL ROUTINE THAT DOES NOT SUPPORT REMOTE ADDRESSING

FT109 - SEVERITY 2

ASSIGN PARAMETER MUST NOT HAVE DENSE ATTRIBUTE

THIS MESSAGE APPLIES TO BIT STRUCTURE NODES.

FT110 - SEVERITY 2

MISMATCHED ARGUMENTS IN %NAMEADD STATEMENT. REMOTE SOURCE NOT ALLOWED WITH NON-REMOTE DESTINATION IN %NAMEADD STATEMENT.

FT111 - SEVERITY 2

MISMATCHED ARGUMENTS IN %NAMECOPY STATEMENT. REMOTE SOURCE NOT ALLOWED WITH NON-REMOTE DESTINATION IN %NAMECOPY STATEMENT.

FT112 - SEVERITY 2

PARAMETER #?? MAY NOT BE NAME (NAMEVAR) IF NAMEVAR LIVES REMOTE

THE REMOTE KEYWORD ON A NAME VARIABLE SPECIFIES WHERE IT POINTS, NOT WHERE IT LIVES. THERE IS NO CURRENT WAY TO DECLARE A REMOTE NAME INSIDE A PROCEDURE/FUNCTION.

FT113 - SEVERITY 2

REMOTE INPUT ARGUMENT CANNOT BE PASSED INTO NON-REMOTE NAME PARAMETER.

FT2 - SEVERITY 2

?? FUNCTION HAS AN ARGUMENT OF INCORRECT TYPE

SEE SPEC, APPENDIX C, FOR COMPLETE INFORMATION ON THE TYPE OF ARGUMENTS USED IN BUILT-IN FUNCTIONS.

FT3 - SEVERITY 2

ILLEGAL TYPE FOR THE FUNCTION ??

FUNCTIONS MAY NOT BE EVENT TYPE.

FT4 - SEVERITY 2

THE SIZE SPECIFICATION FOR THE RETURN TYPE OF FUNCTION ?? DISAGREES WITH THE PREVIOUSLY DECLARED SIZE

THE SPECIFICATION OF THE DIMENSIONALITY OF THE VALUE RETURNED BY A FUNCTION HEADER MUST MATCH THAT OF ANY EARLIER DECLARE STATEMENT IN WHICH THE FUNCTION WAS FIRST MENTIONED.

FT6 - SEVERITY 2

THE STRUCTURE TEMPLATE INDICATED IN THE TYPE SPECIFICATION OF FUNCTION ?? DISAGREES WITH THE TEMPLATE USED IN A PREVIOUS DECLARATION

THE SPECIFICATION OF THE ORGANIZATION OF THE STRUCTURE RETURNED BY A FUNCTION IN THE ACTUAL FUNCTION HEADER MUST MATCH THAT OF ANY EARLIER DECLARE STATEMENT IN WHICH THE FUNCTION WAS FIRST MENTIONED

FT7 - SEVERITY 2

CONFLICTING SINGLE/DOUBLE SPECIFICATION FOR THE FUNCTION ??

THE SPECIFICATION OF THE PRECISION OF THE VALUE RETURNED BY A FUNCTION IN THE ACTUAL FUNCTION HEADER MUST MATCH THAT OF ANY EARLIER DECLARE STATEMENT IN WHICH THE FUNCTION WAS FIRST MENTIONED.

FT8 - SEVERITY 2

THE FUNCTION ?? MAY POSSESS NEITHER A SUBSCRIPT NOR A PRECISION QUALIFIER.

F100 - SEVERITY 2

ASSIGN PARAMETER NOT SYMBOL

ASSIGN PARAMETER CAN ONLY BE A VARIABLE, EXPRESSIONS ARE NOT ALLOWED.

F101 - SEVERITY 2

PERCENT MACRO CALL CONFLICT

F102 - SEVERITY 2

NESTED PERCENT MACRO

%MACROS MAY NOT BE NESTED WITHIN EACH OTHER

F103 – SEVERITY 2

THE LITERAL VALUE FOR THE SHIFT COUNT IS OUT OF RANGE.

THE VALID RANGE IS 1 TO 63.

ERROR MESSAGES FOR MAJOR CLASSIFICATION G

CLASSIFICATION "G" ERRORS DEAL WITH STATEMENT GROUPINGS
(DO STATEMENTS)

GB1 - SEVERITY 2

BIT EXPRESSION IN ?? CLAUSE MUST BE BOOLEAN

CONDITIONAL BIT EXPRESSION MAY ONLY INVOLVE BIT STRINGS OF LENGTH 1. THE TERM, "BOOLEAN" IS EQUIVALENT TO "BIT(1)".

GC1 - SEVERITY 2

CONTROL EXPRESSION IN A DO CASE MUST BE OF UNARRAYED INTEGER OR SCALAR TYPE

SEE SPEC-SECTION 7.6.2

GC2 - SEVERITY 2

BIT EXPRESSION IN WHILE OR UNTIL CLAUSE MAY NOT BE ARRAYED

GC3 - SEVERITY 2

CONTROL EXPRESSIONS IN A DO FOR MUST BE OF UNARRAYED INTEGER OR SCALAR TYPE

SEE SPEC-SECTION 7.6.4

GE1 - SEVERITY 2

EXIT IS EITHER NOT IN A DO...END GROUP, OR NO LABEL MATCH WAS FOUND.

EXIT STATEMENTS ARE ONLY LEGAL WITHIN DO...END GROUPS. EXIT "LABEL" STATEMENTS CAUSE EXIT FROM THE ENCLOSING DO GROUP WITH THE SAME LABEL; HENCE THE LABEL GIVEN IN THE EXIT STATEMENT MUST MATCH SOME SUCH GROUP. CHECK THAT MISTYPING HAS NOT CAUSED A REFERENCE TO A NON-EXISTENT GROUP. SEE SPEC-SECTION 7.7

GE2 - SEVERITY 2

REPEAT IS EITHER NOT IN A DO FOR...END OR DO WHILE/UNTIL...END GROUP, OR NO LABEL MATCH WAS FOUND.

REPEAT STATEMENTS ARE ONLY LEGAL WITHIN DO FOR, DO WHILE, OR DO UNTIL GROUPS (AND NOT LEGAL WITHIN DO CASE OR PLAIN DO...END GROUPS WHICH DO NOT CAUSE REPEATED EXECUTION). REPEAT "LABEL" STATEMENTS CAUSE A JUMP TO THE NEXT EXECUTION OF THE ENCLOSING DO GROUP WITH THE SAME LABEL; HERE THE LABEL SPECIFIED IN THE REPEAT STATEMENT MUST MATCH SOME SUCH GROUP. CHECK TO MAKE SURE THAT MISTYPING HAS NOT CAUSED A REFERENCE TO A NON-EXISTENT GROUP. SPEC-SECTION 7.7

GE3 - SEVERITY 2

EXIT CAUSES ILLEGAL BRANCHING OUT OF CODE BLOCK DEFINITION

EXIT STATEMENTS MAY NOT CAUSE EXECUTION TO BRANCH OUT OF A CODE BLOCK (PROCEDURE, FUNCTION, ETC.). THUS A PROCEDURE NESTED WITHIN A DO...END GROUP MAY NOT INCLUDE AN EXIT STATEMENT CAUSING CONTROL OF EXECUTION TO PASS OUT OF THE PROCEDURE TO THE NEXT EXECUTABLE STATEMENT FOLLOWING THE DO...END GROUP. SEE SPEC-SECTION 7.7

GE4 - SEVERITY 2

REPEAT CAUSES ILLEGAL BRANCHING OUT OF CODE BLOCK DEFINITION

REPEAT STATEMENTS MAY NOT CAUSE EXECUTION TO BRANCH OUT OF A CODE BLOCK (PROCEDURE, FUNCTION, ETC.). THUS, A PROCEDURE NESTED WITHIN, FOR EXAMPLE, A DO FOR...END GROUP, MAY NOT INCLUDE A REPEAT STATEMENT CAUSING CONTROL TO PASS OUT OF THE BLOCK AND BACK TO THE BEGINNING OF THE DO FOR...END GROUP. SEE SPEC-SECTION 7.7

GL1 - SEVERITY 2

LABEL AFTER END STATEMENT DOES NOT MATCH DO STATEMENT LABEL

SEE SPEC-SECTION 7.6.6

GL2 - SEVERITY 2

LABEL IS THE DESTINATION OF A GO TO FROM OUTSIDE THE ENCLOSING DO...END GROUP

SEE SPEC-SECTION 7.7

GL3 - SEVERITY 2

GO TO STATEMENT CAUSES A BRANCH INTO A DO...END GROUP

SEE SPEC-SECTION 7.7

GL4 - SEVERITY 2

LABELED TRUE PART OF IF STATEMENT CANNOT BE DESTINATION OF GO TO STATEMENT

GO TO STATEMENTS MAY NOT CAUSE EXECUTION TO BRANCH TO THE TRUE PART OF AN IF STATEMENT. (I.E., "IF K=0 THEN ZEROFIX:DO..." IS A STATEMENT IN THE PROGRAM, GO TO ZEROFIX IS NOT PERMITTED, ALTHOUGH WITHIN THE DO GROUP LABELED ZEROFIX, STATEMENTS SUCH AS EXIT ZEROFIX AND REPEAT ZEROFIX ARE LEGAL). SPEC-SECTION 7.2

GL5 - SEVERITY 2

LABELED FALSE PART OF IF STATEMENT CANNOT BE DESTINATION OF GO TO STATEMENT

GO TO STATEMENTS MAY NOT CAUSE EXECUTION TO BRANCH TO THE TRUE PART OF AN IF STATEMENT.

GL6 - SEVERITY 2

LABELED ELSE PART OF DO CASE STATEMENT CANNOT BE DESTINATION OF GO TO STATEMENT

GO TO STATEMENTS MAY NOT CAUSE EXECUTION TO BRANCH TO THE TRUE PART OF AN IF STATEMENT. FOR EXAMPLE:

```
IF K=0 THEN DO...
  END;
ZEROFIX:ELSE DO...
```

GOING TO ZEROFIX IS NOT PERMITTED, ALTHOUGH WITHIN THE DO GROUP LABELED ZEROFIX, STATEMENTS SUCH AS "EXIT ZEROFIX" AND "REPEAT ZEROFIX" ARE LEGAL. SEE SPEC-SECTION 7.6.2

GL7 - SEVERITY 2

LABELED TARGET STATEMENT OF ON ERROR STATEMENT CANNOT BE DESTINATION OF GO TO STATEMENT

GO TO STATEMENTS MAY NOT CAUSE EXECUTION TO BRANCH TO THE TARGET STATEMENT OF AN ON ERROR STATEMENT. I.E. IF "ON ERROR 72 FIXUP:DO..." IS FOUND IN THE PROGRAM, "GO TO FIXUP" IS NOT PERMITTED, ALTHOUGH, WITHIN THE DO GROUP LABELED FIXUP, STATEMENTS SUCH AS "EXIT FIXUP" AND "REPEAT FIXUP" ARE LEGAL. INSTEAD OF "GO TO FIXUP", "SEND ERROR" COULD BE USED. SEE SPEC-SECTION 7.1

GV1 - SEVERITY 2

CONTROL VARIABLE IN A DO FOR MUST BE OF UNARRAYED INTEGER OR SCALAR TYPE

SEE SPEC-SECTION 7.6.4

ERROR MESSAGES FOR MAJOR CLASSIFICATION I

CLASSIFICATION "I" ERRORS ARE RELATED TO IDENTIFIERS

IL1 - SEVERITY 2

IDENTIFIER NAME MAY NOT END WITH AN UNDERSCORE CHARACTER

SEE SPEC-SECTION 2.3.2

IL2 - SEVERITY 2

NAME TOO LONG - TRUNCATED

SEE SPEC-SECTION 2.3.2

IR1 - SEVERITY 2

ILLEGAL REPLACEMENT FOR LOCAL NAME: ??

A LOCALLY DECLARED VARIABLE IS LATER SPECIFIED TO BE A REPLACE MACRO NAME.

IR10 - SEVERITY 4

MAXIMUM NUMBER OF PARAMETERS FOR SOURCE MACRO DEFINITION EXCEEDED

NO MORE THAN 12 PARAMETERS ARE PERMITTED IN THE DEFINITION OF ANY REPLACE MACRO. PROGRAMMER'S GUIDE-SECTION 29

IR4 - SEVERITY 3

ILLEGAL USE OR UNDEFINED REFERENCE OF CENT MACRO ??

THE IDENTIFIER BETWEEN THE CENT SIGNS IS NOT A REPLACE MACRO.

IR5 - SEVERITY 2

DUPLICATE REPLACE FOR ??

TWO DIFFERENT REPLACE MACROS CANNOT CONTAIN THE SAME TEXT.

IR6 - SEVERITY 3

STACK LIMIT OF PARAMETERS EXCEEDED; RECURSIVE EXPANSION OF PARAMETER LIKELY

A REPLACE MACRO NAME CANNOT BE THE SAME AS AN ARGUMENT WITHIN THE REPLACE TEXT.

IR7 - SEVERITY 3

REPLACE PARAMETER STRING TOO LONG;?REPLACE NOT PERFORMED

THE MAXIMUM SIZE OF ANY ARGUMENT USED IN THE EXPANSION OF A REPLACE MACRO IS LIMITED TO 250 CHARACTERS. PROGRAMMER'S GUIDE-SECTION 29

IR8 - SEVERITY 3

INCORRECT NUMBER OF PARAMETERS FOR MACRO CALL; REPLACEMENT NOT PERFORMED

IR9 - SEVERITY 3

MACRO EXPANSION STACK OVERFLOW; RECURSIVE DEFINITION LIKELY

REPLACE MACRO EXPANSIONS MAY NOT INVOKE OTHER MACROS BEYOND EIGHT LEVELS.

IS1 - SEVERITY 2

ILLEGAL CONSTRUCTION OF QUALIFIED STRUCTURE NAME

THE DECLARATION OF OR REFERENCE TO A QUALIFIED STRUCTURE MUST COME AFTER THE STRUCTURE TEMPLATE HAVING THAT NAME, I.E., A STATEMENT SUCH AS: "DECLARE Z Q-STRUCTURE" MUST BE PRECEDED BY THE TEMPLATE DESCRIBING "Q:STRUCTURE". SPEC-SECTION 4.6

IS2 - SEVERITY 2

ILLEGAL CONSTRUCTION OF UNQUALIFIED STRUCTURE NAME, INVOLVING ??

ERROR MESSAGES FOR MAJOR CLASSIFICATION L

CLASSIFICATION "L" ERRORS DEAL WITH LITERALS

LB1 - SEVERITY 2

BIT CONSTANTS MAY NOT BE LONGER THAN 32 BITS

LB2 - SEVERITY 2

DECIMAL BIT CONSTANT MUST SPECIFY OR IMPLY A REPETITION FACTOR OF 1

LB3 - SEVERITY 2

ILLEGAL DECIMAL STRING IN DECIMAL BIT CONSTANT

BIT CONSTANTS MAY ONLY CONTAIN 0'S OR 1'S. DIGITS 2-9 ARE INVALID IN STRING.

LB4 - SEVERITY 2

ILLEGAL CHARACTER IN DECIMAL BIT CONSTANT

THE ONLY CHARACTERS PERMITTED IN A DECIMAL BIT CONSTANT ARE 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

LB5 - SEVERITY 2

ILLEGAL CHARACTER IN BINARY BIT CONSTANT

THE ONLY CHARACTERS PERMITTED IN A BINARY BIT CONSTANT ARE 0 AND 1, OFF, ON. TRUE, FALSE, ETC. ARE NOT PERMITTED EXCEPT IN BINARY BIT OF LENGTH 1 (I.E. BOOLEANS).

LB6 - SEVERITY 2

ILLEGAL CHARACTER IN OCTAL BIT CONSTANT

THE ONLY CHARACTERS PERMITTED IN AN OCTAL BIT CONSTANT ARE 0, 1, 2, 3, 4, 5, 6, 7.

LB7 - SEVERITY 2

ILLEGAL CHARACTER IN HEXADECIMAL BIT CONSTANT

THE ONLY CHARACTERS PERMITTED IN A HEXADECIMAL BIT CONSTANT ARE 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

LB8 - SEVERITY 2

REPETITION FACTOR OF A BIT LITERAL MUST BE GREATER THAN ZERO

SEE SPEC-SECTION 2.3.4

LC2 - SEVERITY 2

?? NOT EXPRESSIBLE INTERNALLY

NUMBERS OUTSIDE THE RANGE OF 10**-78 TO 10**75 ARE NOT REPRESENTABLE INTERNALLY.

LF1 - SEVERITY 2

ILLEGAL NUMERIC LITERAL CONSTRUCTION

SEE SPEC 2-8 FOR RULES CONCERNING CONSTRUCTION OF ARITHMETIC LITERALS.

LF2 - SEVERITY 2

ONLY ONE DECIMAL POINT ALLOWED

LF3 - SEVERITY 2

TOO MANY SIGNIFICANT DIGITS - 74 ALLOWED

LF5 - SEVERITY 2

EXPONENT INDICATOR BUT NO EXPONENT DIGITS

IF AN EXPONENT INDICATOR IS SPECIFIED (E, B, OR H), THEN SOME POWER MUST BE SPECIFIED AS WELL. SEE SPEC-SECTION 2.3.4

LS1 - SEVERITY 2

CHARACTER STRING TOO LONG - TRUNCATED TO 255 CHARACTERS

CHARACTER STRINGS OF LENGTH GREATER THAN 255 ARE NOT PERMITTED. THE RESULTING CHARACTER EXPRESSION WILL BE RIGHT TRUNCATED.

LS2 - SEVERITY 2

REPETITION FACTOR OF A CHARACTER LITERAL IS NOT GREATER THAN ZERO

SEE SPEC-SECTION 2.3.4

ERROR MESSAGES FOR MAJOR CLASSIFICATION M

CLASSIFICATION "M" ERRORS DEAL WITH MULTI-LINE FORMATS

MC1 - SEVERITY 2

ILLEGAL CONTEXT FOR OVERPUNCH

ONLY VARIABLES MAY BE OVERPUNCHED.

MC2 - SEVERITY 2

OVERPUNCH ILLEGAL ON FUNCTION NAMES

FUNCTION NAMES MAY NOT BE OVERPUNCHED.

MC3 - SEVERITY 2

OVERPUNCH ILLEGAL ON REPLACED NAME

REPLACED NAMES MAY NOT BE OVERPUNCHED.

MC4 - SEVERITY 2

OVERPUNCH NOT VALID ON RESERVED WORD ??

HAL/S KEYWORDS MAY NOT BE OVERPUNCHED.

MC5 - SEVERITY 2

OVERPUNCH ILLEGAL ON PARAMETER ??

FORMAL PARAMETERS MAY NOT BE OVERPUNCHED.

MC6 - SEVERITY 2

OVERPUNCH ILLEGAL IN DECLARATION OF ??

OVERPUNCHES MAY NOT APPEAR IN DECLARE STATEMENTS, ONLY IN SUBSEQUENT STATEMENTS REFERRING TO THE VARIABLE.

MC7 - SEVERITY 2

OVERPUNCH ILLEGAL ON EQUATE

ME1 - SEVERITY 4

EXPONENT STRING OVERFLOW

E AND S LINES MAY NOT CONTAIN MORE THAN 127 CHARACTERS.

ME2 - SEVERITY 2

E-LINE CHARACTER MORE THAN ONE LINE ABOVE PRECEDING CHARACTER

MAKE SURE IN ENTERING MULTI-LINE FORMAT, THAT EACH EXPONENT LIES ON THE LINE IMMEDIATELY ABOVE THE ITEM IT MODIFIES.

ME3 - SEVERITY 2

E-LINE OVERLAPS M-LINE

ME4 - SEVERITY 2

OVERLAPPING E-LINE CHARACTERS

MO1 - SEVERITY 2

THE OVERPUNCH ON THE CHARACTER Xi??i WITHIN CHARACTER LITERAL IS NOT A LEGAL "ESCAPE" DESIGNATOR. THE OVERPUNCH HAS BEEN IGNORED.

THE ONLY LEGAL "ESCAPE" CHARACTER IS THE CENT SIGN. SEE SPEC-SECTION 2.3.4

MO2 - SEVERITY 2

INVALID OVERPUNCH ON ??

THE SYMBOL TYPE IMPLIED BY THE USER SUPPLIED OVERPUNCH DOESN'T AGREE WITH THE SYMBOL TYPE DETERMINED BY THE COMPILER.

MO3 - SEVERITY 2

MULTIPLE OVERPUNCHES NOT VALID - FIRST ACCEPTED

MO4 - SEVERITY 2

USER SUPPLIED OVERPUNCH CHARACTER NOT VALID - IGNORED

THE ONLY LEGAL OVERPUNCHES ARE *,-,.,/,+ WHICH ARE USED TO ANNOTATE MATRIX, VECTOR, BIT, CHARACTER, AND STRUCTURE DATA TYPES.

MO5 - SEVERITY 2

AN OVERPUNCH IS ILLEGAL ON THE SINGLE QUOTE DELIMITING A CHARACTER LITERAL. THE OVERPUNCH HAS BEEN IGNORED.

MO6 - SEVERITY 2

THE CHARACTER XI??I HAS BEEN USED IN A CHARACTER LITERAL "ESCAPE" CONTEXT. AN ALTERNATE CHARACTER FOR THE ESCAPE LEVEL REQUESTED IS NOT DEFINED. ?THE CHARACTER WILL BE USED WITHOUT TRANSLATION.

SEE SPEC-SECTION 2.3.4

MO7 - SEVERITY 2

WITHIN A CHARACTER LITERAL, "ESCAPE" CHARACTERS HAVE BEEN USED TO INDICATE AN ESCAPE LEVEL GREATER THAN 2 FOR THE CHARACTER Xi??i. AN ESCAPE LEVEL OF 2 WILL BE USED.

MO8 - SEVERITY 2

AN OVERPUNCH ON THE "ESCAPE" CHARACTER IN A CHARACTER LITERAL IS ILLEGAL AND HAS BEEN IGNORED.

MS1 - SEVERITY 4

SUBSCRIPT STRING OVERFLOW

MS2 - SEVERITY 2

S-LINE CHARACTER MORE THAN ONE LINE LOWER THAN PRECEDING CHARACTER

MAKE SURE IN ENTERING MULTI-LINE FORMAT, THAT EACH SUBSCRIPT LIES ON THE LINE IMMEDIATELY BELOW THE ITEM IT MODIFIES.

MS3 - SEVERITY 2

S-LINE OVERLAPS M-LINE

MS4 - SEVERITY 2

OVERLAPPING S-LINE CHARACTERS

M1 - SEVERITY 2

ILLEGAL CARD TYPE - CHANGED TO A COMMENT

THE ONLY LEGAL CARD TYPES (I.E. COLUMN 1 SYMBOLS) ARE C, D, E, S, AND M. ANY OTHER TYPES SPECIFIED WILL BE ASSUMED TO BE A C.

M2 - SEVERITY 2

INVALID SEQUENCE OF CARD TYPES

AN E (EXPONENT LINE) CARD TYPE MAY NOT BE FOLLOWED BY AN S (SUBSCRIPT LINE) CARD TYPE.

M3 - SEVERITY 1

COMMENT LONGER THAN 256 CHARACTERS - HAS BEEN TRUNCATED

COMMENTS MAY NOT EXCEED 256 CHARACTERS.

ERROR MESSAGES FOR MAJOR CLASSIFICATION P

CLASSIFICATION "P" ERRORS INDICATE FLOW CONTROL PROBLEMS

PA1 - SEVERITY 2

A MEMBER CORRESPONDING TO THE PROGRAM IDENTIFICATION ?? CANNOT BE FOUND IN THE PROGRAM ACCESS FILE. NO ACCESS VALIDATION WILL BE PERFORMED.

SEE PROGRAMMER'S GUIDE-SECTION 30

PA2 - SEVERITY 2

PROCESSING OF THE PROGRAM ACCESS FILE FOR PROGRAM ID ?? HAS CAUSED DETECTION OF ONE OR MORE ERRORS AND/OR INCONSISTENCIES WHICH ARE LISTED BELOW:

SEE PROGRAMMER'S GUIDE-SECTION 30

PC1 - SEVERITY 2

COMPOOL BLOCK CONTAINS STATEMENT(S) OTHER THAN DECLARATIONS

NO STATEMENTS OTHER THAN DECLARATIONS ARE PERMITTED IN A COMPOOL BLOCK. SPEC-SECTION 3.5

PC2 - SEVERITY 2

COMPOOL TEMPLATE CONTAINS STATEMENT(S) OTHER THAN DECLARATIONS

NO STATEMENTS OTHER THAN DECLARATIONS ARE PERMITTED IN A COMPOOL TEMPLATE. SPEC-SECTION 3.6

PE1 - SEVERITY 2

EXTERNAL TEMPLATES MUST NOT APPEAR WITHIN A BLOCK DEFINITION

TEMPLATE DEFINITIONS MUST BE INCLUDED BEFORE MAIN COMPILATION UNIT.

PE100 - SEVERITY 2

ILLEGAL NONHAL FUNCTION TYPE

ONLY INTEGERS AND SCALARS ARE ALLOWABLE. MATRICES, VECTORS, AND STRUCTURES ARE INVALID.

PE101 - SEVERITY 2

ILLEGAL NONHAL FUNCTION INVOCATION

INVOKING ILLEGAL NON-HAL FUNCTIONS IS NOT ALLOWED. HAL/S COMPILER IS UNABLE TO COMPILE THE OUTPUT SUCH FUNCTIONS GENERATE. (ISSUED IN CONJUNCTION WITH PE100).

PE2 - SEVERITY 2

EXTERNAL TEMPLATES MUST NOT BE PLACED AFTER A BLOCK DEFINITION

TEMPLATE DEFINITIONS MUST BE INCLUDED BEFORE MAIN COMPILATION UNIT.

PF1 - SEVERITY 2

RETURN FROM FUNCTION BLOCK MUST BE FOLLOWED BY AN EXPRESSION

FUNCTIONS BY THEIR DEFINITION ARE PROCEDURES WHICH RETURN SOME VALUE, AND THUS THE 'RETURN' STATEMENT WITHIN THE FUNCTION MUST CONTAIN A VALUE AS AN ARGUMENT.

PF100 - SEVERITY 2

CANNOT RETURN VALUE FROM NON-FUNCTION

FUNC(X) HAS A VALUE ONLY IF FUNC IS A FUNCTION. IF IT'S A PROCEDURE, CALL FUNC(X) MUST BE USED.

PF2 - SEVERITY 2

RETURN MAY ONLY BE FOLLOWED BY AN EXPRESSION IN A FUNCTION BLOCK

RETURN STATEMENTS IN PROCEDURE BLOCKS SIMPLY RETURN CONTROL TO THE CALLING PROGRAM, AND MUST BE CODED AS A SINGLE WORD. FUNCTION BLOCKS REQUIRE, E.G., RETURN X OR RETURN P+Q, CAUSING THE VALUE SPECIFIED TO BE SUBSTITUTED FOR THE FUNCTION INVOCATION.

PF3 - SEVERITY 2

EXPRESSION TO BE RETURNED MAY NOT POSSESS ARRAYNESS

FUNCTIONS MAY NOT RETURN ARRAYS AS VALUES (ALTHOUGH, IF THE ARGUMENT TO THE FUNCTION IS ARRAYED, ELEMENT BY ELEMENT EVALUATION WILL OCCUR, RESULTING IN AN ARRAY OF SINGLE FUNCTION VALUES).

PF4 - SEVERITY 2

ILLEGAL TYPE CONVERSION OF RETURNED EXPRESSION REQUIRED.

ONLY INTEGER TO SCALAR, SCALAR TO INTEGER, AND INTEGER AND SCALAR TO CHARACTER IMPLICIT CONVERSIONS ARE PERMITTED.

PF5 - SEVERITY 2

MATRIX DIMENSIONS OF FUNCTION DISAGREE WITH THOSE OF RETURN EXPRESSION

THE DIMENSIONS DECLARED IN THE FUNCTION HEADER MUST AGREE PRECISELY WITH THE LENGTH OF ALL VALUES SENT IN RETURN STATEMENTS. SPEC-SECTION 7.5

PF6 - SEVERITY 2

VECTOR LENGTH OF FUNCTION DISAGREES WITH THAT OF RETURN EXPRESSION

THE LENGTH DECLARED IN THE FUNCTION HEADER MUST AGREE PRECISELY WITH THE LENGTH OF ALL VALUES SENT IN RETURN STATEMENTS. SPEC-SECTION 7.5

PF7 - SEVERITY 2

TREE ORGANIZATION OF FUNCTION DOES NOT MATCH THAT OF RETURN EXPRESSION

THE TREE ORGANIZATION IMPLIED BY THE STRUCTURE TEMPLATE REFERRED TO IN THE FUNCTION HEADER MUST BE IDENTICAL TO THAT OF ALL STRUCTURES SENT IN RETURN STATEMENTS.

PF9 - SEVERITY 2

RETURN EXPRESSION MAY NOT BE A NAME PSEUDO-FUNCTION OR NULL.

ARGUMENTS IN A RETURN STATEMENT MAY CONTAIN ONLY A VALUE, AND NOT A NAME PSEUDO-FUNCTION.

PL1 - SEVERITY 2

THE FUNCTION ?? HAS BEEN DECLARED BUT NOT DEFINED

IF "DECLARE F FUNCTION SCALAR" APPEARS IN THE DECLARATION GROUP, A FUNCTION DEFINITION HEADED BY "F: FUNCTION SCALAR" MUST APPEAR SOMEWHERE IN THE CODE BLOCK.

PL10 - SEVERITY 2

THE BLOCK BEING DEFINED IS INVOKED FROM OUTSIDE THE ENCLOSING DO...END GROUP

PL11 - SEVERITY 2

ENCLOSING DO...END GROUP

PL2 - SEVERITY 3

?? IS A DUPLICATE LABEL

ALL BLOCK LABELS MUST BE UNIQUE THROUGHOUT A UNIT OF COMPILATION. SPEC-SECTION 3.8

PL3 - SEVERITY 2

LABEL ON CLOSE DOES NOT MATCH BLOCK DEFINITION
LABEL: ??

IF A CLOSE STATEMENT HAS CLOSE FOLLOWED BY A LABEL, THE LABEL SPECIFIED MUST BE IDENTICAL TO THE ONE AT THE OPENING OF THE BLOCK. SPEC-SECTION 3.7.4

PL4 - SEVERITY 2

FUNCTION LABEL CONFLICT

THE FUNCTION LABEL OR THE FUNCTION DEFINITION MUST BE DECLARED BEFORE INVOCATION. TYPE ATTRIBUTES BOTH IN THE DECLARE STATEMENT AND ON THE FUNCTION HEADER MUST AGREE.

PL5 - SEVERITY 2

LABEL ?? IS NOT DEFINED WITHIN THE CURRENT SCOPE

A LABEL IS NOT ACCESSIBLE FROM WITHIN ANY CODE BLOCKS NESTED WITHIN THE BLOCK WHERE IT OCCURS. THIS PREVENTS GO TO STATEMENTS FROM CAUSING A BRANCH OUT OF A CODEBLOCK. SEE SPEC-SECTION 3.8

PL6 - SEVERITY 2

A DEFINITION BLOCK FOR THE PROCEDURE OR TASK ?? IS ABSENT FROM THE COMPILATION

ALL PROCEDURE OR TASK BLOCKS REFERRED TO MUST BE INCLUDED IN THE COMPILATION UNIT, UNLESS THEY HAVE BEEN EXPLICITLY DECLARE'D AS EXTERNAL.

PL7 - SEVERITY 2

USED IN A CALL STATEMENT, PROCEDURE LABEL ?? MAY NOT POSSESS ARRAYNESS.

LABELS MAY NOT POSSESS ARRAYNESS.

PL8 - SEVERITY 2

THE PROCEDURE ?? MAY NOT BE CALLED FROM OUTSIDE THE DO GROUP WITHIN WHICH IT WAS DEFINED

PL9 - SEVERITY 2

THE FUNCTION ?? MAY NOT BE INVOKED FROM OUTSIDE THE DO GROUP WITHIN WHICH IT AS DEFINED

PM1 - SEVERITY 3

DUPLICATE DEFINITION FOR ??

ALL NAMES MUST BE UNIQUE WITHIN THE NAME SCOPE WHERE THEY ARE KNOWN. SEE SPEC-SECTION 3.8 FOR NAME-SCOPING RULES.

PM2 - SEVERITY 3

DUPLICATE DEFINITION OF STRUCTURE TEMPLATE ??

STRUCTURE TEMPLATE NAMES MAY DUPLICATE OTHER KINDS OF NAMES WITHIN THE SAME NAME SCOPE, BUT MUST NOT DUPLICATE OTHER STRUCTURE TEMPLATE NAMES. SEE SPEC-SECTIONS 3.8 AND 4.3 FOR NAME-SCOPING RULES.

PM3 - SEVERITY 2

EARLIER DEFINITION OVERRIDDEN FOR ??

A VARIABLE NAME DECLARED IN AN OUTER SCOPE HAS BEEN REDEFINED AS A LABEL IN THE CURRENT SCOPE, AND THE LATTER DEFINITION OVERRIDES THE PREVIOUS.

PM4 - SEVERITY 1

OUTER DEFINITION OVERRIDDEN FOR ??

A NAME DECLARED IN AN OUTER SCOPE IS BEING ACCESSED AS A STATEMENT LABEL (AS IN 'GOTO NAME').

PP1 - SEVERITY 2

A ?? DEFINITION MUST BE THE OUTERMOST BLOCK DEFINITION

COMPOOLS AND PROGRAMS CANNOT BE IMBEDDED IN CODE BLOCKS, BUT MUST BE OUTERMOST COMPILATION UNITS.

PP10 - SEVERITY 3

INLINE FUNCTIONS MAY NOT BE NESTED WITHIN INLINE FUNCTION BLOCKS.

SEE SPEC-SECTION 11.2.1

PP11 - SEVERITY 3

INLINE FUNCTIONS MUST NOT APPEAR IN EXPRESSIONS WHICH ARE REQUIRED TO BE EVALUATED AT COMPILE TIME

PP2 - SEVERITY 2

BLOCK DEFINITION IS NOT THE FIRST OUTERMOST BLOCK DEFINITION

IT IS INVALID TO FOLLOW THE MAIN COMPILATION BODY WITH ANOTHER PROGRAM. FOR INSTANCE:

```
TEST: PROGRAM;
CLOSE TEST;
TEST1: PROGRAM
CLOSE TEST1;
```

IS INCORRECT.

PP3 - SEVERITY 2

A ?? DEFINITION CANNOT BE AN OUTERMOST BLOCK DEFINITION

TASK AND UPDATE BLOCKS MUST BE EMBEDDED WITHIN CODE BLOCKS.

PP4 - SEVERITY 3

NO BLOCK DEFINITIONS WERE ENCOUNTERED IN COMPILATION

EVERY COMPILATION MUST CONTAIN A PROGRAM, FUNCTION, PROCEDURE, OR COMPOOL DEFINITION.

PP5 - SEVERITY 2

AN INLINE FUNCTION MAY NOT CONTAIN AN I/O STATEMENT.

SEE SPEC-SECTION 11-3

PP6 - SEVERITY 2

AN INLINE FUNCTION MAY NOT CONTAIN A REAL TIME STATEMENT.

SEE SPEC-SECTION 11.2.1

PP7 - SEVERITY 2

AN INLINE FUNCTION MAY NOT CONTAIN A PROCEDURE CALL.

SEE SPEC-SECTION 11.2.1

PP8 - SEVERITY 2

AN INLINE FUNCTION MAY NOT CONTAIN A USER FUNCTION INVOCATION.

SEE SPEC-SECTION 11.2.1

PP9 - SEVERITY 2

AN INLINE FUNCTION MAY NOT CONTAIN A PROCEDURE OR FUNCTION DEFINITION BLOCK

SEE SPEC-SECTION 11.2.1 DEFINITION BLOCK.

PR1 - SEVERITY 1

CLOSE OF TASK OR PROGRAM BLOCK REACHED BUT PREVIOUS STATEMENT IS NOT A %SVCI

PR2 - SEVERITY 1

CLOSE OF TASK OR PROGRAM BLOCK REACHED BUT NO %SVCI MACRO WAS FOUND IN THE BLOCK. A TERMINATE SVC HAS BEEN GENERATED AT THE CLOSE OF THE BLOCK

PR3 - SEVERITY 1

THE ON ERROR STATEMENT IS NOT WITHIN A PROGRAM OR TASK BLOCK

PR4 - SEVERITY 1

ONLY ONE ON ERROR <STATEMENT> FORM IS ALLOWED

PR5 - SEVERITY 1

ONLY ONE <ON ERROR> ALTERNATE ENTRY POINT IS ALLOWED PER TASK OR PROGRAM

PR6 - SEVERITY 2

THE NAME ?? DOES NOT SATISFY HAL/S NAME UNIQUENESS CRITERIA FOR TASKS. NAMES OF ALL TASKS AND PROGRAMS MUST BE UNIQUE TO SEVEN CHARACTERS

PS1 - SEVERITY 2

EXTERNAL PROCEDURE/FUNCTION TEMPLATE CONTAINS STATEMENT(S) OTHER THAN DECLARATIONS

NO EXECUTABLE STATEMENTS ARE PERMITTED IN AN EXTERNAL PROCEDURE/FUNCTION TEMPLATE, SINCE ITS PURPOSE IS ONLY TO POINT TO A PROCEDURE OR FUNCTION WHICH HAS BEEN INDEPENDENTLY COMPILED. SEE SPEC-SECTION 3.6

PS10 - SEVERITY 2

MULTIPLE APPEARANCE OF ACCESS KEYWORD

THE APPEARANCE OF TWO OR MORE ACCESS KEYWORDS IS INVALID. EXAMPLE: LABEL:COMPOOL ACCESS ACCESS.

PS11 - SEVERITY 2

MULTIPLE APPEARANCE OF RIGID KEYWORD

THE APPEARANCE OF TWO OR MORE RIGID KEYWORDS IS INVALID.

PS12 - SEVERITY 2

RIGID KEYWORD MAY ONLY APPEAR IN A COMPOOL BLOCK OR TEMPLATE

SEE SPEC-SECTION 4.5

PS13 - SEVERITY 2

'REMOTE' KEYWORD MAY ONLY BE USED ON EXTERNAL COMPOOL TEMPLATES.

SEE SPEC-SECTION 4.5

PS2 - SEVERITY 2

ONLY PROCEDURES OR FUNCTIONS MAY BE DESIGNATED ??

ONLY PROCEDURES OR FUNCTIONS MAY BE DESIGNATED EXCLUSIVE OR REENTRANT.

PS3 - SEVERITY 2

ILLEGAL ACCESS ATTRIBUTE OR BLOCK HEADER.

ACCESS BLOCK HEADERS MAY NOT BE APPLIED TO UPDATE OR TASK BLOCKS, NOR TO PROCEDURES OR FUNCTIONS WHICH ARE NOT EXTERNAL. THE ACCESS ATTRIBUTE MAY ONLY APPLY TO SIMPLE VARIABLE AND MAJOR STRUCTURE NAMES IN A COMPOOL BLOCK OR COMPOOL TEMPLATE. SPEC-SECTIONS 3.7, 4.5

PS4 - SEVERITY 2

THE ACCESS ATTRIBUTE MAY ONLY BE USED ON THE DEFINITION OF AN OUTERMOST BLOCK.

THE ACCESS ATTRIBUTE MAY ONLY BE DESIGNATED IN THE OUTERMOST COMPOOL OR CODE BLOCK. NESTED BLOCKS CANNOT CONTAIN THE ACCESS KEYWORD.

PS5 - SEVERITY 2

THE PROGRAM NAMED ?? IS ACCESS CONTROLLED. THE CURRENT COMPILATION UNIT IS NOT AUTHORIZED TO SCHEDULE THIS PROGRAM.

IN ORDER TO OBTAIN ACCESS TO THE NAMED PROGRAM, A REFERENCE TO IT SHOULD BE INCLUDED IN THE PROGRAM ACCESS FILE. SEE PROGRAMMER'S GUIDE-SECTION 30

PS6 - SEVERITY 2

THE PROCEDURE NAMED ?? IS ACCESS CONTROLLED. THE CURRENT COMPILATION UNIT IS NOT AUTHORIZED TO CALL THIS PROCEDURE.

IN ORDER TO OBTAIN ACCESS TO THE NAMED PROCEDURE, A REFERENCE TO IT SHOULD BE INCLUDED IN THE PROGRAM ACCESS FILE. SEE PROGRAMMER'S GUIDE-SECTION 30

PS7 - SEVERITY 2

THE FUNCTION NAMED ?? IS ACCESS CONTROLLED. THE CURRENT COMPILATION UNIT IS NOT AUTHORIZED TO INVOKE THIS FUNCTION.

IN ORDER TO OBTAIN ACCESS TO THE NAMED PROCEDURE, A REFERENCE TO IT SHOULD BE INCLUDED IN THE PROGRAM ACCESS FILE. SEE PROGRAMMER'S GUIDE-SECTION 30

PS8 - SEVERITY 2

THE VARIABLE NAMED ?? IS ACCESS CONTROLLED. THE CURRENT COMPILATION UNIT IS NOT AUTHORIZED TO CHANGE THE VALUE OF THIS VARIABLE.

IN ORDER TO BE ABLE TO CHANGE THE NAMED VARIABLE, A REFERENCE TO IT SHOULD BE INCLUDED IN THE PROGRAM ACCESS FILE. SEE PROGRAMMER'S GUIDE-SECTION 30

PS9 - SEVERITY 2

VARIABLE ?? IS DEFINED WITHIN A COMPOOL BLOCK WHICH IS ACCESS PROTECTED. THE VARIABLE MAY NOT BE USED BY THIS COMPILATION UNIT.

IN ORDER TO OBTAIN ACCESS TO THE VARIABLES IN THE NAMED COMPOOL, A REFERENCE TO IT SHOULD BE INCLUDED IN THE PROGRAM ACCESS FILE. PROGRAMMER'S GUIDE-SECTION 30

PT1 - SEVERITY 2

TASK DEFINITIONS OR DECLARATIONS MAY ONLY APPEAR IN THE OUTER MOST BLOCK OF A PROGRAM COMPILATION

SEE SPEC-SECTION 3.3

PT2 - SEVERITY 2

TASK DEFINITIONS MAY NOT APPEAR WITHIN DO GROUPS

PU3 - SEVERITY 2

INVOCATIONS IN AN UPDATE BLOCK OF PROCEDURES OR USER FUNCTIONS DEFINED OUTSIDE THE BLOCK ARE ILLEGAL.

SEE SPEC-SECTION 3.4

P1 - SEVERITY 2

END-OF-FILE AT INVALID POINT IN SOURCE TEXT

EVERY UNIT OF COMPILATION MUST END WITH A CLOSE STATEMENT.

P4 - SEVERITY 3

CONFLICTING USE OF ??

P5 - SEVERITY 2

TOO MANY MACRO EXPANSIONS FOR ??

P6 - SEVERITY 2

PROGRAM LAYOUT TABLE EXCEEDED

THE PROGRAM LAYOUT TABLE CANNOT CONTAIN AN EXCESS OF 255 PROGRAM BLOCKS.

A PREVIOUSLY DEFINED VARIABLE HAS BEEN USED AS EITHER A PROCEDURE LABEL OR STATEMENT LABEL.

P8 - SEVERITY 2

THE FOLLOWING SYMBOL IS SYNTACTICALLY ILLEGAL IN THE CONTEXT USED: ??
ERROR RECOVERY MAY CAUSE SUBSEQUENT SPURIOUS ERRORS

CHECK TO MAKE SURE THAT MISTYPING HAS NOT PRODUCED GIBBERISH. ALSO BE SURE THAT A HAL/S RESERVED KEYWORD IS NOT BEING INADVERTENTLY USED AS AN IDENTIFIER OR LABEL. SEE SPEC, APPENDIX B FOR A LIST OF THESE KEYWORDS.

ERROR MESSAGES FOR MAJOR CLASSIFICATION Q

CLASSIFICATION "Q" ERRORS DEAL WITH SHAPING FUNCTIONS

QA1 - SEVERITY 2

ARRAYNESS OF SINGLE ARGUMENT OF INTEGER/SCALAR CONVERSION FUNCTION DOES NOT MATCH THAT OF EXPRESSION CONTAINING FUNCTION

THE RESULT OF APPLYING THE INTEGER OR SCALAR FUNCTION TO A LIST OF N ELEMENTS IS A 1-DIMENSIONAL ARRAY OF LENGTH N. THIS MUST MATCH THE CURRENT ARRAYNESS OF THE EXPRESSION CONTAINING THE FUNCTION. SEE SPEC-SECTION 6.5.1

QA2 - SEVERITY 2

ARRAYNESS OF RESULT OF INTEGER/SCALAR CONVERSION FUNCTION IS UNCOMPUTABLE

THE LENGTH OF THE ARRAY PRODUCED BY THE INTEGER OR SCALAR FUNCTIONS MUST BE COMPUTABLE AT COMPILE TIME. SEE SPEC-SECTION 6.5.1

QA3 - SEVERITY 2

SPECIFIED ARRAYNESS OF INTEGER/SCALAR CONVERSION FUNCTION IS INCONSISTENT WITH NUMBER OF DATA ELEMENTS SUPPLIED IN ARGUMENT LIST

THE SUBSCRIPTS TO THE INTEGER AND SCALAR FUNCTIONS DESCRIBE THE DIMENSIONALITY OF THE ARRAY TO BE PRODUCED. THUS, E.G., INTEGER(2+3,6) WILL PRODUCE A 5X6 ARRAY. ACCORDINGLY, THE NUMBER OF LIST ELEMENTS SPECIFIED AS THE ARGUMENT TO THE FUNCTION MUST EQUAL THE PRODUCT OF THE SUBSCRIPTS. SPEC-SECTION 6.5.1

QA4 - SEVERITY 2

ARRAYNESS OF RESULT OF INTEGER/SCALAR CONVERSION FUNCTION DOES NOT MATCH THAT OF EXPRESSION CONTAINING FUNCTION

THE ARRAYNESS OF THE RESULT OF APPLYING THE INTEGER OR SCALAR FUNCTIONS, AS SPECIFIED BY THE SUBSCRIPTS OR IMPLIED BY THE LENGTH OF THE ARGUMENT LIST, MUST MATCH THE CURRENT ARRAYNESS OF THE EXPRESSION CONTAINING THE FUNCTION. SEE SPEC-SECTION 6.5.1

QD1 - SEVERITY 2

DIMENSIONS OF VECTOR/MATRIX CONVERSION FUNCTION DO NOT AGREE WITH THE NUMBER OF DATA ELEMENTS SUPPLIED IN THE ARGUMENT LIST

THE DIMENSIONS OF THE RESULT PRODUCED BY APPLYING THE VECTOR OR MATRIX FUNCTIONS ARE SPECIFIED BY THE SUBSCRIPTS TO THESE FUNCTIONS, OR DEFAULT TO A 3-VECTOR OR A 3X3 MATRIX. ACCORDINGLY, THE NUMBER OF LIST ELEMENTS SPECIFIED AS THE ARGUMENT TO THE FUNCTION MUST EQUAL THE PRODUCT OF THE SUBSCRIPTS (OR, IN THE DEFAULT CASE, 3 OR 9).

QD100 - SEVERITY 2

LEVEL MISMATCH ON SHAPING FUNCTION ARGUMENT.

INTERNAL COMPILER FAILURE.

QD2 - SEVERITY 2

BIT OR CHARACTER CONVERSION FUNCTION MAY ONLY HAVE ONE ARGUMENT

SEE SPEC-SECTIONS 6.5.2, 6.5.3

QS1 - SEVERITY 2

COLONS AND SEMICOLONS MAY NOT APPEAR IN SUBSCRIPT OF CONVERSION FUNCTIONS

ONLY COMMAS ARE PERMITTED. IN INTEGER, SCALAR, VECTOR AND MATRIX, SUBSCRIPTING INDICATES THE DIMENSIONALITY OF THE RESULT; IN BIT AND CHARACTER IT INDICATES A CHOICE FROM THE BIT OR CHARACTER STRING CREATED.

QS10 - SEVERITY 2

BIT OR CHARACTER CONVERSION FUNCTION MAY ONLY HAVE ONE SUBSCRIPT

ARRAY SUBSCRIPTING IS NOT PERMITTED IN THIS CONTEXT. TO CHOOSE THE FIFTH CHARACTER FROM THE EIGHT ARRAY ELEMENT OF CHARACTER(A), WHERE A IS AN ARRAY OF CHARACTER STRINGS, WRITE:

```
(CHARACTER  ?(A  ?))
 5  8
      ...NOT...
 CHARACTER  (A).
 8:5
```

QS11 - SEVERITY 2

SUBBIT CONVERSION FUNCTION MAY ONLY HAVE ONE SUBSCRIPT

SEE SPEC-SECTION 6.5.4

QS12 - SEVERITY 2

COLONS AND SEMICOLONS MAY NOT APPEAR IN THE SUBSCRIPT OF A SUBBIT PSEUDO-VARIABLE

SEE SPEC-SECTION 6.5.4

QS13 - SEVERITY 2

SUBSCRIPT OF SUBBIT PSEUDO-VARIABLE MAY NOT CONTAIN A PRECISION QUALIFIER.

SEE SPEC-SECTION 6.5.4

QS2 - SEVERITY 2

MATRIX CONVERSION FUNCTION DOES NOT HAVE TWO SUBSCRIPTS

THE MATRIX CONVERSION FUNCTION MUST HAVE TWO SUBSCRIPTS TO SPECIFY ROW AND COLUMN DIMENSIONS, OR ELSE NO SUBSCRIPTS AT ALL, IN WHICH CASE A 3X3 MATRIX IS ASSUMED. SPEC-SECTION 6.5.1

QS3 - SEVERITY 2

VECTOR CONVERSION FUNCTION DOES NOT HAVE ONE SUBSCRIPT

THE VECTOR CONVERSION MAY HAVE NO MORE THAN ONE SUBSCRIPT. IF A SUBSCRIPT OCCURS, IT SPECIFIES THE LENGTH OF THE VECTOR, AND IF NOT, A LENGTH OF 3 IS ASSUMED. SPEC-SECTION 6.5.1

QS4 - SEVERITY 2

INTEGER OR SCALAR CONVERSION FUNCTION HAS MORE THAN MAXIMUM PERMITTED NUMBER OF SUBSCRIPTS

SINCE NO MORE THAN 3 DIMENSIONS ARE PERMITTED IN AN ARRAY, NO MORE THAN 3 SUBSCRIPTS MAY APPEAR IN THE INTEGER OR SCALAR FUNCTION. SEE SPEC-SECTION 6.5.1

QS5 - SEVERITY 2

SUBSCRIPT OF ARITHMETIC CONVERSION FUNCTION IS NOT A SINGLE INDEX

THE AT, TO, *, CONVENTIONS ARE NOT PERMITTED IN THE CONTEXT OF THE BUILT-IN SHAPING FUNCTIONS INTEGER, SCALAR, VECTOR, MATRIX, SINCE SUBSCRIPTING HERE DOES NOT SERVE TO PICK OUT ELEMENTS, BUT RATHER TO SPECIFY THE SHAPE OF THE RESULTING DATA ITEM.

QS6 - SEVERITY 2

SUBSCRIPT OF ARITHMETIC CONVERSION FUNCTION MAY NOT CONTAIN # VALUES

THE CONVENTION IS NOT PERMITTED IN THE CONTEXT OF THE BUILT-IN SHAPING FUNCTIONS INTEGER, SCALAR, VECTOR, MATRIX, SINCE SUBSCRIPTING HERE DOES NOT SERVE TO PICK OUT ELEMENTS, BUT RATHER TO SPECIFY THE SHAPE (DIMENSIONS) OF THE RESULTING DATA ITEM. SEE SPEC-SECTION 6.5.1

QS7 - SEVERITY 2

SUBSCRIPT OF ARITHMETIC CONVERSION FUNCTION MUST BE AN UNARRAYED INTEGER/SCALAR EXPRESSION COMPUTABLE AT COMPILE TIME

SEE SPEC, APPENDIX F.

QS8 - SEVERITY 2

VALUE OF SUBSCRIPT OF ARITHMETIC CONVERSION FUNCTION LIES OUTSIDE LEGAL RANGE

FOR VECTOR OR MATRIX SHAPING FUNCTIONS, THE DIMENSION SIZE DEFINED BY THE SUBSCRIPT MUST BE IN THE RANGE: 1<SIZE<=64. FOR ARRAYED INTEGER OR SCALAR CONVERSION FUNCTION, THE ARRAY SIZE MUST BE IN THE RANGE: 1<SIZE<=32767, AND THE TOTAL NUMBER OF ELEMENTS MAY NOT BE GREATER THAN 32767.

QS9 - SEVERITY 2

SUBSCRIPT OF BIT OR CHARACTER CONVERSION FUNCTION MAY NOT CONTAIN A PRECISION QUALIFIER.

THE PRECISION SPECIFICATION IS ONLY ALLOWABLE FOR ARITHMETIC CONVERSIONS (INTEGER, SCALAR, VECTOR, OR MATRIX.) SEE SPEC-SECTION 6.5.2

QX1 - SEVERITY 2

CONVERSION FUNCTIONS MAY NOT HAVE ARGUMENTS OF STRUCTURE TYPE

SEE SPEC-SECTION 6.5.1

QX2 - SEVERITY 2

MATRIX/VECTOR CONVERSION FUNCTIONS MAY NOT HAVE ARGUMENTS OF BIT TYPE

SEE SPEC-SECTIONS 6.5.1, 6.5.5

QX3 - SEVERITY 2

MATRIX/VECTOR CONVERSION FUNCTIONS MAY NOT HAVE ARGUMENTS OF CHARACTER TYPE

SEE SPEC-SECTIONS 6.5.1, 6.5.5

QX4 - SEVERITY 2

MATRIX OR VECTOR ARGUMENT IS ILLEGAL IN BIT OR CHARACTER CONVERSION FUNCTION

BIT OR CHARACTER CONVERSIONS MUST HAVE ARGUMENTS OF INTEGER, SCALAR, BIT OR CHARACTER TYPE.

SPEC-SECTIONS 6.5.1, 6.5.2, 6.5.5

QX5 - SEVERITY 2

CHARACTER CONVERSION FUNCTION WITH RADIX DOES NOT HAVE ARGUMENT OF BIT TYPE

SPEC-SECTIONS 6.5.3, 6.5.5

QX6 - SEVERITY 2

BIT CONVERSION FUNCTION WITH RADIX DOES NOT HAVE ARGUMENT OF CHARACTER TYPE

SPEC-SECTIONS 6.5.2, 6.5.5

QX7 - SEVERITY 2

IN AN ASSIGNMENT CONTEXT THE ARGUMENT OF A SUBBIT PSEUDO-VARIABLE MAY NOT ITSELF BE A SUBBIT PSEUDO-VARIABLE

NESTED SUBBIT FUNCTIONS ARE ALSO ILLEGAL AS ASSIGN PARAMETERS AND IN READ AND READALL STATEMENTS. SPEC-SECTION 6.5.4

QX8 - SEVERITY 2

ARGUMENT OF ILLEGAL TYPE IN SUBBIT PSEUDO-VARIABLE

THE ARGUMENT MUST BE OF INTEGER, SCALAR, BIT, OR CHARACTER TYPE. SPEC-SECTION 6.5.4.

QX9 - SEVERITY 2

THE ARGUMENT OF A SUBBIT PSEUDO-VARIABLE MAY NOT BE A NAME PSEUDO-FUNCTION

ERROR MESSAGES FOR MAJOR CLASSIFICATION R

CLASSIFICATION "R" ERRORS ARE RELATED TO REAL-TIME STATEMENT ERRORS

RE0 - SEVERITY 2

ILLEGAL FORM OF OFF ERROR SUBSCRIPT

EACH ERROR SUBSCRIPT MUST BE A SINGLE LITERAL NUMBER IN THE RANGE 0 TO 62. THE CORRECT FORM IS: OFF ERROR$ <GROUP:NUMBER>. SPEC-SECTION 9.1

RE1 - SEVERITY 2

ILLEGAL FORM OR VALUE OF ON ERROR SUBSCRIPT

EACH ERROR SUBSCRIPT MUST BE A SINGLE LITERAL NUMBER IN THE RANGE 1 TO 127. THE VALID RANGE OF ERROR NUMBERS IS 0 TO 62. THE CORRECT FORM IS: ON ERROR$ <GROUP:NUMBER>.

RE2 - SEVERITY 2

ILLEGAL FORM OR VALUE OF SEND ERROR SUBSCRIPT

EACH ERROR SUBSCRIPT MUST BE A SINGLE LITERAL NUMBER IN THE RANGE 1 TO 127. THE VALID RANGE OF ERROR NUMBERS IS 0 TO 62. THE CORRECT FORM IS: SEND ERROR$ <GROUP:NUMBER>.

RE3 - SEVERITY 2

TOO MANY ON ERROR STATEMENTS ACTIVE

NO MORE THAN 100 ON ERROR STATEMENTS MAY BE POTENTIALLY ACTIVE AT ANY GIVEN TIME.

RT1 - SEVERITY 2

SCHEDULE STATEMENT CONTAINS AN ILLEGAL FORM OF ?? TIMING EXPRESSION

SCHEDULE...AT, MUST BE FOLLOWED BY AN UNARRAYED INTEGER OR SCALAR EXPRESSION REPRESENTING A TIME ON THE RTE CLOCK, AT WHICH TIME THE PROCESS REFERRED TO WILL BE PUT IN THE READY STATE. SCHEDULE IN, REPEAT AFTER, OR REPEAT EVERY, MUST BE FOLLOWED BY AN EVENT EXPRESSION. UNTIL CAN BE FOLLOWED BY AN UNARRAYED INTEGER OR SCALAR EXPRESSION REPRESENTING A DURATION AFTER WHICH THE PROCESS REFERRED TO WILL BE PUT IN THE READY STATE. SCHEDULE ON, OR UNTIL, OR WHILE, MUST BE FOLLOWED BY UNARRAYED INTEGER/SCALAR EXPRESSION. SPEC-SECTION 8.3

RT10 - SEVERITY 2

AN UNLATCHED EVENT MAY NOT BE SET OR RESET

ONLY EVENT VARIABLES WHICH HAVE BEEN DECLARE'D WITH THE LATCHED ATTRIBUTE MAY BE SET OR RESET. SPEC-SECTION 8.8

RT11 - SEVERITY 2

USED IN A REAL TIME STATEMENT OR AS A PROCESS EVENT, LABEL ?? MAY NOT POSSESS ARRAYNESS.

RT14 - SEVERITY 2

?? IS NOT A PROGRAM OR TASK LABEL

DATA ITEMS REFERRED TO IN EVENT EXPRESSIONS, IF NOT EVENT VARIABLES, MUST BE NAMES OF PROGRAMS OR TASKS. LABELS EMPLOYED IN SCHEDULE, CANCEL, TERMINATE, AND UPDATE PRIORITY STATEMENTS MUST ALSO BE NAMES ONLY OF PROGRAMS OR TASKS.

RT2 - SEVERITY 2

WHILE EXPRESSION MAY NOT BE A TIMING EXPRESSION

WHILE MUST BE FOLLOWED BY AN EVENT EXPRESSION IN REAL-TIME STATEMENTS. SPEC-SECTION 8.8

RT3 - SEVERITY 2

SCHEDULE STATEMENT CONTAINS AN ILLEGAL FORM OF ?? EVENT EXPRESSION

THIS MESSAGE APPEARS FROM UTILIZING AN EVENT EXPRESSION THAT SPECIFIES AN ARRAYED EVENT, BUT WITHOUT SUBSCRIPTING DOWN TO ONE ITEM.

RT4 - SEVERITY 1

PRIORITY EXPRESSION IN ?? STATEMENT ABSENT OR ILLEGAL

SCHEDULE STATEMENTS MUST SPECIFY A PRIORITY FOR THE PROCESS SCHEDULED. PRIORITIES MUST LIE IN THE RANGE FROM 0 TO 255.

RT5 - SEVERITY 2

SCHEDULE STATEMENT CONTAINS DUPLICATED AT/IN/ON EXPRESSIONS

ONLY ONE AT, IN, OR ON MAY BE INCLUDED IN EACH SCHEDULE STATEMENT.

RT6 - SEVERITY 2

WAIT STATEMENT CONTAINS ILLEGAL FORM OF ?? EXPRESSION

A "WAIT FOR" STATEMENT MUST BE FOLLOWED BY "DEPENDENT", OR EVENT EXPRESSION. "WAIT UNTIL" OR "WAIT" ALONE MUST BE FOLLOWED BY AN ARITHMETIC EXPRESSION.

RT8 - SEVERITY 2

AN ARRAYED EVENT MAY NOT BE SIGNALED

SEE SPEC-SECTION 8.8

RT9 - SEVERITY 2

USED IN A REAL TIME STATEMENT OR AS A PROCESS EVENT, LABEL ?? MUST BE A PROGRAM OR TASK.

DATA ITEMS REFERRED TO IN EVENT EXPRESSIONS, IF NOT EVENT VARIABLES, MUST BE NAMES OF PROGRAMS OR TASKS. LABELS EMPLOYED IN SCHEDULE, CANCEL, TERMINATE, AND UPDATE PRIORITY STATEMENTS MUST ALSO BE NAMES ONLY OF PROGRAMS OR TASKS.

RU1 - SEVERITY 2

SIGNAL STATEMENTS ARE THE ONLY REAL-TIME STATEMENTS WHICH MAY APPEAR INSIDE AN UPDATE BLOCK

SEE SPEC-SECTION 3.4

ERROR MESSAGES FOR MAJOR CLASSIFICATION S

CLASSIFICATION "S" ERRORS INDICATE INCORRECT SUBSCRIPT USAGE

SC1 - SEVERITY 2

?? HAS TOO MANY STRUCTURE SUBSCRIPTS

ONLY ONE STRUCTURE SUBSCRIPT MAY BE SPECIFIED; HENCE, ONLY ONE EXPRESSION MAY APPEAR PRIOR TO A SEMICOLON IN A SUBSCRIPT EXPRESSION. SPEC-SECTION 5.3.3

SC2 - SEVERITY 2

?? HAS TOO MANY ARRAY SUBSCRIPTS

THE NUMBER OF ARRAY SUBSCRIPTS SUPPLIED (I.E., PRIOR TO A COLON, IF COMPONENT SUBSCRIPTING IS PRESENT) MUST EQUAL THE NUMBER OF ARRAY DIMENSIONS SPECIFIED FOR THE DATA ITEM BEING SUBSCRIPTED.

SC3 - SEVERITY 2

?? HAS TOO FEW ARRAY SUBSCRIPTS

THE NUMBER OF ARRAY SUBSCRIPTS SUPPLIED (I.E., PRIOR TO A COLON, IF COMPONENT SUBSCRIPTING IS PRESENT) MUST EQUAL THE NUMBER OF ARRAY DIMENSIONS SPECIFIED FOR THE DATA ITEM BEING SUBSCRIPTED - EVEN IF ALL ELEMENTS FROM SOME DIMENSION ARE BEING CHOSEN.

SC4 - SEVERITY 2

?? HAS TOO MANY COMPONENT SUBSCRIPTS

THE NUMBER OF COMPONENT SUBSCRIPTS (I.E., FOLLOWING A COLON OR SEMICOLON, IF OTHER SUBSCRIPTING IS PRESENT) MUST BE 2, IF THE DATA ITEM IS A MATRIX, AND 1 IF THE DATA ITEM IS OF BIT, CHARACTER, OR VECTOR TYPES. SPEC-SECTION 5.3.5

SC5 - SEVERITY 2

?? HAS TOO FEW COMPONENT SUBSCRIPTS

THE NUMBER OF COMPONENT SUBSCRIPTS (I.E., FOLLOWING A COLON OR SEMICOLON, IF OTHER SUBSCRIPTING IS PRESENT) MUST BE 2, IF THE DATA ITEM IS A MATRIX, AND 1 IF THE DATA ITEM IS OF BIT, CHARACTER, OR VECTOR TYPES. SPEC-SECTION 5.3.5

SP1 - SEVERITY 2

SUBSCRIPTING CONTAINS MORE THAN ONE LIST OF STRUCTURE SUBSCRIPTS

ONLY ONE STRUCTURE SUBSCRIPT MAY BE GIVEN; THUS ONLY ONE SEMICOLON MAY APPEAR IN A SUBSCRIPT. MAKE SURE MISTYPING HAS NOT CAUSED AN EXTRA SEMICOLON INSTEAD OF A COLON.

SP2 - SEVERITY 2

SUBSCRIPTING CONTAINS MORE THAN ONE LIST OF ARRAY SUBSCRIPTS

ONLY ONE LIST OF ARRAY SUBSCRIPTS MAY BE GIVEN; THUS ONLY ONE COLON MAY APPEAR IN A SUBSCRIPT. MAKE SURE MISTYPING HAS NOT CAUSED AN EXTRA COLON INSTEAD OF A SEMICOLON.

SP3 - SEVERITY 2

SUBSCRIPT CONTAINS A LEADING COLON, OR A COLON PRECEDED BY A SEMICOLON, COLON, OR COMMA

IF NO STRUCTURE SUBSCRIPTING IS PRESENT, NO SEMICOLON SHOULD BE CODED. IF REFERENCE TO ALL THE COPIES OF A STRUCTURE IS DESIRED, "*" SHOULD BE USED.

SP4 - SEVERITY 2

SUBSCRIPT CONTAINS LEADING SEMICOLON, OR A SEMICOLON PRECEDED BY A SEMICOLON, COLON, OR COMMA

IF NO ARRAY SUBSCRIPTING IS PRESENT, NO COLON SHOULD BE CODED. IF REFERENCE TO ALL THE ELEMENTS OF A 1-DIMENSIONAL ARRAY IS DESIRED, AND COMPONENT SUBSCRIPTING IS PRESENT, "*" SHOULD BE CODED.

SP5 - SEVERITY 2

SUBSCRIPT CONTAINS A LEADING COMMA, OR A COMMA PRECEDED BY A SEMICOLON, COLON, OR COMMA

IF REFERENCE TO ALL THE ELEMENTS IN A PARTICULAR ARRAY OR MATRIX DIMENSION IS DESIRED, "*" SHOULD BE CODED, AND NOT THE COMMA ALONE.

SP6 - SEVERITY 2

SUBSCRIPT IS EMPTY OR CONTAINS A TRAILING COMMA

IF REFERENCE TO ALL THE ELEMENTS IN A PARTICULAR ARRAY OR MATRIX DIMENSION IS DESIRED, "*" SHOULD BE CODED, AND NOT THE COMMA ALONE.

SQ1 - SEVERITY 2

?? IS OF INCORRECT TYPE TO POSSESS A PRECISION QUALIFIER.

ONLY INTEGER, SCALAR, VECTOR, AND MATRIX DATA TYPES (AND CONVERSION FUNCTIONS) MAY POSSESS PRECISION ATTRIBUTES.

SQ2 - SEVERITY 2

SUBSCRIPTED VARIABLE ?? MAY NOT POSSESS A PRECISION QUALIFIER.

A SUBSCRIPTED VARIABLE MAY NOT POSSESS A PRECISION QUALIFIER, BUT A SUBSCRIPTED VARIABLE WITHIN PARENTHESES IS AN "ARITHMETIC EXPRESSION", AND MAY POSSESS ONE. SEE SPEC-SECTION 5.3

SQ3 - SEVERITY 2

?? IS IN AN ASSIGNMENT CONTEXT AND THEREFORE MAY NOT POSSESS A PRECISION QUALIFIER.

THE LEFT HAND SIDE OF AN ASSIGNMENT STATEMENT MUST SIMPLY BE A VARIABLE. IF ITS PRECISION DIFFERS FROM THAT OF THE EXPRESSION ON THE RIGHT, IMPLICIT CONVERSION WILL BE MADE.

SQ4 - SEVERITY 3

SCALING QUALIFIER IS NOT OF INTEGER OR SCALAR TYPE. SEE CR12712.

THIS PROCESSING IS NOT SUPPORTED (REMOVED WITH CR12712).

SQ5 - SEVERITY 3

COMPILE-TIME EXECUTION OF SCALING OPERATION FAILED. SEE CR12712.

COMPUTATION OF THE SCALING FACTOR GENERATED A NUMBER OUTSIDE OF THE RANGE OF INTERNALLY REPRESENTABLE FLOATING POINT NUMBERS.

THIS PROCESSING IS NOT SUPPORTED (REMOVED WITH CR12712).

SR1 - SEVERITY 2

PARTITION SIZE IN SUBSCRIPT OF ?? IS UNKNOWN

PARTITION SIZES IN ALL SUBSCRIPTING MUST BE COMPUTABLE AT COMPILE TIME. THUS, IN "A$(N AT X)", N MUST BE KNOWN AT COMPILE TIME, AND IN "B$(MTO P)", BOTH M AND P MUST BE KNOWN AT COMPILE TIME.

SR100 - SEVERITY 2

SUBBIT SUBSCRIPT OUT OF RANGE

A NON-EXISTENT ELEMENT OF THE BIT STRING IS BEING REFERRED TO.

SR2 - SEVERITY 2

PARTITION SIZE IN SUBSCRIPT OF ?? IS EITHER LESS THAN THE LEGAL MINIMUM, OR IMPLIES AN INDEX VALUE EXCEEDING THE LEGAL MAXIMUM

PARTITION SIZES MUST LIE IN THE RANGE FROM 1 TO THE TOTAL LENGTH OF THE DIMENSION REFERRED TO. IN "A$(N AT X)", THEREFORE, N+X MUST NOT BE GREATER THAN THE TOTAL LENGTH; IN "B$(M TO P)", M MUST NOT BE LESS THAN 1, ETC.

SR3 - SEVERITY 2

INDEX VALUE IN SUBSCRIPT OF ?? IS GREATER THAN THE LEGAL MAXIMUM

THE INDEX VALUE EXCEEDS THE TOTAL LENGTH OF THE DIMENSION REFERRED TO.

SR4 - SEVERITY 2

INDEX VALUE IN SUBSCRIPT OF ?? IS LESS THAN 1

THE VALUE OF A SUBSCRIPT EXPRESSION MAY NOT BE LESS THAN 1.

SR5 - SEVERITY 2

?? CONTAINED AN ILLEGAL # SUBSCRIPT

A SUBSCRIPT OCCURRED WHERE NO SUBSCRIPTING WAS ALLOWED.

SR6 - SEVERITY 2

INDEX VALUE IN SUBSCRIPT OF ?? IS UNKNOWN

THE SUBBIT FUNCTION WAS APPLIED AND THE SUBSCRIPT SIZE IS INDETERMINATE.

SS1 - SEVERITY 2

IN SUBSCRIPT OF ?? ONLY TRAILING ASTERISKS MAY BE OMITTED

TO CHOOSE ALL OF SOME SPECIFIC STRUCTURE, ARRAY, OR VECTOR-MATRIX DIMENSION, USE THE * CONVENTION: THE ONLY TIME THIS IS OPTIONAL IS IF THE SUBSCRIPT SPECIFIED IS THE LAST ONE IN THE SUBSCRIPT LIST. SEE SPEC-SECTION 5.3

ST1 - SEVERITY 2

A SUBSCRIPT OF ?? WAS NOT OF INTEGER OR SCALAR TYPE

SUBSCRIPTS MUST ALWAYS BE ONLY OF INTEGER OR SCALAR TYPE, ALTHOUGH THEY MAY BE ARRAYED. SPEC-SECTION 5.3

SV1 - SEVERITY 2

SUBSCRIPTING OF ?? IS ILLEGAL IN CONTEXT OF USE AS AN ASSIGN ARGUMENT

COMPONENT SUBSCRIPTING IS NOT PERMITTED FOR ASSIGN ARGUMENTS TO PROCEDURES OF BIT OR CHARACTER TYPES. ALL OTHER SUBSCRIPTING MUST BE DONE SO THAT A SINGLE UNARRAYED ELEMENT IS REFERRED TO. SPEC-SECTION 7.4

SV2 - SEVERITY 2

USER SUPPLIED OVERPUNCH NOT CONSISTENT WITH SUBSCRIPTING FOR VARIABLE ??

MAKE SURE THAT SUBSCRIPTING HAS NOT CAUSED A NEW DATA TYPE TO EXIST. IF U IS A 3-VECTOR, AND M, A 3X3 MATRIX, V(1) AND M(2,3) ARE NONETHELESS SCALARS; AND M(*:2) IS A VECTOR. SEE SPEC - SECTION 5.3

SV3 - SEVERITY 2

?? MAY NOT POSSESS SUBSCRIPTS

INTEGER, SCALAR, AND EVENT VARIABLES MUST POSSESS ARRAYNESS OR BE TERMINALS OF STRUCTURES WITH MULTIPLE COPIES TO BE ABLE TO BE SUBSCRIPTED. FUNCTIONS, PROCEDURES, BUILT-IN FUNCTIONS, KEYWORDS, ETC. MAY NOT BE SUBSCRIPTED (WITH SOME EXCEPTIONS). SEE SPEC - SECTION 5.3.1

ERROR MESSAGES FOR MAJOR CLASSIFICATION T

CLASSIFICATION "T" ERRORS DEAL WITH INPUT/OUTPUT STATEMENTS

TC1 - SEVERITY 2

ARGUMENT OF I/O CONTROL FUNCTION MUST BE OF UNARRAYED INTEGER OR SCALAR TYPE

SEE SPEC-SECTION 10.1.3

TD1 - SEVERITY 2

I/O DEVICE NUMBER IS NOT IN RANGE 0 THROUGH ??

DEVICE NUMBERS MUST BE IN THE RANGE 0-9

TD2 - SEVERITY 2

RECORD ADDRESS IS NOT AN UNARRAYED INTEGER OR SCALAR

SEE SPEC-SECTION 10.2

T1 - SEVERITY 2

VARIABLE IN READALL IS NOT OF CHARACTER TYPE

THE VARIABLE SPECIFIED IN A READALL STATEMENT MUST BE OF CHARACTER TYPE: OTHERWISE, USE THE READ STATEMENT. SPEC-SECTION 10.1.1

T2 - SEVERITY 2

VARIABLE IN READ MAY NOT BE OF EVENT TYPE

T3 - SEVERITY 2

VARIABLE IN READ/READALL MAY NOT BE A SUBBIT PSEUDO-VARIABLE

SPEC-SECTION 6.5.4

T4 - SEVERITY 2

A FILE STATEMENT MAY NOT READ INTO A SUBBIT PSEUDO-VARIABLE

T5 - SEVERITY 2

AN I/O STATEMENT MAY NOT CONTAIN A NAME PSEUDO-FUNCTION OR NULL.

T6 - SEVERITY 2

I/O STATEMENT CONTAINS A STRUCTURE WHOSE TEMPLATE CONTAINS AT LEAST ONE TERMINAL NODE WITH THE NAME ATTRIBUTE.

STRUCTURE TERMINALS WITH THE NAME ATTRIBUTE ARE SIMPLY IGNORED IN SEQUENTIAL I/O OPERATIONS: NO VALUE IS READ OR WRITTEN. SEE SPEC-SECTION 11.4.11

T7 - SEVERITY 2

IN FILE STATEMENT VARIABLE TO BE READ POSSESSES AN ILLEGAL SUBSCRIPT.

COMPONENT SUBSCRIPTING IS NOT PERMITTED ON VARIABLES OF BIT OR CHARACTER TYPE USED IN FILE I/O. OTHER TYPES OF VARIABLES, IF SUBSCRIPTED, MUST BE SUBSCRIPTED SO AS TO PRODUCE A SINGLE UNARRAYED ELEMENT. SPEC-SECTION 10.2

T8 - SEVERITY 2

IN FILE STATEMENT VARIABLE TO BE READ INTO MAY ONLY POSSESS MULTIPLE COPIES IF IT IS A MAJOR STRUCTURE NAME.

SEE SPEC-SECTION 10.2

ERROR MESSAGES FOR MAJOR CLASSIFICATION U

CLASSIFICATION "U" ERRORS DEAL WITH UPDATE BLOCKS

UI1 - SEVERITY 2

?? IS LOCKED: IT MAY ONLY APPEAR IN UPDATE BLOCKS OR ASSIGN ARGUMENT LISTS.

SEE SPEC-SECTION 8.10

U12 - SEVERITY 2

UPDATE BLOCK DEFINITION MAY NOT APPEAR INSIDE AN UPDATE BLOCK

SEE SPEC-SECTION 3.4

UP1 - SEVERITY 2

THE PROCEDURE, TASK, OR PROGRAM ?? MAY NOT BE INVOKED WITHIN THE CURRENT UPDATE BLOCK

BLOCKS NOT NESTED WITHIN AN UPDATE BLOCK MAY NOT BE INVOKED FROM WITHIN IT. SPEC-SECTION 3.4

UP2 - SEVERITY 2

UPDATE BLOCKS MAY NOT CONTAIN RETURN STATEMENTS

UT1 - SEVERITY 2

I/O STATEMENTS ARE ILLEGAL INSIDE UPDATE BLOCKS

SEE SPEC-SECTION 3.4

ERROR MESSAGES FOR MAJOR CLASSIFICATION V

CLASSIFICATION "V" ERRORS ARE RELATED TO COMPILE-TIME VARIABLE ERRORS

VA1 - SEVERITY 2

COMPILE TIME INTEGER/SCALAR ADDITION FAILED.

DURING COMPILE TIME, AN OVERFLOW OR UNDERFLOW OCCURRED. INTERMEDIATE CODE HAS GENERATED RUNTIME EXECUTABLE CODE FOR ADDITION. THIS TAKES PLACE BECAUSE ALTHOUGH THE HOST MACHINE WAS NOT ABLE TO HANDLE THE CALCULATIONS, IT IS PRESUMED THE TARGET COMPUTER MAY HAVE THE CAPABILITY.

VA2 - SEVERITY 2

COMPILE-TIME INTEGER/SCALAR SUBTRACTION FAILED

SAME AS ABOVE, BUT FOR SUBTRACTION

VA3 - SEVERITY 2

COMPILE TIME INTEGER/SCALAR MULTIPLICATION FAILED

SAME AS ABOVE, BUT FOR MULTIPLICATION

VA4 - SEVERITY 2

COMPILE-TIME INTEGER/SCALAR DIVISION FAILED

SAME AS ABOVE, BUT FOR DIVISION

VA5 - SEVERITY 2

COMPILE-TIME INTEGER/SCALAR EXPONENTIATION FAILED

SAME AS ABOVE, BUT FOR EXPONENTIATION

VC1 - SEVERITY 2

COMPILE-TIME CATENATION PRODUCED TOO LONG A CHARACTER STRING - TRUNCATED TO 255 CHARACTERS

CHARACTER LITERALS AND VARIABLES MAY NOT HAVE A LENGTH GREATER THAN 255.

VE1 - SEVERITY 2

AN EXPRESSION NOT COMPUTABLE AT COMPILE-TIME HAS BEEN USED IN A CONTEXT WHERE A VALUE MUST BE KNOWN

ALL VALUES SPECIFIED IN DECLARE STATEMENTS, FOR INSTANCE, AS WELL AS THE WIDTHS OF ALL SUBSCRIPT PARTITIONS, MUST BE COMPUTABLE AT COMPILE TIME.

VF1 - SEVERITY 2

COMPILE-TIME EVALUATION OF ?? FUNCTION FAILED

ERROR MESSAGES FOR MAJOR CLASSIFICATION X

CLASSIFICATION "X" ERRORS DEAL WITH IMPLEMENTATION DEPENDENT FEATURES
NOTE: OCCURS EITHER BECAUSE THE FUNCTION WAS GIVEN CONSTANTS FOR ARGUMENTS, OR THE ARGUMENTS ARE OUTSIDE ALLOWABLE VALUES

XA1 - SEVERITY 2

ONLY ONE PROGRAM IDENTIFICATION DIRECTIVE IS ALLOWED IN A COMPILATION UNIT.

EACH COMPILATION UNIT MAY RECEIVE, AT MOST, ONE PROGRAM IDENTIFICATION NAME FOR THE PURPOSE OF ACCESS CONTROL.

XA2 - SEVERITY 2

THE PROGRAM IDENTIFICATION DIRECTIVE DOES NOT CONTAIN A VALID IDENTIFICATION. THE DIRECTIVE MUST BE OF THE FORM: D PROGRAM ID=<ID>

THE <ID> FIELD MUST CONSIST OF A 1 TO 8 CHARACTER IDENTIFYING NAME.

XA3 - SEVERITY 2

A PROGRAM IDENTIFICATION DIRECTIVE MUST APPEAR FOLLOWING ANY EXTERNAL TEMPLATES AND PRIOR TO THE BEGINNING OF THE PRIMARY UNIT OF COMPILATION. THE CURRENT DIRECTIVE IS OUT OF PLACE AND WILL NOT BE PROCESSED.

SEE U.G.-SECTION 5.2

XD1 - SEVERITY 2

UNINTELLIGIBLE INFORMATION IN DEVICE DIRECTIVE

THE DEVICE DIRECTIVE MUST TAKE THE FORM OF: "D DEVICE CHANNEL = N <OPTION>" WHERE N SPECIFIES THE CHANNEL INVOLVED (0-9), AND <OPTION> MAY BE PAGED, UNPAGED, OR EMPTY.

XD2 - SEVERITY 2

DUPLICATE DEVICE DIRECTIVE FOR CHANNEL ??

NO MORE THAN ONE DEVICE DIRECTIVE SHOULD BE SPECIFIED FOR EACH CHANNEL USED IN A COMPILATION UNIT.

XD3 - SEVERITY 2

DEVICE DIRECTIVE DOES NOT CONTAIN A VALID CHANNEL INDICATION

THE CHANNEL SPECIFIED MUST BE IN THE RANGE FROM 0-9.

XD4 - SEVERITY 2

CHANNEL NUMBERS MUST BE IN RANGE 0 TO 9

XD5 - SEVERITY 2

DEFINE DIRECTIVE DOES NOT CONTAIN A NAME

EVERY DEFINE DIRECTIVE, USED TO SPECIFY A LOCAL MEMBER OF AN INCLUDE LIBRARY, MUST GIVE THIS MEMBER A NAME. SEE U.G. - SECTION 5.2

XD6 - SEVERITY 2

CLOSE ?? DOES NOT MATCH NAME ON DEFINE DIRECTIVE

THE CLOSE DIRECTIVE MUST USE THE SAME NAME AS THE OPENING DEFINE DIRECTIVE. U.G. - SECTION 5.2

XD7 - SEVERITY 1

DEFINE SEQUENCE IS EMPTY

THE DEFINE DIRECTIVE IS IMMEDIATELY FOLLOWED BY A CLOSE DIRECTIVE.

XD8 - SEVERITY 2

DEFINE DIRECTIVES NOT ALLOWABLE FROM INCLUDED SEQUENCE

DEFINE DIRECTIVES MAY NOT BE NESTED WITHIN INCLUDE DIRECTIVES.

XD9 - SEVERITY 1
A DEFINE DIRECTIVE WITH THE NAME ??
ALREADY EXISTS AND WILL BE REPLACED
NON-EMPTY DEFINE DIRECTIVES MAY NOT HAVE THE SAME NAME.

XI1 - SEVERITY 2
NESTED INCLUDE DIRECTIVES NOT ALLOWED
SEE U.G.-SECTION 5.2. THE NESTED DIRECTIVE IS IGNORED.

XI10 - SEVERITY 1
?? NOT FOUND IN SDF LIBRARY

XI11 - SEVERITY 1
WARNING--UNIT NAME ON INCLUDE CARD DIFFERS FROM THAT IN THE SDF: ??
POSSIBLY TWO COMPILATION UNITS HAVE THE SAME FIRST 6 CHARACTERS, OR THE UNIT NAME WAS MISSPELLED.

XI12 - SEVERITY 2
ILLEGAL TO INCLUDE A TEMPLATE WITHIN A DEFINE SEQUENCE--INCLUDE IGNORED

XI2 - SEVERITY 2
INCLUDE DIRECTIVE DOES NOT CONTAIN A NAME
THE INCLUDE DIRECTIVE MUST EITHER SPECIFY OR IMPLY A NAME FOR THE INCLUDE'D UNIT, USING THE D INCLUDE <NAME> OR THE D INCLUDE TEMPLATE <UNIT NAME> FORMATS, RESPECTIVELY. SEE U.G.-SECTION 5.2. THE DIRECTIVE WILL BE IGNORED.

XI3 - SEVERITY 2
?? NOT IN INCLUDE LIBRARY
INCLUDE DIRECTIVES MUST SPECIFY UNITS TO BE INCLUDE'D WHICH ARE AVAILABLE IN A SYMBOLIC LIBRARY, DEFINED BY THE APPROPRIATE JCL. SEE U.G.-SECTION 5.2

XI4 - SEVERITY 2
'REMOTE' KEYWORD ONLY ALLOWED ON 'INCLUDE TEMPLATE' DIRECTIVES
THE KEYWORD, 'REMOTE' IS ILLEGAL IN THE INCLUDE SDF DIRECTIVE.

XI5 - SEVERITY 2
'REMOTE' KEYWORD SPECIFIED ON NON-COMPOOL TEMPLATE
CODE BLOCKS CANNOT BE REMOTE, ONLY DATA MAY BE.

XI6 - SEVERITY 2
STRUCTURE ?? NOT IN SDF
A STRUCTURE TEMPLATE NAME CONTAINED IN THE VARIABLE LIST FORM OF THE INCLUDE DIRECTIVE WAS NOT FOUND IN THE INCLUDED SDF.

XI7 - SEVERITY 2
SYMBOL ?? NOT IN SDF
A SYMBOL NAME CONTAINED IN THE VARIABLE LIST FORM OF THE INCLUDE SDF DIRECTIVE WAS NOT FOUND IN THE INCLUDED SDF.

XI8 - SEVERITY 2
ILLEGAL TO INCLUDE UNQUALIFIED STRUCTURE TERMINAL ?? FROM AN SDF-INCLUDE
THE APPROPRIATE STRUCTURE TEMPLATE AND STRUCTURE VARIABLE INSTEAD

XI9 - SEVERITY 1

?? WAS FOUND IN SDF LIBRARY, BUT IT WAS CREATED BY A VERSION OF THE COMPILER NOT SUPPORTING SDF INCLUDES.

THE INCLUDED MODULE SHOULD BE RECOMPILED BY A LATER VERSION OF THE COMPILER TO ALLOW ITS SDF TO BE INCLUDED.

XM1 - SEVERITY 2

?? IS A NONEXISTENT %MACRO

THE ONLY LEGAL %MACROS ARE SVC, NAMECOPY, NAMEADD, NAMEBIAS, AND COPY. (NAMEADD AND NAMEBIAS ARE ILLEGAL FOR BFS)

XM10 - SEVERITY 2

ARGUMENT OF %MACRO HAS ILLEGAL MULTIPLE NAME COPYNESS

XM2 - SEVERITY 2

?? HAS AN INCORRECT NUMBER OF ARGUMENTS

THE SVC %MACRO ACCEPTS 1 ARGUMENT; NAMECOPY ACCEPTS 2; COPY TAKES 2 OR 3, NAMEBIAS ACCEPTS 2, AND NAMEADD TAKES 3. SEE U.G.-SECTION 8.7

XM3 - SEVERITY 2

ARGUMENT OF %MACRO MAY NOT BE A LITERAL

XM4 - SEVERITY 2

ARGUMENT OF %MACRO HAS ILLEGAL TYPE

THE ARGUMENTS TO NAME COPY MUST EACH BE STRUCTURES.

XM5 - SEVERITY 2

ARGUMENT OF %MACRO MAY NOT BE A VARIABLE

THE ARGUMENT MUST BE A NAME VARIABLE.

XM7 - SEVERITY 2

ARGUMENT OF %MACRO MAY NOT POSSESS ARRAYNESS

XM8 - SEVERITY 2

ARGUMENT OF %MACRO MAY NOT BE A VARIABLE WITH THE TEMPORARY ATTRIBUTE

XM9 - SEVERITY 2

ARGUMENT OF %MACRO POSSESSES ILLEGAL SUBSCRIPTING

A %MACRO IS SUBJECT TO THE SAME RESTRICTIONS AS SUBSCRIPTION OR WHEN BEING USED AS ASSIGN PARAMETERS FOR PROCEDURE ARGUMENTS. NAMEBIAS ACCEPTS NO SUBSCRIPTS.

XQ101 - SEVERITY 2

ATTEMPTED AN ILLEGAL STACK WALKBACK LOOP FOR VARIABLE??

THIS SITUATION ARISES WHEN ONE PROCEDURE BLOCK IS TRYING TO INDIRECTLY LOCATE A STACK VARIABLE IN AN OUTER BLOCK. INSTEAD, PASS THE VARIABLE TO THE NESTED PROCEDURE.

XQ102 - SEVERITY 2

ATTEMPT TO ASSIGN NAME OF REMOTE DATA ITEM TO A 16 BIT NAME VARIABLE.

XR1 - SEVERITY 2

THE DATA_REMOTE DIRECTIVE MUST BE PLACED BEFORE THE PROGRAM/PROCEDURE STATEMENT. THE DIRECTIVE IS IMPROPERLY PLACED AND WILL NOT BE PROCESSED.

XR2 - SEVERITY 2

THE DATA_REMOTE DIRECTIVE IS ILLEGAL FOR A COMPOOL COMPILATION.

XR3 - SEVERITY 1

THE REMOTE ATTRIBUTE IS IGNORED FOR NON-NAME VARIABLES WHEN THE DATA_REMOTE DIRECTIVE IS IN EFFECT.

XR4 - SEVERITY 2

THE FOLLOWING DATA_REMOTE FCOS RESTRICTION WAS ENCOUNTERED: ??

(REFERENCE: HALS-FC COMPILER SYSTEM SPECIFICATION, USA003089, SECTION 4.1.3.4 FOR MORE INFORMATION)

XR5 - SEVERITY 2

THE REMOTE ATTRIBUTE IS ONLY SUPPORTED FOR NAME VARIABLES OR FORMAL PARAMETERS WHEN SDL OPTION IS ON. TO PLACE DECLARED DATA IN A REMOTE SECTOR, USE DATA_REMOTE OR INCLUDE A COMPOOL REMOTE.

SEE SECTION 8.12 FOR MORE INFORMATION

XS1 - SEVERITY 1

THE FOLLOWING FUNCTION IS NOT IMPLEMENTED IN THE CURRENT LANGUAGE SUBSET: ??

XS2 - SEVERITY 1

THE ABOVE CONSTRUCT IS NOT IMPLEMENTED IN THE CURRENT LANGUAGE SUBSET. (VIOLATION AT PRODUCTION ??)

XS3 - SEVERITY 1

ATTEMPT TO INVOKE UNVERIFIED RUNTIME LIBRARY ROUTINE ??

XU1 - SEVERITY 2

D CARD CONTAINS UNKNOWN DIRECTIVE

THE ONLY LEGAL DIRECTIVES ARE DEVICE, INCLUDE, PROGRAM, VERSION, EJECT, SPACE, DEFINE, AND CLOSE. THE DIRECTIVE NAME MUST BE THE FIRST WORD TO FOLLOW THE D IN COLUMN 1

XV1 - SEVERITY 2

LAST LINE OF TEMPLATE LIBRARY MEMBER ?? IS NOT A VALID VERSION DIRECTIVE

A D VERSION STATEMENT IS EITHER MISSPELLED OR THE TEMPLATE DOES NOT HAVE ONE TO BEGIN WITH

ERROR MESSAGE FOR MAJOR CLASSIFICATION Y

CLASSIFICATION "Y" IS FOR ADVISORY MESSAGES TO THE PROGRAMMER

YA100 - PROGRAMMING ADVISORY

BIT LENGTH MISMATCHES IN AN ASSIGNMENT STATEMENT

SIZE OF RHS OPERATOR IS NOT EQUAL TO SIZE OF THE LHS OPERATOR IN THE ASSIGNMENT STATEMENT

YC100 - PROGRAMMING ADVISORY

BIT LENGTH MISMATCHES IN A COMPARISON

SIZE OF RHS OPERATOR IS NOT EQUAL TO SIZE OF THE LHS OPERATOR IN THE BIT COMPARISON

YD100 - PROGRAMMING ADVISORY

REMOTE ATTRIBUTE IGNORED FOR NON-NAME FORMAL INPUT PARAMETER

SINCE YCONS ARE ALWAYS PASSED INTO NON-NAME INPUT PARAMETERS, THE REMOTE ATTRIBUTE IS MEANINGLESS AND WILL BE IGNORED BY THE COMPILER.

YE100 - PROGRAMMING ADVISORY

BIT LENGTH MISMATCHES IN AN EXPRESSION

SIZE OF OPERATORS ARE NOT EQUAL IN THE BIT EXPRESSION

YF100 - PROGRAMMING ADVISORY

USE OF A %NAMECOPY WHEN A NAME ASSIGNMENT COULD BE USED

A %NAMECOPY WAS USED WHEN BOTH THE SOURCE AND DESTINATION USE THE SAME OR TREE EQUIVELENT STRUCTURE TEMPLATE. A NAME ASSIGNMENT STATEMENT SHOULD BE USED INSTEAD.

YF101 - PROGRAMMING ADVISORY

USE OF A %NAMEADD WHEN A %NAMECOPY COULD BE USED

A %NAMEADD WAS USED WHEN BOTH THE SOURCE AND DESTINATION ARE STRUCTURES AND AN INCREMENT OF 0 WAS USED. A %NAMECOPY SHOULD BE USED IN THIS CASE.

YF102 - PROGRAMMING ADVISORY

USE OF A %NAMEADD WHEN A NAME ASSIGNMENT COULD BE USED

A %NAMEADD WAS USED WHEN AN INCREMENT OF 0 WAS USED AND THE SOURCE AND DESTINATION MATCH IN TYPE AND DIMENSION. A NAME ASSIGNMENT SHOULD BE USED IN THIS CASE.

YF103 - PROGRAMMING ADVISORY

BIT LENGTH MISMATCHES IN A FUNCTION/PROCEDURE CALL OR FUNCTION RETURN

SIZE OF ACTUAL INPUT PARAMETER IN THE FUNCTION/PROCEDURE CALL IS NOT EQUAL TO SIZE OF THE FORMAL PARAMETER, OR SIZE OF RETURN VALUE IS NOT EQUAL TO SIZE OF THE DECLARED VALUE IN A FUNCTION BLOCK.

YF104 - PROGRAMMING ADVISORY

NAME CHARACTER LENGTHS MAY MISMATCH IN THE FUNCTION/PROCEDURE CALL

IT IS NOT POSSIBLE TO CHECK THE LENGTHS OF NAME CHARACTER ARRAYS, SO ALL NAME CHARACTER (ARRAYED OR NON-ARRAYED) ARGUMENTS GENERATE AN ADVISORY MESSAGE SO THE USER CAN VERIFY THE LENGTHS MATCH. IF THE LENGTHS DO NOT MATCH, THEN THE LENGTH OF THE ARGUMENT WILL OVERRIDE THE LENGTH OF THE PARAMETER.

ERROR MESSAGE FOR MAJOR CLASSIFICATION Z

CLASSIFICATION "Z" ERRORS DEAL WITH "TRAP" MESSAGES FROM THE COMPILER

ZC1 - SEVERITY 2

THE COMPILER CANNOT HANDLE THE MULTI-COPIED STRUCTURE REFERENCE BECAUSE OF IMPROPER SUBSCRIPTING OR A NON-CONTIGUOUS REGION SPECIFICATION

ZO1 - SEVERITY 1

A SIN/COSINE OPTIMIZATION FAILURE AT STATEMENT ?? HAS BEEN DETECTED. THE COMPILER GENERATES INCORRECT OBJECT CODE FOR THIS STATEMENT. SIMPLIFY THE EXPRESSION AND RECOMPILE.

ZO2 - SEVERITY 2

THERE IS A PROBLEM IN THE OPTIMIZER FOR THIS STATEMENT WHICH CAUSES THE COMPILER TO ABEND. SEE THE FC USERS MANUAL FOR FURTHER DETAILS.

IF YOU ARE ATTEMPTING TO REFERENCE AND MODIFY A STRUCTURE ELEMENT IN THE SAME STATEMENT, TRY SPLITTING THE STATEMENT INTO TWO PARTS WITH A TEMPORARY VARIABLE AS AN INTERMEDIATE.

ZO3 - SEVERITY 1

LOOP INVARIANT PULLED FROM IF-THEN GROUP AT STATEMENT ?? COULD RESULT IN UNINTENTIONAL GPC ERROR...DR103032.

CODE THAT WAS IN THE CONDITIONAL EXECUTION PART OF AN IF-THEN BLOCK WAS PULLED OUTSIDE THE LOOP AS AN INVARIANT EXPRESSION. POSSIBLE RUNTIME ERRORS MAY OCCUR WHEN THIS CODE IS EXECUTED. TO PREVENT AN EXPRESSION FROM BEING PULLED FROM THE LOOP, REFERENCE A VARIABLE CONTAINED IN THE EXPRESSION IN THE CONDITIONAL PART OF THE IF STATEMENT.

ZO4 - SEVERITY 2

THERE IS A PROBLEM FOR THIS STATEMENT IN THE OPTIMIZER WHICH COULD RESULT IN AN INFINITE LOOP, INCORRECT OBJECT CODE, OR UNEXPECTED ERRORS.

ZP1 - SEVERITY 1

THE COMPILER GENERATES INCORRECT OBJECT CODE AT STATEMENT ?? BECAUSE OF A REGISTER PRESSURE PROBLEM. SIMPLIFY THE EXPRESSION AND RECOMPILE.

ZS1 - SEVERITY 1

THE COMPILER GENERATES INCORRECT OBJECT CODE FOR THE SUBBIT EXPRESSION AT STATEMENT ??.

User Abend Codes During Compilation

The following user abend codes may occur during execution of the HAL/S-FC or HAL/S-360 compiler. The codes marked by a "*" indicate internal compiler failures.

100	Unable to open one of the files: PROGRAM, SYSIN, or SYSPRINT. A missing or misspelled DD card may be the cause.
200*	Unexpected end of file while reading in the XPL program.
300*	Synad error while loading the compiler.
400	Compiler won't fit in the amount of memory available; increase REGION size.
500*	Invalid service code from the XPL program.
600	Printed-page limit exceeded; under control of PAGES= option.
700*	Linked Phases specified different size common areas.
800	Error on output file; possible bad DCB attributes or missing DD card.
900*	Invalid output file used by compiler.
1000	Error on input file; possible bad DCB attributes or missing DD card.
1100*	Linking process overlaid common string area.
1200*	End of file error on input file.
1300*	Impossible to move the common strings up during linking.
1400*	Invalid input file used by the compiler.
1500*	Unknown request by 'MONITOR' function of compiler.
1600*	Unknown DD in 'MONITOR' request.
1700	Directory error on PDS; possibly not enough directory space.
1800	Error on output PDS file; bad DCB attributes or bad DD card.
1900*	Invalid internal member name specified by compiler.
2000+n	Synad error on direct access file n; possible not enough direct access space.
2100*	Attempt to read from an input PDS without issuing the "FIND" MONITOR request first.
2200*	End of file error on direct access file.
2300*	Invalid member to be found.
2400	Error on PDS input file; bad DCB attributes or bad DD card.
2500*	File blocking specification error.
2600*	Bad name for option processor.
3000*	Mon #9/10 error or misaligned #5.
4000*	Compiler forced an abend (and a possible core dump).

Appendix C EXECUTION-TIME ERRORS

The following tables indicate runtime error conditions which may occur during execution of a HAL/S-FC program. The tables list any standard fixes performed by the runtime system.

These errors are detected by the HAL/S-FC library and emitted code. They are classified as group 4 errors within the HAL/S error grouping scheme.

ERROR NUMBER	MESSAGE	EXPLANATION	STANDARD FIXUP
4	Exponentiation of zero to power < = 0	A**B where B < = 0	The result is set to zero
5	SQRT has argument < 0		The result is the square root of the absolute value of the argument
6	EXP function has argument > 174.673		The result is set to the maximum representable value (7.237 x 10**75)
7	LOG function (natural log) has argument < = 0		If the argument was zero then the result is set to the maximum representable negative value (-7.237 x 10**75), else it is set to the natural log of the absolute value of the argument
8	SIN or COS function has |argument| > ~2**18π (823,296)		The result is set to $\frac{\sqrt{2}}{2}$ …
9	SINH or COSH function has argument > 175,366		The result is set to the maximum representable value (7.237 x 10**75)
10	ARCSIN or ARCCOS function has |argument| > 1		ARCSIN (>1) = π/2 ARCSIN (<-1) = -π/2 ARCCOS (>1) = 0 ARCCOS (<-1) = π
11	TAN function has |argument| > ~2**18π (823,549.625) (SP) or ~2**50π (3.537 x 10**15) (DP)		The result is set to one

ERROR NUMBER	MESSAGE	EXPLANATION	STANDARD FIXUP
12	TAN function too close to singularity	The argument is too close to an odd multiple of $\pi/2$	The result is set to the maximum representable value (7.237 x 10**75)
14	No RETURN statement in function	No RETURN statement was encountered prior to reaching the close of the function	Continue
15	SCALAR too large for INTEGER conversion		The result is set to the maximum representable integer value (32767 or -32768)
16	Division by zero in REMAINDER	In REMAINDER (A, B), B=0	The result is set to A
17	Illegal CHARACTER subscript	Character component subscribing out-of-bounds	The out-of-bounds subscript(s) set to first or last character
18	Bad length in LJUST or RJUST	The length is less than the string length	Truncation to the specified length occurs on the left (RJUST) or right (LJUST)
19	MOD domain error	In MOD (A, B), B=0 and A<0	The result is set to zero
20	CHARACTER to SCALAR conversion	The CHARACTER variable has a length of zero or is all blanks	The result is set to zero
22	CHARACTER to INTEGER conversion	The CHARACTER variable has a length of zero or is all blanks	The result is set to zero
24	Negative base in exponentiation	A**B where A<0	The result is |A| **B
25	VECTOR/MATRIX division by zero		The result is the original vector/matrix
27	Argument of INVERSE is a singular matrix		The result is the identity matrix

ERROR NUMBER	MESSAGE	EXPLANATION	STANDARD FIXUP
28	Argument of UNIT function is null vector	Every component of the vector was zero in value	The result is a vector all of whose components are zero (i.e. the input vector)
29	Illegal BIT string		The result is set to zero
30	Illegal SUBBIT subscript	The subscript for SUBBIT exceeded bit length	Subscript takes value of exceeded limit
31	BIT@OCT - Invalid character		The result is set to zero
32	BIT@HEX - Invalid character		The result is set to zero
33	MOD relative magnitude error	In MOD (A, B), $\lvert A/B \rvert$ > ~16**6 (SP) $\lvert A/B \rvert$ > ~16**14 (DP)	The result is set to zero
50	Error in HAL/S source	Execution of a statement with compile time error	Continue
59	ARCCOSH function has argument <1		The result is set to zero
60	ARCTANH function has $\lvert$argument$\rvert \geq 1$		The result is set to zero
62	ARCTAN2 arguments are zero	In ARCTAN2(X,Y), X = Y = 0	The result is set to zero

This page is intentionally left blank.

Appendix D HAL/S-FC RUNTIME LIBRARY NAMES

The following names are members of the HAL/S-FC runtime library. Care should be taken that user-written modules which are not processed by the HAL/S compiler be named in such a way that no conflict occurs with names in this system library.

The members which are actual entry points are shown as being verified or unverified. If a call to an unverified routine is generated during a HAL/S compilation, the severity 1 warning message XS3 will be emitted (see Appendix B for message).

MEMBER NAME	ALIAS OF	VERIFIED
ACOS		YES
ACOSH		NO
ASIN	ACOS	YES
ASINH		NO
ATAN	EATAN2	YES
ATANH		NO
BIN	HIN	NO
BOUT	IOINIT	NO
BTOC		NO
CAS	CASV	YES
CASP	CASPV	NO
CASPV		YES
CASR	CASRV	YES
CASRP	CASRPV	NO
CASRPV		NO
CASRV		NO
CASV		YES
CAT	CATV	NO
CATV		YES
CEIL	#0ROUND	YES
CIN		NO
CINDEX		NO
CINP		NO
CLJSTV		NO
COLUMN	IOINIT	NO
COS	SNCS	YES
COSH	SINH	NO
COUT	COUTP	NO
COUTP		NO
CPAS		YES
CPASP		YES
CPASR		NO
CPASRP		NO

MEMBER NAME	ALIAS OF	VERIFIED
CPR		YES
CPRA		NO
CPRC	CPR	NO
CPSLD	CSLD	NO
CPSLDP	CSLD	NO
CPSST	CSLD	NO
CPSSTP	CSLD	NO
CRJSTV		NO
CSHAPQ		NO
CSLD		NO
CSLDP	CSLD	NO
CSST	CSLD	NO
CSSTP	CSLD	NO
CSTR		NO
CSTRUC		YES
CTOB		NO
CTOD	CTOE	NO
CTOE		NO
CTOH	CTOI	NO
CTOI		NO
CTOK	CTOI	NO
CTOO	CTOX	NO
CTOX		NO
CTRIMV		NO
DACOS		YES
DACOSH		NO
DASIN	DACOS	YES
DASINH		NO
DATAN	DATAN2	YES
DATAN2		YES
DATANH		NO
DCEIL	#0ROUND	YES
DCOS	DSNCS	YES
DCOSH	DSINH	NO
DEXP		YES
DFLOOR	#0ROUND	YES
DIN	HIN	NO
DLOG		YES
DMAX		NO
DMDVAL		YES
DMIN		NO

MEMBER NAME	ALIAS OF	VERIFIED
DMOD		YES
DOUT	IOINIT	NO
DPROD		NO
DPWRD		YES
DPWRH	DPWRI	YES
DPWRI		NO
DROUND	#0ROUND	YES
DSIN	DSNCS	YES
DSINH		NO
DSLD		NO
DSNCS		YES
DSQRT		YES
DSST		NO
DSUM		NO
DTAN		YES
DTANH		NO
DTOC	ETOC	NO
DTOH	ETOH	YES
DTOI	#0ROUND	YES
DTRUNC	#0ROUND	YES
EATAN2		YES
EIN	HIN	NO
EMAX		YES
EMIN		YES
EMOD		YES
EOUT	IOINIT	NO
EPROD		NO
EPWRE		YES
EPWRH	EPWRI	YES
EPWRI		NO
ESUM		NO
ETOC		NO
ETOH		YES
ETOI	#0ROUND	YES
EXP		YES
FLOOR	ROUND	NO
GTBYTE		YES
HIN		NO
HMAX		YES
HMIN		YES
HMOD	IMOD	YES

MEMBER NAME	ALIAS OF	VERIFIED
HOUT	IOINIT	NO
HPROD		NO
HPWRH	IPWRI	NO
HREM	IREM	YES
HSUM		YES
HTOC	ITOC	YES
IIN	HIN	NO
IMAX		NO
IMIN		NO
IMOD		YES
INTRAP	IOINIT	NO
IOBUF	IOINIT	NO
IOCODE	IOINIT	NO
IOINIT		NO
IOUT	IOINIT	NO
IPROD		NO
IPWRH	IPWRI	NO
IPWRI		NO
IREM		YES
ISUM		NO
ITOC		NO
ITOD	#0ITOD	YES
ITOE	#0ITOE	YES
KTOC		NO
LINE	IOINIT	NO
LOG		YES
MM0DNP		YES
MM0SNP		NO
MM11D3		YES
MM11DN		YES
MM11S3		YES
MM11SN		NO
MM12D3		YES
MM12DN		NO
MM12S3		NO
MM12SN		NO
MM13D3		NO
MM13DN		NO
MM13S3		YES
MM13SN		NO
MM14D3		YES

MEMBER NAME	ALIAS OF	VERIFIED
MM14DN		NO
MM14S3		NO
MM14SN		NO
MM15DN		YES
MM15SN		NO
MM17D3		NO
MM17DN	MM17D3	NO
MM17S3		NO
MM17SN	MM17S3	NO
MM1DNP		YES
MM1SNP		NO
MM1TNP		NO
MM1WNP		NO
MM6D3		YES
MM6DN		YES
MM6S3		YES
MM6SN		NO
MMRDNP		NO
MMRSNP		NO
MMWDNP		NO
MMWSNP		NO
MR0DNP		NO
MR0SNP		NO
MR1DNP		NO
MR1SNP		NO
MR1TNP		NO
MR1WNP		NO
MSTR		YES
MV6D3		YES
MV6DN		YES
MV6S3		YES
MV6SN		NO
OTOC	XTOC	NO
OUTER1	IOINIT	NO
PAGE	IOINIT	NO
QSHAPQ		NO
RANDG	RANDOM	NO
RANDOM		NO
ROUND	#0ROUND	YES
SIN	SNCS	YES
SINH		NO

MEMBER NAME	ALIAS OF	VERIFIED
SKIP	IOINIT	NO
SNCS		YES
SQRT		YES
STBYTE		YES
TAB	IOINIT	NO
TAN		YES
TANH		NO
TRUNC	#0ROUND	YES
VM6D3		YES
VM6DN		YES
VM6S3		YES
VM6SN		NO
VO6D3		YES
VO6DN		YES
VO6S3		YES
VO6SN		NO
VR0DN		NO
VR0DNP		NO
VR0SN		NO
VR0SNP		NO
VR1DN		NO
VR1DNP		NO
VR1SN		YES
VR1SNP		NO
VR1TN		NO
VR1TNP		NO
VR1WN		NO
VR1WNP		NO
VV0DN		YES
VV0DNP		YES
VV0SN		YES
VV0SNP		NO
VV10D3		YES
VV10DN	VV10D3	NO
VV10S3		YES
VV10SN	VV10S3	NO
VV1D3		YES
VV1D3P		YES
VV1DN		YES
VV1DNP	VV1D3P	YES
VV1S3		YES

MEMBER NAME	ALIAS OF	VERIFIED
VV1S3P		YES
VV1SN		YES
VV1SNP	VV1S3P	NO
VV1T3		YES
VV1T3P		YES
VV1TN		YES
VV1TNP	VV1T3P	NO
VV1W3		YES
VV1W3P		NO
VV1WN		YES
VV1WNP	VV1W3P	NO
VV2D3		YES
VV2DN		NO
VV2S3		YES
VV2SN		NO
VV3D3		YES
VV3DN		YES
VV3S3		YES
VV3SN		NO
VV4D3		YES
VV4DN		YES
VV4S3		YES
VV4SN		NO
VV5D3		YES
VV5DN		NO
VV5S3		YES
VV5SN		NO
VV6D3		YES
VV6DN		YES
VV6S3		YES
VV6SN		YES
VV7D3		YES
VV7DN		NO
VV7S3		YES
VV7SN		NO
VV8D3		NO
VV8DN	VV8D3	NO
VV8S3		YES
VV8SN	VV8S3	NO
VV9D3	VV10D3	YES
VV9DN	VV10D3	NO

MEMBER NAME	ALIAS OF	VERIFIED
VV9S3		YES
VV9SN	VV10S3	YES
VX6D3		YES
VX6S3		YES
XTOC		NO
#0ETOH		YES
#0ITOD		YES
#0ITOE		YES
#0ROUND		YES
#LACOS	ACOS	N/A
#LACOSH	ACOSH	N/A
#LATANH	ATANH	N/A
#LCASPV	CASPV	N/A
#LCASRPV	CASRPV	N/A
#LCLJSTV	CLJSTV	N/A
#LCPAS	CPAS	N/A
#LCPASR	CPASR	N/A
#LCRJSTV	CRJSTV	N/A
#LCSHAPQ	CSHAPQ	N/A
#LCSLD	CSLD	N/A
#LCTOB	CTOB	N/A
#LCTOE	CTOE	N/A
#LCTOI	CTOI	N/A
#LCTOX	CTOX	N/A
#LDACOS	DACOS	N/A
#LDACOSH	DACOSH	N/A
#LDATAN2	DATAN2	N/A
#LDATANH	DATANH	N/A
#LDEXP	DEXP	N/A
#LDLOG	DLOG	N/A
#LDMOD	DMOD	N/A
#LDPWRD	DPWRD	N/A
#LDPWRI	DPWRI	N/A
#LDSINH	DSINH	N/A
#LDSLD	DSLD	N/A
#LDSNCS	DSNCS	N/A
#LDSQRT	DSQRT	N/A
#LDSST	DSST	N/A
#LDTAN	DTAN	N/A
#LEATAN2	EATAN2	N/A
#LEMOD	EMOD	N/A

MEMBER NAME	ALIAS OF	VERIFIED
#LEPWRE	EPWRE	N/A
#LEPWRI	EPWRI	N/A
#LETOC	ETOC	N/A
#LEXP	EXP	N/A
#LIMOD	IMOD	N/A
#LIOINIT	IOINIT	N/A
#LIPWRI	IPWRI	N/A
#LIREM	IREM	N/A
#LLOG	LOG	N/A
#LMM14D3	MM14D3	N/A
#LMM14DN	MM14DN	N/A
#LMM14S3	MM14S3	N/A
#LMM14SN	MM14SN	N/A
#LRANDOM	RANDOM	N/A
#LROUND	ROUND	N/A
#LSINH	SINH	N/A
#LSNCS	SNCS	N/A
#LSQRT	SQRT	N/A
#LTAN	TAN	N/A
#LVV10D3	VV10D3	N/A
#LVV10S3	VV10S3	N/A
#LVV5D3	VV5D3	N/A
#LVV5DN	VV5DN	N/A
#LVV5S3	VV5S3	N/A
#LVV5SN	VV5SN	N/A

This page is intentionally left blank.

Appendix E CHANGE HISTORY

REV.	RELEASE	DATE	CR/DR Number	Sections Changed
02	19.8	6/10/81		Title page, Table of Contents, pp. iii, iv, Type 1 Options, pp. 5-2, 5-4, Type 2 Options, p. 5-5, Chapter 9: Using the SPILL Capability, pp. 9-1 - 9-11
03	21.0	1/1/85		Title page / Foreword, pp.i-1/i-2, 4-11/4-12, 4-13/4-14, 5-11/5-12, B-1/B-99, C-1/C-2
04	21.1	9/30/85		Title page / Foreword, p. B-96
05	21.3	9/26/86		Title page / Foreword, pp. 8-17, B-53
06	21.4	5/17/87		Title page / Foreword, pp. 5-12 - 5-13
07	21.7	8/12/88		Title page / Foreword, pp. iii, 5-2 - 5-4, 8-14 - 8-16, 8-24 - 8-27, 9-11 - 9-12, C-3
08	23.0	5/15/89		Title page, pp. 8-14, B-17 - B-21, B-30, B-31, B-31.1, B-53, B-96
09	23.1	2/4/91		Title page, p. iii, 4-3, 4-6, 5-12, 8-15, 8-17, 8-23, 8-23.1, 8-24, B-29, B-30, B-31, B-95, Appendix D (all pages)
10	23.2	4/22/91		Title page, p. iii, iv, 4-7, 5-3, 5-4, 5-5, Delete Chapter 9, pp. 9-1 - 9-12
11	24.0	3/30/92		Title page, p. 4-7, 5-3.1, 5-14,8-28, B-5, B-95, B-96, B-96.1
12	7.0	11/10/92		Title page, pp. 5-2, 5-2.1, 5-3, 5-5, 8-28, B-4, B-29, B-70
13	8.0	3/15/93		Title page, pp. 5-3.1, 8-28
14	24.1/25.0	9/3/93		Title page, pp. 5-11, 5-12, (blank) & -13 (blank) (item 9 'Downgrade directive' moved to IR-95-12), pp.8-24, 8-28 and 8-29 (new), B-42, B-95, C-3 & C-95
15	24.2	10/22/93		Title page, Table of Contents, pp. iii, 8-14 - 8-18
16	25.1/9.1	1/11/94		Title page, Chapter 8, pp. 8-29, Appendix B, B-17
17	26.0/10.0	9/2/94		Title page, Chapter 5 p.5-2, Appendix B (replace entire appendix)
18	27.0/11.0	6/22/96		Total Reprint
19	27.1/11.1	7/1/96		Pages B-22 - B-38
20	28.0/12.0	08/22/97		Total Reprint to Bring to HAL/S Documentation Standards and HTML Compatibility
			CR12157	4.4 - Deleted
			CR12432A	8.7 - p. 8-10 8.10 - p. 8-19 8.10.3 - p. 8-20 App. B - p. B-13

REV.	RELEASE	DATE	CR/DR Number	Sections Changed
			CR12712	App. B - pp. B-20, B-25, B-29, B-36, B-51, B-52, B-56 App. E - Deleted
			CR12713	4.1 - pp. 4-2, 4-3, 4-4 4.2 - p. 4-9 5.1 - pp. 5-2, 5-3
			CR12714C	App. B - p. B-9
			DR107716	4.2 - p. 4-8
			DR109042	8.11 - pp. 8-27, 8-28
			DR109044	App. B - pp. B-20, B-21, B-28
			DR109046	8.10.3 - p. 8-20 App. B - p. B-35
			DR109047	App. B - p. B-28
			DR109048	App. B - p. B-20
			Clean-up	3.2 - p. 3-2 4.1 - p. 4-2 5.1 - pp. 5-4, 5-6 6.1.2 - p. 6-2 7.4 - p. 7-22 8.9 - p. 8-15 App. A - pp. A-2, A-3 App. C.1 - p. C-6
21		04/30/98		Title page, Table of Contents
				2.1 - p. 2-3 2.3 - p. 2-5 2.4 - p. 2-7, 2-15 3.1 - p. 3-1, 3-2, 3-6 4.2 - p. 4-8 5.1 - p. 5-12 8.0 - p. 8-1 8.9 - p. 8-18 8.11 - p. 8-23, 8-24, 8-26 App. A -p. A-2 App. B - p. B-1 App. C - p. C-1, C-2

Index

Symbols

%COPY....8-10
user note....8-11, 8-13, 8-29
%macro....8-9
%NAMEADD....8-13 to 8-14
user note....8-14
%NAMECOPY....8-9 to 8-10
%SVC....8-9
¢....8-20

A

abend codes....B-74
Access....8-5
ADDRS....5-2
arrays
size limit....8-1

B

BIT
input format....6-2
output format....6-2
overpunch....4-1
size limit....8-1
BITLENGTH....8-8
BLKSIZE....6-4
Block Summary....4-4 to 4-5
BLOCKSUM....5-5
Built-In Function Cross Reference Table....4-7
BUILTINS....8-8

C

CARDTYPE....5-5
catalogued procedures
creating and using....2-16
processing HAL/S programs....2-21
CHARACTER
conversion....8-4
input format....6-2
output format....6-3
overpunch....4-1
size limit....8-1, 8-3
user note....8-34
character set....8-20
CLOCKTIME....8-4
CLOSE....5-8, 5-11
comments....2-6
size limit....8-1
Compilation Layout Summary....4-5

compiler directives 5-6
compiler listings 4-1
Block Summary ..... 4-4 to 4-5
Built-In Function Cross Reference Table 4-7
Compilation Layout Summary 4-5
current scope 4-2
Downgrade Summary 4-11
error messages 4-9
overpunches ..... 4-1 to 4-2
Relocation Dictionary (RLD) 4-8
REPLACE macro 4-2
Symbol and Cross Reference Table 4-5
compiler options 5-1
compiler phases 2-1
COMPOOL 3-1
compiling 3-3
INCLUDE 3-1, 5-7
SDF 3-9
templates 3-8
COMPUNIT 5-5
COMSUB 3-1
conversions
CHARACTER 8-4, 8-34
INTEGER ..... 8-3 to 8-4
SCALAR ..... 8-3 to 8-4
COPY 8-10
user note 8-11, 8-13, 8-29
CSECT
naming conventions 8-18, 8-20
current scope 4-2

D

data size restrictions ..... 8-1 to 8-2
DATA_REMOTE ..... 5-11 to 5-12
NAME 5-12
DATE 8-4
DD Card 2-11
DD names
ERROR 2-24
LISTING2 2-23
OUTPUT3 2-23
OUTPUT4 2-23
OUTPUT5 2-24
OUTPUT6 2-24
PROGRAM 2-23
STEPLIB 2-23
SYSIN 2-23
SYSLIB 2-24

SYSLIN....2-24
SYSLMOD....2-24
SYSPRINT....2-23 to 2-24
SYSUT1....2-24
DECK....5-2
DEFINE....5-8, 5-11
DEVICE....5-7
directives, compiler....5-6
DISP....2-12
DO CASE
level limit....8-2
user note....4-3
DO-END
level limit....8-2
DOWNGRADE....5-11
Downgrade Summary....4-11
DSR....5-5
DUMP....5-2

E

EJECT....5-11
ERROR....2-24
error messages....4-9
compile-time....B-1
user note....8-32
user-defined limits....8-2
escape character
REPLACE macro....4-3
special characters....8-20
EXEC Card....2-8
exponents
level limit....8-2
external names
number limit....8-2

F

FLOOR
user note....8-28
formats
input....6-1
output....6-2 to 6-3
source program....2-6
FUNCTION
argument number limit....8-3
compiling....3-3
INCLUDE....3-1, 5-7
level limit....8-2
SDF....3-9

templates 3-8

H

HALFCLG A-1
HALMAT 5-2
HIGHOPT 5-2

I

INCLUDE 3-1, 5-7
NOLIST 5-9
NOSDF 5-9
REMOTE 5-9
INITIAL
user note 8-25
input 2-13
input formats 6-1
BIT 6-2
CHARACTER 6-2
INTEGER 6-1
SCALAR 6-1
INTEGER
conversion ..... 8-3 to 8-4
input format 6-1
output format 6-2
range limit 8-3

J

JCL 2-7
catalogued procedures 2-16
DD Card 2-11
DISP 2-12
input 2-13
JOBLIB ..... 2-11 to 2-12
output 2-14
STEPLIB ..... 2-11 to 2-12
EXEC Card 2-8
JOB Card ..... 2-7 to 2-8
JOB Card ..... 2-7 to 2-8
JOBLIB ..... 2-11 to 2-12

L

LABELSIZE 5-5, 8-2
LFXI 5-2
library routines D-1
limits
data size ..... 8-1 to 8-2
DO CASE levels 8-2
DO-END levels 8-2

external names....8-2
FUNCTION levels....8-2
INTEGER range....8-3
literals....8-2
ON ERROR....8-3
program organization....8-2 to 8-3
REPLACE macro....8-2
SCALAR range....8-3
subroutine arguments....8-3
subscript and exponent levels....8-2
user-defined errors....8-2
user-defined symbols....8-2
LINECT....5-5
linkediting....2-3 to 2-4
LIST....4-7, 5-3
LISTING2....2-23, 4-7, 5-3
literals
number limit....8-2
LITSTRING....5-5, 8-2
LRECL....6-3
LSTALL....5-3

M

macro....8-9
MACROSIZE....5-6
MATRIX
overpunch....4-1
size limit....8-2
MFID....5-6
MICROCODE....5-3

N

NAME
DATA_REMOTE....5-12
REMOTE....8-23
user note....8-30
NAMEADD....8-13 to 8-14
user note....8-14
NAMECOPY....8-9 to 8-10
NOTABLES....4-8

O

OLDTPL....5-6
ON ERROR
number limit....8-3
options, compiler....5-1
output....2-14
output formats....6-2 to 6-3

BIT....6-2
CHARACTER....6-3
INTEGER....6-2
SCALAR....6-2
output listings....4-1
OUTPUT3....2-23
OUTPUT4....2-23
OUTPUT5....2-24
OUTPUT6....2-24
overpunch....4-1 to 4-2
BIT....4-1
CHARACTER....4-1
MATRIX....4-1
STRUCTURE....4-1
VECTOR....4-1

P

PAGED....6-1
BIT....6-2
carriage control....6-4
CHARACTER....6-3
DEVICE....5-7
PAGES....5-6
PARSE....5-3
precision
CHARACTER conversion....8-4, 8-34
INTEGER conversion....8-3 to 8-4
SCALAR conversion....8-3 to 8-4
user note....8-28
PROCEDURE
argument number limit....8-3
compiling....3-3
INCLUDE....3-1, 5-7
SDF....3-9
templates....3-8
PRODUCTIONS....8-8
PROGRAM....2-23, 5-9
program organization limits....8-2 to 8-3

R

READ....6-1, 6-3
READALL....6-1, 6-3
RECFM....6-3
REGOPT....5-3
Relocation Dictionary (RLD)....4-8
REMOTE....8-22
NAME....8-23
REPLACE macro....4-2

size and nesting limits 8-2
routines, library D-1
RUNTIME 8-4
runtime characteristics 8-3
runtime errors 8-4
runtime library routines D-1

S

SCAL 5-3
SCALAR
conversion ..... 8-3 to 8-4
input format 6-1
output format 6-2
range limit 8-3
SDL 5-4
Simulation Data File (SDF) 3-9
source program format 2-6
SPACE 5-11
SREF 5-4
SRN 5-4
STEPLIB ..... 2-11 to 2-12, 2-23
STRUCTURE
multi-copied size limit 8-2
overpunch 4-1
user note ..... 8-24 to 8-25, 8-27, 8-34
SUBBIT
user note 8-24
subscripts
level limit 8-2
SVC 8-9
Symbol and Cross Reference Table 4-5
SYMBOLS 5-6, 8-2
symbols
user-defined limits 8-2
SYSIN 2-23
SYSLIB 2-24
SYSLIN 2-24
SYSLMOD 2-24
SYSPRINT ..... 2-23 to 2-24
SYSUT1 2-24

T

TABDMP 5-4
TABLES 5-4
TABLST 4-8, 5-4
TEMPLATE 5-4
templates 3-8
user note 8-26

TEMPORARY
user note....8-29
TITLE....5-6

U
UNPAGED....6-1
BIT....6-3
carriage control....6-4
CHARACTER....6-3
DEVICE....5-7
unverified library routines....D-1
UPDATE PRIORITY
user note....8-27
user notes
%COPY....8-11, 8-13, 8-29
%NAMEADD....8-14
CHARACTER....8-34
DO CASE....4-3
error messages....8-32
FLOOR....8-28
INITIAL....8-25
NAME....8-30
precision....8-28
STRUCTURE....8-24 to 8-25, 8-27, 8-34
SUBBIT....8-24
templates....8-26
TEMPORARY....8-29
UPDATE PRIORITY....8-27

V
VARSYM....5-4
VECTOR
overpunch....4-1
size limit....8-2
verified library routines....D-1
VERSION....5-9

W
WRITE....6-2 to 6-3

X
XREFSIZE....5-6, 8-2

Z
ZCON....5-4

This is the Last Page of this Document

NASA-JSC

*BV	N. Moses
MS4	D. Stamper
EV111	EV Library (D. Wall)

USA-Houston

*USH-121G	SFOC Technical Library
USH-634G	Abel Puente
USH-64A6X	L.W. Wingo
USH-633L	Anita Senviel
USH-633L	Benjamin L. Peterson
USH-633L	Cory L. Driskill
USH-633L	Judy M. Hardin
USH-633L	Mark E. Lading
USH-633L	Quinn L. Larson
USH-633L	James T. Tidwell
USH-633L	Vicente Aguilar
USH-633L	Betty A. Pages
USH-633L	Jeremy C. Battan
USH-633L	George H. Ashworth
USH-634L	Mark Caronna
USH-634L	Burk J. Royer
*USH-635L	Joy C. King
USH-635L	Ling J. Kuo
USH-635L	Trang K. Nguyen
USH-635L	Billy L. Pate
USH-635L	Karen H. Pham
*USH-635L	Dan A. Strauss
USH-635L	Pete Koester
USH-632L	Renne Siewers
*USH-635L	Barbara Whitfield (2)

Boeing

HS1-40	B. Frere

blake.a.frere@boeing.com

* Denotes hard copy

Submit NASA distribution changes, including initiator's name and phone number, to JSC Data Management/BV or call 281-244-8506. Submit USA distribution changes to USA Data Management/USH-121E or via e-mail to *usadm@usa-spaceops.com*. Most documents are available electronically via USA Intranet Web (*usa1.unitedspacealliance.com*), Space Flight Operations Contract (SFOC), SFOC Data and Deliverables.

Indicates hardcopy