JAVS TECHNICAL REPORT
Reference Manual

General Research Corporation

Approved for public release; distribution unlimited.
Some of the pages of this report are not of the highest printing quality but because of economical consideration, it was determined in the best interest of the government that they be used in this publication.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and approved for publication.

APPROVED:  
FRANK S. LA MONICA  
Project Engineer

APPROVED:  
ROBERT D. KRUTZ  
Chief, Information Sciences Division

FOR THE COMMANDER:  
JOHN P. HUSS  
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.
The J0VIAL Automated Verification System (JAVS) is a tool for analyzing source programs written in the J3 dialect of the J0VIAL language. From the user's viewpoint, JAVS consists of a sequence of processing steps which (1) analyze his J0VIAL source text, (2) guide him in preparing test cases for his code, and (3) analyze the results of tests executed by his code.

The purpose of this document is to describe in detail JAVS processing and each of the JAVS commands. The organization of the Reference Manual follows a
top-down approach. Starting with a system level description in the first section, each subsequent section describes a subset of the total system in detail.
LIST OF JAVS REPORTS

- JAVS Technical Report: Vol. 1, User's Guide. This report is an introduction to using JAVS in the testing process. Its primary purpose is to acquaint the user with the innate potential of JAVS to aid in the program testing process so that an efficient approach to program verification can be undertaken. Only the basic principles by which JAVS provides this assistance are discussed. These give the user a level of understanding necessary to see the utility of the system. The material on JAVS processing in the report is presented in the order normally followed by the beginning JAVS user. Adequate testing can be achieved using JAVS macro commands and the job streams presented in this guide. The Appendices include a summary of all JAVS commands and a description of JAVS operation at RADC with both sample command sets and sample job control statements.

- JAVS Technical Report: Vol. 2, Reference Manual. This report describes in detail JAVS processing and each of the JAVS commands. The Reference Manual is intended to be used along with the User's Guide which contains the machine-dependent information such as job control cards and file allocation. Throughout the Reference Manual, modules from a sample JOVIAL program are used in the examples. Each JAVS command is explained in detail, and a sample of each report produced by JAVS is included with the appropriate command. The report is organized into two major parts: one describing the JAVS system and the other containing the description of each JAVS command in alphabetical order. The Appendices include a complete listing of all error messages directly produced by JAVS processing.

- JAVS Technical Report: Vol. 3, Methodology Report. This report describes the methodology which underlies and is supported by JAVS. The methodology is tailored to be largely independent of implementation and language. The discussion in the text is intended to be intuitive and demonstrative. Some of the methodology is based upon the experience of using JAVS to test a large information management system. A long-term growth path for automated verification systems that supports the methodology is described.

- JAVS Computer Program Documentation: Vol. 1, System Design and Implementation. This report contains a description of JAVS software design, the organization and contents of the JAVS data base, and a description of the software for each JAVS component: its function, each of the modules in the component, and the global data structures used by the component. The report is intended primarily as an informal reference for use in JAVS software maintenance as a companion to the Software Analysis reports described below. Included in the appendices are the templates for probe code inserted by instrumentation processing for both structural and directive instrumentation and an alphabetical list of all modules in the system (including system routines) with the formal parameters and data type of each parameter.

- JAVS Computer Program Documentation: Vol. 2, Software Analysis. This volume is a collection of computer output produced by JAVS standard processing steps. The source for each component of the JAVS software has been analyzed.
to produce enhanced source listings of JAVS with indentation and control structure identification, inter-module dependence, all module invocations with formal and actual parameters, module control structure, a cross reference of symbol usage, tree report for each leading module, and report showing size of each component. It is intended to be used with the System Design and Implementation Manual for JAVS software maintenance. The Software Analysis reports, on file at RADC, are an excellent example of the use of JAVS for computer software documentation.

- JAVS Preprocessor for JOVIAL. This report, prepared for GRC by its subcontractor, System Development Corporation (SDC), describes the software for the JAVS-2 component: its origin as the GENI part of the SAM-D ED Compiler, the modifications made in GENI to adapt the code for JAVS-2, the JAVS-2 code modules, and the data structures. It contains excerpts of other SDC reports on the SAM-D ED JOVIAL Compiler System. The report reflects the status of the software for JAVS-2 as delivered by SDC to GRC in September 1974. The description of JAVS-2 software contained in the System Design and Integration report reflects the status of JAVS-2 as delivered to RADC by GRC in September 1975 and thereby supercedes the SDC report.

- JAVS Final Report. The final report for the project describes the implementation and application of a methodology for systematically and comprehensively testing computing software. The methodology utilizes the structure of the software undergoing test as the basis for analysis by an automated verification system (JVS). The report also evaluates JAVS as a tool for software development and testing.
# CONTENTS

## LIST OF JAVS REPORTS

### 1 INTRODUCTION
1.1 Brief Description of JAVS
1.2 Command Structure
1.3 Example Program
1.4 Text Identification
1.5 JAVS Computation Directives

### 2 JAVS PROCESSING STEPS
2.1 Source Text Processing
2.2 Structural Analysis
2.3 Module Instrumentation
2.4 Module Testing Assistance and Segment Analysis
2.5 Retesting Guidance and Analysis
2.6 Test Execution
2.7 Test Effectiveness Measurement
2.8 Processing Using JAVS Overlay

### 3 JAVS CONSTRAINTS
3.1 Universal Constraints
3.2 JAVS Basic Constraints
3.3 JAVS Structural Constraints
3.4 JAVS Instrument Constraints
3.5 JAVS Assist Constraints
3.6 Test Execution Constraints
3.7 JAVS Dependence Constraints
3.8 JAVS Analyzer Constraints

### 4 JAVS COMMANDS
4.1 Library Commands
4.2 Startup Command
<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3 Module Selection Commands</td>
<td>4-8</td>
</tr>
<tr>
<td>4.4 Process Option Commands</td>
<td>4-10</td>
</tr>
<tr>
<td>4.5 Process Execution Commands</td>
<td>4-13</td>
</tr>
<tr>
<td>4.6 Standard Print and Punch Commands</td>
<td>4-15</td>
</tr>
<tr>
<td>4.7 Run Termination Command</td>
<td>4-16</td>
</tr>
<tr>
<td>5 JAVS COMMANDS (Alphabetical Descriptions)</td>
<td>5-1</td>
</tr>
<tr>
<td>APPENDIX A JOVIAL KEYWORDS RECOGNIZED BY JAVS</td>
<td>A-1</td>
</tr>
<tr>
<td>APPENDIX B ERROR MESSAGES</td>
<td>B-1</td>
</tr>
<tr>
<td>APPENDIX C STATEMENT TYPE ABBREVIATIONS</td>
<td>C-1</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>R-1</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

JAVS is a system of compatible tools intended to be applied during program testing to aid in the recognition of untested program paths, to assist in the development of additional testcases appropriate to improvement of testing coverage, and to automatically document the computer program. Program validation is provided by analysis of program structures, instrumentation of the system through insertion of appropriate software probes to measure testing coverage, and comprehensive reports which pinpoint paths in the program structure that remain to be exercised. In addition, guidance is provided for the generation of testcases that will assure coverage of the untested portions. JAVS can be thought of as a partner during the testing process, supplying a wide variety of automated aids to the user in comprehensive testing activities.

This Reference Manual is intended to be used along with the JAVS User's Guide and describes in detail JAVS processing and each of the available JAVS commands.

The machine-dependent features are described in the JAVS User's Guide which contains a description of the overlay version of JAVS, the JAVS macro commands, examples of job setups for RADC, and file and processing time information.

The format for this Reference Manual is as follows:

Section 1: JAVS system overview and organization, library and file description, data management, command structure, example test program, START-TERM source text identification and JAVS computation directives. The last two items deal with additions the user makes to his source code before JAVS testing begins.

Section 2: Description and diagram of each processing step.

Section 3: JAVS constraints

Section 4: System level description of JAVS commands

Section 5: Description of each JAVS command
1.1 BRIEF DESCRIPTION OF JAVS

1.1.1 Processing Steps

In the standard version, JAVS is organized as a series of independent processing steps, each devoted to a unique task, that communicate through a common data base to provide thorough software verification assistance. The overlay version of JAVS maintains the same processing tasks, but it allows the user to perform multiple processing steps in a single module selection or in a single activity. The various steps and their basic functions are listed below:

BASIC, Source Text Analysis: Source text input, lexical analysis, and initial source library creation

STRUCTURAL, Structural Analysis: Structural analysis and execution path identification; library update with structure and path information

INSTRUMENT, Module Instrumentation: Program instrumentation for path coverage analysis and program performance directed by user; library update with probe test instrumentation

ASSIST, Module Testing Assistance and Segment Analysis: Testing assistance for improved program coverage

DEPENDENCE, Retesting Guidance and Analysis: Retesting requirements analysis for changed modules

Test Execution: Execution of instrumented code and analysis of directed program performance

ANALYZER, Test Effectiveness Measurement: Detailed analysis of program path coverage; execution traces and summary statistics

These steps need not be performed in the above order. Other orders may be preferable at times.

An overview of how JAVS is used in the testing process is shown in Fig. 1.1.
JOVIAL source code is input for processing and analysis. A special form of comment (optional) inserted by the user directs JAVS processing for program performance analysis.

JAVS analyzes the code and generates a directed graph of the control structure. The possible flows through the program are determined. All pertinent data is stored in a data base for later use. Additional or changed source code causes an existing data base to be updated.

JAVS automatically inserts software probes into the source code to intercept and record program flow during execution. A second type of instrumentation is used to record statistics on program performance according to directives inserted by the user in the source.

JAVS provides a variety of services which assist in the establishing of a continuing testing strategy both at the detailed level within a module and at the module interdependence level. Test cases are constructed by the user. Reports can be used as program documentation.

Program execution provides normal computational results as well as outputs from the instrumentation. Structural flow output is recorded for later post-test analysis by JAVS. User-directed program performance output is interspersed with normal program output.

JAVS includes detailed post-test analysis facilities which provide measures of testing thoroughness, both individually and cumulatively for a set of test cases.

The results are examined by the user to determine if test goals have been met and testing is completed.

Figure 1.1. An Overview of the JAVS System
1.1.2 Library Definition and Contents

JAVS utilizes a random-access file which includes the original source text, as well as subsidiary tables that contain structural information, symbols and their classification, and other module-descriptive information. The initial random library is created during the first processing step (BASIC). The second step (STRUCTURAL) uses the initial library data to create an updated library containing program structure data as well as the source text. INSTRUMENTATION adds instrumented text to the library. The library and instrumented file are then used as input for other analyses. The major sections of a completed library are listed below:

1. Analyzed source text (BASIC)
2. Module descriptions (BASIC)
3. Statement descriptions (BASIC)
4. Symbol descriptions (BASIC)
5. Program structure information (STRUCTURAL)
6. Probe text (INSTRUMENTATION)

1.1.3 File Identification

JAVS makes use of various files during execution of the standard and overlay processing steps. The files used in JAVS processing are shown in Table 1.1.

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Function</th>
<th>Mode</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LIBOLD</td>
<td>Old library</td>
<td>Binary</td>
<td>Random</td>
<td>Read/write</td>
</tr>
<tr>
<td>2</td>
<td>LIBNEW</td>
<td>New library</td>
<td>Binary</td>
<td>Random</td>
<td>Read/write</td>
</tr>
<tr>
<td>3</td>
<td>LIBESP</td>
<td>(Temporary) workspace</td>
<td>Binary</td>
<td>Random</td>
<td>Read/write</td>
</tr>
<tr>
<td>4</td>
<td>COMAUX</td>
<td>(Temporary) command iteration</td>
<td>BCD</td>
<td>Serial</td>
<td>Read/write</td>
</tr>
<tr>
<td>5</td>
<td>(COMMAC)</td>
<td>JAVS command input</td>
<td>BCD</td>
<td>Serial</td>
<td>Read only</td>
</tr>
<tr>
<td></td>
<td>(COMMAC*)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>LOUT</td>
<td>Standard printer output</td>
<td>BCD</td>
<td>Serial</td>
<td>Write only</td>
</tr>
<tr>
<td>7</td>
<td>LPUNCH</td>
<td>Instrumented source text</td>
<td>BCD</td>
<td>Serial</td>
<td>Write only</td>
</tr>
<tr>
<td>8</td>
<td>AUDIT</td>
<td>Test execution probe data</td>
<td>BCD</td>
<td>Serial</td>
<td>Read/write</td>
</tr>
<tr>
<td>9</td>
<td>READER</td>
<td>Source text input</td>
<td>BCD</td>
<td>Serial</td>
<td>Read only</td>
</tr>
<tr>
<td>10</td>
<td>(COMMAC)</td>
<td>(Temporary) JAVS command workspace</td>
<td>BCD</td>
<td>Serial</td>
<td>Read/write</td>
</tr>
</tbody>
</table>

* Used only in JAVS overlay version.
1.1.4 System Execution Control

JAVS utilizes a command language to provide control over many of the processing steps. The command language provides the means to select various processing options within the selected step and, in addition, allows the user to repeatedly execute certain sequences of commands for sets of program modules contained within the library file. Certain commands are available for use in all steps and are therefore termed "universal commands." In the standard version of JAVS, some commands are unique to each processing step and are recognized only when employed within that step. The overlay version of JAVS allows commands from any processing step to be utilized in a single run.

1.1.5 Data Management

Included within the standard processing steps is a data management component which provides the following capabilities:

1. Library merging
2. Processing initialization commands
3. Module selection
4. Standard printouts of library contents
1.2 COMMAND STRUCTURE

A JAVS run is defined by a set of commands. These commands—one to a card—are freeform; blanks are ignored. The card scan ends with a period (.) or with the end of the card. Each command consists of a sequence of terms separated by a comma or an equals sign. A term with an initial digit is assumed to be an integer. If a command must be continued on more than one card, a comma must appear as the last non-blank character of each card preceding a continuation card. Up to three continuation cards may be used.

A JAVS command set always begins with some library commands ending with the command START*. Next comes a set of action and information commands describing what processing is to be performed, listing which modules are to be used, and which options are to be included. A JAVS run terminates on the END command, which provides for correctly closing any files and writing a wrapup summary.

The commands shown in Fig. 1.2 for a hypothetical JAVS run were designed to help show the structure and use of JAVS commands.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLD LIBRARY = OLD1.</td>
<td>Library command</td>
</tr>
<tr>
<td>CREATE LIBRARY = NEW2.</td>
<td>Library command</td>
</tr>
<tr>
<td>START.</td>
<td>Startup command</td>
</tr>
<tr>
<td>FOR LIBRARY.</td>
<td>Module selection command</td>
</tr>
<tr>
<td>STRUCTURAL.</td>
<td>Execution command</td>
</tr>
<tr>
<td>END FOR.</td>
<td>Module selection command</td>
</tr>
<tr>
<td>MODULE = EXMPL1.</td>
<td>Module selection command</td>
</tr>
<tr>
<td>PRINT, MODULE.</td>
<td>Universal print command</td>
</tr>
<tr>
<td>END.</td>
<td>Run termination command</td>
</tr>
</tbody>
</table>

Figure 1.2. Hypothetical Command Sequence

---

*The JAVS macro commands contain the START and additional commands. The macro commands require the JAVS overlay version and are described in the JAVS User's Guide, along with other system dependent information.

1-6
1.3 EXAMPLE PROGRAM

Throughout this Reference Manual, modules from a sample JOVIAL program are used in the examples. Figure 1.3 shows a complete listing of the sample program with its five modules: COMPOOL EXCOMPL, PROGRAM EXPROGM, PROC EXAMPL1, CLOSE EXAMPL2 and CLOSE EXAMPL3. These modules are so simple that they do not in any way demonstrate the power of JAYS; their purpose is solely to provide simple output examples for each of the commands. The output from the execution of the program is shown in Fig. 1.4. This output is generated from the MONITOR statements which appear in the main program EXPROGM.

DEPENDENCE processing deals primarily with inter-module dependencies. A program with a large number of modules is used as an example to illustrate the power of inter-module analysis performed by JAYS. STRUCTURAL analysis (Sec. 2.2) is illustrated using an example procedure from the JAYS User's Guide. This small module is used to demonstrate the identification of DD-paths.

---

Figure 1.3. Example Program Used in the Reference Manual

(Continued on next page)
BEGIN
BYTE $(3,85H)$ [IN] = BYTE $(3,85H)$ [CARD]\
BYTE $(3,85H)$ [ITER1] = BYTE $(3,85H)$ [CARD]\
BYTE $(3,85H)$ [ITER2] = BYTE $(3,85H)$ [CARD]\
EXEMPLARY [ITER1, ITER2]\
CLOSE EXEMPLAR

** MAIN CLOSE **
BEGIN
ITER1 = 1$
ITER2 = 1$
END #EXEMPLAR$
IF ITEM03 100$
GOTO EXEMPLAR$
GOTO #IFS$
STOP$
PROC EXEMPLAR [LIMIT1,LIMIT2]$
BEGIN
ITEM LIMIT1 $INTG$
ITEM LIMIT2 $INTG$
ARRAY FILL 100, $INTG$
ITEM RESULT $INTG$
ITEM INDEX $INTG$
##TRACE RESULT##
IF LIMIT1 30 100$
LIMIT1 = 99$
RESULT = 4$
FOR I = LIMIT1,LIMIT1$
BEGIN
##EXPECT RESULT##$
FILL [I-1,1] = $IS$
RESULT = (RESULT+I)/1$
##ASSERT RESULT OR 10##
FOR J = LIMIT1,LIMIT1$
BEGIN
CLOSE EXEMPLAR
## PROC CLOSE ##
BEGIN
RESULT = 2$I$
END #CLOSE$
IF ITEM03 LIMIT1$
INDEX$ = $IS$
ORIF$ IS$
INDEX$ = 4$
END #IFITEM$
SWITCH PICK = [LABEL1, LABEL1, LABEL1, LABEL1, LABEL1, LABEL1, LABEL2]$
GOTO PICK $(INDEXS-1)$
LABEL2,
LABEL1.$
IF ITEM RESULT $IS$
MESSAG$ = $IS$
ORIF RESULT EQ $IS$
MESSAG$ = $IS$
ORIF$ IS$
MESSAG$ = $IS$
END #IFITEM$
OUTPUT PRINT MESSAGE$
END #IFS$
END #IFS$
##OFFTARGET RESULT##
END #EXEMPLAR$
TERMS$

Figure 1.3. (Continued)
JAVS TEST CASE

*** MONITORED MOLLERITH DATA ID = TWO
*** MONITORED MOLLERITH DATA ITERIA = 300
*** MONITORED MOLLERITH DATA ITER2A = 199

RESULT GT 4

*** MONITORED MOLLERITH DATA ID = ONE
*** MONITORED MOLLERITH DATA ITERIA = 45
*** MONITORED MOLLERITH DATA ITER2A = 14

RESULT GT 4

Figure 1.4. Output from Execution of Sample Program
1.4 TEXT IDENTIFICATION

JAVS automatically separates a START-TERM sequence of JOVIAL source which contains executable JOVIAL statements (i.e., not a COMPOOL) into modules of invokable code (e.g., a PROC, a CLOSE). After the source code has been instrumented, it is reassembled into the START-TERM sequence in preparation for compilation and Test Execution. If more than one START-TERM sequence is contained on the library, it is necessary to identify each sequence separately with a unique name. JAVS has provision for naming a START-TERM sequence and distinguishing between types of text through the use of a special form of comment called a JAVTEXT directive:

```
".JAVTEXT<name><type><(related-text-list)>)<description>
```

where

- `<name>` is the name given to the START-TERM sequence (name must be 8 or less characters, in which the first 6 characters must be unique among the JAVTEXT names)
- `<type>` is COMPUTE for an executable START-TERM text (e.g., a program) or PRESET for nonexecutable text (e.g., a COMPOOL)
- `<related-text-list>` is an optional list of related text names separated by commas (e.g., the name of a COMPOOL text used during compilation with a program text)
- `<description>` is optional descriptive information

The JAVTEXT directive should appear immediately before the START for a main program or for a PROC subprogram (external PROC). In the case of CLOSE subprogram (external CLOSE), the JAVTEXT name must be different than the CLOSE name, and the JAVTEXT directive should appear before the CLOSE statement.

In the example program, the JAVTEXT directive

```
".JAVTEXT EXCOMPL PRESET"
```

is used to inform JAVS that a COMPOOL is being processed. The JAVTEXT directive

```
".JAVTEXT EXPROGM COMPUTE (EXCOMPL)"
```

is used to inform JAVS that an executable START-TERM sequence which references a COMPOOL (EXCOMPL) is being processed. This directive is required, even if the COMPOOL is not to be processed, if COMPOOL defined names are referenced in the program text. Only one COMPOOL may be put in the database library for each execution of the BASIC step, when processing both COMPOOL and executable text. If a COMPOOL is to be processed by JAVS, it should be analyzed only by the BASIC step. The COMPOOL will then be available on the library for documentation printing.
1.5 JAVS COMPUTATION DIRECTIVES

JAVS contains facilities for analyzing a special form of JOVIAL comment, called the JAVS directive. There are two types of JAVS directives: computation directives and text identification directives (Sec. 1.4). The computation directives, described in this section, are intended for use in monitoring computations performed by the program.

The directives are inserted by the user in the JOVIAL source before it is processed by BASIC. They are stored as separate statements in the statement block on the library created by BASIC. Later INSTRUMENT analyzes the computation directives and produces probe text (under option MODE = FULL/DIRECTIVES) which, when added to the source text, generates execution-time output interspersed with normal program output.

There are four JAVS computation directives: ASSERT, EXPECT, TRACE, and OFFTRACE. Each directive generates JOVIAL text which becomes part of the probe text. The user is cautioned that BASIC does no checking for errors in JAVS directives. Since the probe text is ultimately compiled, it must be error-free.

1.5.1 ASSERT Directive

The ASSERT directive provides the user with a means to monitor the truth value of a particular logical condition. The format of the ASSERT directive is:

".ASSERT,<JOVIAL-boolean-expression>"

The ASSERT directive may be placed wherever it is legal to place a simple IF statement followed by an assignment statement in the executable text. During Test Execution of a directive-instrumented module, a message is generated if the assertion is false.

During INSTRUMENT the ASSERT directive will generate the following code:

ITEM JAVS'ASSERT H 60 $

MONITOR JAVS'ASSERT $

IF NOT <JOVIAL-boolean-expression> $

JAVS'ASSERT = xxH(<JOVIAL-boolean-expression> AT STMT.

xxxx NOT TRUE.)$

where xx is determined and inserted automatically. If the <JOVIAL-boolean-expression> is longer than provided for in monitor output, it will be truncated. The statement number xxxx is computed during INSTRUMENTATION to be the statement number assigned by JAVS. The statement numbers are located in the PRINT, MODULE and PRINT, JAVSTEXT reports.
1.5.2 **EXPECT Directive**

The EXPECT directive provides the user with a means to monitor the value of selected variables and a method by which variables which exceed certain boundaries may be detected and reported. There are two formats for the EXPECT directive:

```
".EXPECT,<variable> = <low-value>,<high-value>"
".EXPECT,<variable> = <value>, TOLERANCE = <error>"
```

In the first form the expected range of values of the named variable is specified by `<low-value>,<high-value>`. In the second form the expected value together with a tolerance for error of the named variable are specified. The EXPECT directive must be placed in the body of executable text for a module. During Test Execution a message is produced if the assigned value of the named variable falls outside the declared value range.

The EXPECT directive generates a monitor statement, as follows:

```
for "<variable> = <low-value>,<high-value>"
MONITOR (<variable> LS <low-value> OR <variable> GR <high-value>)<variable>$
```

```
for "<variable> = <value>, TOLERANCE = <error>"
MONITOR (<variable> LS <value> - <error> OR <variable> GR <value> + <error>)<variable>$
```

1.5.3 **TRACE and OFFTRACE Directives**

The TRACE and OFFTRACE directives allow the user to observe the effect of computation upon the variables in the program at execution time. The TRACE directive establishes the place in the program flow where tracing is to begin for a specified variable; the OFFTRACE directive establishes the place where tracing ceases. The format for these directives are:

```
".TRACE,<name-1>,<name-2>,...,<name-n>"
".OFFTRACE,<name-1>,<name-2>,...,<name-n>"
```

The value of the named variable is monitored during Test Execution. For a particular variable, monitoring commences when the TRACE directive is encountered in the execution of the instrumented module; monitoring is terminated when the OFFTRACE directive is encountered.

The TRACE directive generates the following statements:
The OFFTRACE directive sets the specified flags to zero.
JAVS processes as a system of compatible tools which communicate through a common database. The order in which the processing steps are executed depends upon the task that the user wishes to accomplish. Source text (BASIC) processing must be executed first in order to build the database library.

The following tasks can be aided by JAVS, using the processing steps in the specified order:

<table>
<thead>
<tr>
<th>TASK</th>
<th>PROCESSING STEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text reformatting</td>
<td>BASIC</td>
</tr>
<tr>
<td>Documenting</td>
<td>BASIC, STRUCTURAL, ASSIST, DEPENDENCE</td>
</tr>
<tr>
<td>Debugging using directives</td>
<td>BASIC, STRUCTURAL, INSTRUMENT, Test Execution</td>
</tr>
<tr>
<td>Testing</td>
<td>BASIC, STRUCTURAL, INSTRUMENT, Test Execution, ANALYZER</td>
</tr>
<tr>
<td>Retesting</td>
<td>BASIC, STRUCTURAL, INSTRUMENT, Test Execution, ANALYZER, ASSIST, DEPENDENCE</td>
</tr>
</tbody>
</table>

In most cases BASIC, STRUCTURAL, and INSTRUMENT processing need to be performed only once.

The overlay version of JAVS consists of the same processing steps as the standard (non-overlay) version augmented by the following features:

- Smaller core size required to operate
- Macro commands to permit shorter command streams
- Operation of multiple JAVS processing steps within a single activity
- Extensibility for a planned interactive version of JAVS
2.1 SOURCE TEXT PROCESSING

The analysis of a JOVIAL program begins with BASIC, where a LIBNEW is made with information describing the texts (see Fig. 2.1). Each START-TERM sequence in the JOVIAL source is broken down into a series of invokable modules. The information saved for each module is a module descriptor block, a statement descriptor table, a statement table, a symbol locator table, and a symbol table. The results of BASIC processing can be displayed with the standard print commands (Sec. 4, PRINT, [option]).

Figure 2.1. BASIC processing
2.2 STRUCTURAL ANALYSIS

The analysis of a set of modules continues with STRUCTURAL, where information describing the graphical properties of each module is added to the information generated in BASIC. STRUCTURAL begins with a library created by BASIC. In BASIC, processing is done on one or more START-TERM texts, breaking each such text down into one or more invokable modules. STRUCTURAL processing is done on a single module at a time.

Normally, the library generated during BASIC is processed as LIBNEW, a read/write library during STRUCTURAL. However, if it is used as LIBOLD, a read only library, then a LIBNEW must also be provided. If LIBOLD is used, as each module is processed by STRUCTURAL, the BASIC data for the module is first copied to LIBNEW from LIBOLD, and then the new structural data is added. STRUCTURAL processing is shown in Fig. 2.2.

Figure 2.2. STRUCTURAL Processing
STRUCTURAL performs two types of computations:

1. Inter-statement analysis which establishes statement-to-statement links (e.g., in IF-THEN-ELSE statement sequences)
2. Module structural analysis which identifies the nature of program flow within the module.

The structural analysis performed by STRUCTURAL on each invokable module (program, procedure, or close) consists of determining all decision-to-decision paths (DD-paths) in the module. Decision-to-decision paths are sequences of statements between decision, or branch, points. Programs are represented as a directed graph of DD-paths, as shown in Fig. 2.3 and Fig. 2.4. Figure 2.3 shows the statement membership for each DD-path in module EXAMPLE. This module contains 12 DD-paths. Below each DD-path number is listed the order in which the statements are placed on each DD-path. For example, DD-path 2 consists of statements 15, 16, 29, 30, and 31, in that order.
Figure 2.3. Example of Structural Analysis
Legend

3 Numbers represent DD-paths
• Dots represent decision points
IFEITH Words represent decision statement types
(31) Parenthetical numbers represent statement numbers

Figure 2.4. Pictorial Example of Structural Analysis
2.3 MODULE INSTRUMENTATION

This step performs the instrumentation of modules which have been processed through BASIC and STRUCTURAL. The primary result of this step is a set of probed modules which can then be compiled in instrumented form for use in Test Execution (see Sec. 2.4). The instrumented modules are logically identical to the original source code. The probe statements are saved on the library along with data generated by BASIC and STRUCTURAL (e.g., the source text, etc.). INSTRUMENT processing is shown in Fig. 2.5.

* FROM STRUCTURAL

Figure 2.5. INSTRUMENT Processing
There are two types of instrumentation generated during INSTRUMENT:

1. **Structural Instrumentation.** Software probes are automatically inserted into the source text at each invocation point, each return point, and each statement which begins a DD-path. Each probe includes a call to special auditing routines which capture and record information concerning flow of control in the executing module(s).

2. **JAVS Directive Instrumentation.** Software probes are manually inserted in the source text for JAVS directives (Sec. 1.3) which are automatically expanded into JOVIAL code to monitor the results of assignment and exchange statements during Test Execution. Each directive controls execution-time output which is interspersed with normal program output.
2.4 MODULE TESTING ASSISTANCE AND SEGMENT ANALYSIS

The purpose of ASSIST is to provide computational and output reports for use by program testers to meet the following objectives:

- To provide "source text flow" information needed to support detailed analyses of the legitimacy of test cases
- To provide for identification of properties of sequences of DD-paths

ASSIST commands are designed to provide detailed information on specific DD-paths in modules. Some of the commands require that the DD-paths for which test assistance is needed be identified. Therefore, ASSIST is usually run after an initial series of test cases is run, and additional information is needed to assist the user in setting up new test cases that give coverage of specific DD-paths. ASSIST reports are most effectively used for retesting assistance and for computer documentation in conjunction with reports from the standard print commands and from DEPENDENCE (Sec. 2.5).

ASSIST operates from LIBOLD and, under active user control, generates reports of various kinds. It is assumed that BASIC and STRUCTURAL have been completed for each module present on LIBOLD for which analyses are requested by ASSIST commands. ASSIST processing is shown in Fig. 2.6.

---

**Figure 2.6. ASSIST Processing**
2.5 RETESTING GUIDANCE AND ANALYSIS

The purpose of DEPENDENCE is to provide the means to assess the impact of program modifications for a set of test cases. In contrast with ASSIST (which examines the details of program flow within a module), DEPENDENCE is concerned primarily with inter-module dependencies. Taken with the reports from both ASSIST and ANALYZER, the analyses performed by DEPENDENCE allow the tester to specify the subset of test cases which must be re-executed in order to maintain comprehensive testing coverage. The reports from DEPENDENCE coupled with some of the ASSIST reports provide valuable software documentation.

DEPENDENCE operates from LIBOLD, generating reports under active user control. It is assumed that BASIC and STRUCTURAL have been completed for every module present on LIBOLD. DEPENDENCE processing is shown in Fig. 2.7.

---

![Diagram of DEPENDENCE Processing](image)

*FROM STRUCTURAL

Figure 2.7. DEPENDENCE Processing
2.6 TEST EXECUTION

The purpose of Test Execution is to collect test execution data and per-
formance data from JAVS directives while executing a set of instrumented modu-
les produced by INSTRUMENT. Test Execution differs from other JAVS steps in
that the instrumented program itself is executed, not a JAVS processing step.
There is no access to the JAVS library during Test Execution. Rather, the
probe data is recorded on a test file; these data include the information
needed by ANALYZER to properly correlate with data on the library (e.g., module
name, test name, and DD-path number).

The instrumented program under test is first compiled, then loaded with
JAVS data collection modules. The program operates normally, reading its own
data and writing its own outputs. During program execution, the instrumented
modules call the data collection routines which record on AUDIT (the test file)
an execution trace and accumulated data on module invocations and return and
on DD-path traversals. Performance data resulting from JAVS computation
directives (Sec. 1.5) is interspersed with normal program output for the
printer. Test Execution processing is shown in Fig. 2.8.

Each Test Execution may consist of a number of test cases. The program
identifies the start of each new test case by executing a call to one of the
data collection routines; the end of all test cases is similarly treated.
These identification calls can be manually inserted by the user in the program
prior to compilation, or they can be inserted automatically during instrumen-
tation by using the INSTRUMENT, STARTTEST or INSTRUMENT, STOPTEST commands
(Sec. 5). All other instrumentation of the source is automatically performed
by INSTRUMENT.

2.6.1 JAVS Data Collection for Structural Instrumentation

There are four procedures in the JAVS data collection routines which may
be invoked during test execution:

- PROBE* is called at the beginning of each DD-path in an instru-
  mented module (INSTRUMENT option DDPATHS or FULL)
- PROBM* is called upon module invocation or return in an instru-
  mented module (INSTRUMENT option INVOCATION, DDPATHS or FULL)
- PROBI* is called to identify a new test case, or to end all
test cases
- PROBD* is called to record descriptive information about a
test on the test file

INSTRUMENT automatically generates the probes for PROBE and PROBM. The calls
to PROBI are inserted by the user, manually or by command, in the instrumented
source (see Sec. 2.6.2). The optional calls to PROBD are inserted manually by
the user.

* An arbitrary name used in data collection routines.
Figure 2.8. Test Execution Processing
The data collection routines utilize a COMMON block (PROBEC*) for communication with one another and for storage of data by module and DD-path. In the standard version of the COMPOOL for the data collection routines, provision is made for a maximum of 100 instrumented modules and 2000 DD-paths for those modules whose DD-paths are instrumented. If larger space is required, the data collection routines must be recompiled with appropriate changes made in their COMPOOL.

Caution: The program being analyzed may include symbols which conflict with the names used in the data collection routines for the procedures, the COMMON block, or the test file. Should such a conflict exist, (1) appropriate changes must be made in the source for the data collection routines and their COMPOOL and (2) INSTRUMENT processing must be done with the PROBE option command(s) to change the names of the data collection routines.

Summary data is accumulated in the workspace only after the first call is made to PROBI. Thereafter whenever a call is made to PROBI signalling the beginning of a new test case (or the end of all test cases) the accumulated data is recorded on AUDIT and the workspace is cleared. For each call to PROBE or PROBM, an execution trace record is recorded; for each call to PROBI, test case identification data is recorded; for each call to PROBD, descriptive information is recorded.

For each module the amount of data which it is possible to accumulate is controlled by the level of structural instrumentation done during INSTRUMENT. If the module is instrumented at the invocation level, only PROBM is called and DD-path executions cannot be recorded. If the module is instrumented at the DD-path level, both PROBM and PROBE are called, and full data acquisition can be made.

The amount of data actually recorded during a test is further controlled by a parameter to PROBE (see Sec. 2.6.2.2).

2.6.2 Compiling The Instrumented Program

During INSTRUMENT processing, the instrumented program source must first be written to a punch file from the library by using the command

```
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = <specification>.
```

while processing under JAVS execution. Next the instrumented program source is prepared for the compiler as described in the following subsections.

2.6.2.1 References to Data Collection Routines and Test File.

If the program has a COMPOOL, the procedure declarations for the data collection routines must be added to the COMPOOL.* If the program has no

*If the JOVIAL compiler accepts a list of processed COMPOOLS while compiling executable modules, then one for the source shown in Fig. 2.9 must be included in the list.

2-13
COMPOOL, the one shown in Fig. 2.9 must be provided. This is necessary in order to compile the instrumented source.

A FILE declaration for the test file (AUDIT) may also be necessary in the compilation process.* The location of the FILE declaration (i.e., in a COMPOOL or in the main program) is dependent on the JOVIAL compiler being used. Figure 2.9 contains the proper file statement for RADG.

---

* Some JOVIAL compilers require the FILE statement to be in the main program; others require it to be in a processed COMPOOL loaded with the executable code.
2.6.2.2 Test Case Identification and Test File Control.

At appropriate places in the instrumented source program (i.e., where a new test case begins and at the end of all test cases) a call to PROBI must be inserted. PROBI performs two services: it identifies each test case and controls the recording of data on the test file. PROBI has two parameters: the first is used as a test case name on ANALYZER reports and is a Hollerith name of 8 characters; the second is used to control the amount of data actually recorded on the test file.

The possible values for the test file control parameter TFLAG are shown in Table 2.1. A zero value signals the end of all test cases. A nonzero value signals the start of a new test case (and the end of the previous test case, if any). The value of the second parameter (if nonzero) determines the amount of execution tracing. If TFLAG is 1, no tracing is maintained; if TFLAG is 2, invocations and returns are traced; if TFLAG is 3, invocations, returns, and DD-paths are traced.

<table>
<thead>
<tr>
<th>TFLAG</th>
<th>SIGNAL</th>
<th>TEST-FILE DATA RECORDED</th>
<th>ANALYZER REPORT OPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>end-of-test file</td>
<td>(last) test case summary</td>
<td>SUMMARY, HIT, NOTHIT, MODLST, DDPATHS</td>
</tr>
<tr>
<td>1</td>
<td>new test case</td>
<td>test-case summary</td>
<td>SUMMARY, HIT, NOTHIT, MODLST, DDPATHS</td>
</tr>
<tr>
<td>2</td>
<td>new test case</td>
<td>test-case summary, module invocation/return trace</td>
<td>SUMMARY, HIT, NOTHIT, MODLST, TIME, MODTRACE, DDPATHS</td>
</tr>
<tr>
<td>3</td>
<td>new test case</td>
<td>test-case summary, module invocation/return trace, DD-path execution trace</td>
<td>SUMMARY, HIT, NOTHIT, MODLST, TIME, MODTRACE, DDTRACE, DDPATHS</td>
</tr>
</tbody>
</table>

2.6.2.3 Test Description.

At user option, descriptive information about the test can also be recorded on the test file by inserting PROBD calls. There are two parameters to PROBD (see Fig. 2.9): the first parameter is a Hollerith test description string of 80 characters, and the second parameter controls the initialization of data workspace. The value zero indicates the start of a test set (i.e., a new set of test cases), and a nonzero value indicates the start of a new test case. PROBD calls, if used, should immediately precede PROBI calls.

Each PROBD call results in a single line of output during ANALYZER processing. There is no limitation on the number of consecutive calls to PROBD. When the test file is analyzed by ANALYZER, the descriptive information is
printed on the output file as encountered. The purpose of PROBD is to allow
the user a facility for correlating program behavior to test results. For
example, strategically placed PROBD invocations can identify program linkages
executed for specific tests.

2.6.2.4 Compilation.

The procedure for compiling the instrumented source program with the
JOVIAL compiler is as follows:

1. Punch the instrumented text after INSTRUMENT execution.
   (Sec. 2.6.2).

2. If the text contains structural instrumentation (INSTRUMENT
default option, INVOCATION, DDPATHS or FULL) then
   - Provide COMPOOL definitions for data collection
     routines (Sec. 2.6.2.1).
   - Insert PROBI and PROBD calls (Secs. 2.6.2.2 and
     2.6.2.3).

3. Compile the COMPOOL and the instrumented program. If the text
   contains JAVS directives for performance data (INSTRUMENT
   options DIRECTIVES, FULL), use the MONITOR option* for the
   JOVIAL compiler. Additional core storage may be required for
   compiling the instrumented code.

Figure 2.10 shows an example listing of probed text with PROBI calls inserted.

*In the present JAVS implementation, JAVS performance directives make use of
the MONITOR feature available in some JOVIAL compilers, including the JOCIT
JOVIAL compiler at RADG.
Figure 2.10. Excerpt from Probed Text

2-17
2.6.3 Loading the Instrumented Program

If the instrumented program contains structural probes (i.e., calls to PROBM, PROBE, PROBI or PROBD) the JAVS data collection routines must be added to the load module for Test Execution. In addition, provision must be made for the test file to be saved on a suitable medium (e.g., tape or disk).

The requirements for main memory are larger in Test Execution than in normal program execution due to the following factors:

- Instrumented code is larger than non-instrumented code.
- JAVS data collection routines, workspace, and test file are needed for structural instrumentation.

Code expansion due to instrumentation depends on the nature of the source program. Experience has shown that code expansion of programs with moderately complex control structures can have 100% expansion from DD-path instrumentation. The JAVS data collection routines, although relatively small, utilize about 3000 words for data space in the standard configuration; the space required for the test file control information and buffer space is installation-dependent.

2.6.4 Test Execution Input and Output

The input data requirements for test execution are those for normal program execution. There is no input data read by the JAVS data collection routines.

In addition to normal program output, two types of output result from test execution of an instrumented program: a test file (for ANALYZER) which contains execution data collected from structural instrumentation probes and printout from the JAVS performance probes. Figure 2.11 contains a sample output from JAVS performance probes.

```
JAVS TEST CASE
*** MONITOR HOISTED DATA ID   # ONE
*** MONITOR HOISTED DATA ITERA # 10
*** MONITOR INTEGER DATA RESULT #
*** MONITOR HOISTED DATA JAVSASSER #
*** MONITOR INTEGER DATA INDEX #
*** MONITOR HOISTED DATA MESSAGE # RESULT OF 4

RESULT OF 4
*** MONITOR HOISTED DATA ID   # TWO
*** MONITOR HOISTED DATA ITERA # 300
*** MONITOR INTEGER DATA RESULT #
*** MONITOR HOISTED DATA JAVSASSER #
*** MONITOR INTEGER DATA INDEX #
*** MONITOR HOISTED DATA MESSAGE # RESULT OF 4

RESULT OF 4
```

Figure 2.11. Performance Data from JAVS Directives

2-18
2.7 TEST EFFECTIVENESS MEASUREMENT

The purpose of ANALYZER is to provide test effectiveness measurement and test execution tracing in terms of module invocation, DD-paths exercised, and statements executed. For a specified set of modules, ANALYZER supplies the following measures of test case effectiveness.

- A summary at the module level of the coverage achieved.
- A table of DD-paths exercised and the number of times each DD-path was traversed.
- A list of the DD-paths not executed.
- A source listing of each module including the number of times each statement was executed.
- A source listing of the key statements for each DD-path including the number of times each DD-path was executed.

ANALYZER supplies the following types of execution tracing:

- A list of the module invocations and returns and the level of invocation, or
- A list of the DD-paths interspersed with the module invocations and returns.

In addition, ANALYZER summarizes the time spent in each module as measured by the system clock.

ANALYZER operates from LIBOLD and a saved test file AUDIT produced by Test Execution. It is assumed that BASIC through INSTRUMENT and Test Execution have been accomplished for the specified modules. ANALYZER processing is shown in Fig. 2.12.
Figure 2.12. ANALYZER Processing
2.8 PROCESSING USING JAVS OVERLAY

+ JAVS STANDARD AND/OR MACRO COMMANDS
* SEE FIG. 2.8
** REPORTS FROM ANY PROCESSING STEP EXCEPT BASIC
++ FOR CORE SIZE EFFICIENCY TWO JAVS OVERLAY PROGRAMS ARE USED:
OVERLAY 1 IS THE COMPLETE SYSTEM; OVERLAY 2 DOES NOT CONTAIN
SYNTAX ANALYSIS (BASIC)
JAVS imposes certain restrictions on the size of the database library, the command language, and the source text to be analyzed. Most of the limitations based on size are generous (e.g., the maximum number of nested IF statements is 100). Some of the limitations, such as those in the Test Execution process, can be raised by recompiling parts of JAVS. Some of the restrictions are based upon computer page dimensions and cannot be changed.

A number of the constraints which affect the incoming JOVIAL source code are founded in the principle of analyzing invokable modules separately. The dialects of JOVIAL recognized by JAVS allow certain constructs involving jumps to global labels, invocations of externally declared switches, and TEST statements where the FOR variable is external to the module. Some of these constructs require modification of the JOVIAL source code, and some cause a warning message to be issued to make the user aware of limitations in the DD-path test measurement report and execution tracing.

The constraints are listed in sections (Sec. 3.2 through 3.8) appropriate to a particular JAVS processing step. Universal constraints (Sec. 3.1) affect all of JAVS processing. The terminology used in describing the constraints requires knowledge of JOVIAL and JAVS. The user should refer to the JOCIT Compiler Users Manual\textsuperscript{3} for JOVIAL terms and to the index in this manual for direction to the descriptions and references to JAVS terms.
3.1 UNIVERSAL CONSTRAINTS

1. Maximum 3 continuation cards for any given JAVS command.
3. Maximum 150 modules selected for any given JAVS command iteration loop.
4. Maximum 250 word-pairs in a statement block.*
5. Maximum 100 statements on any DD-path.
7. 100 JAVS errors produce a fatal error.
8. Maximum 150 known modules during any one JAVS execution (i.e., total modules in one or both libraries).
9. In order to change a previously made library, the name specified for an ALTER LIBRARY must be the same as when it was a CREATE LIBRARY.
10. Maximum 150 modules per JAVSTEXT.
11. Maximum 30 characters per card image read.
12. Maximum 128 characters per line of output.
13. No analysis is performed on a DIRECT code sequence.
14. The recognized dialects of JOVIAL include JOVIAL/J3, CDC JOVIAL, and Honeywell JOVIAL. The combination of these is processed by JAVS.

* BASIC automatically separates long source statements into a sequence of statements. The first statement is given the statement type according to the original JOVIAL source statement. Subsequent statements are of type continuation (CONT).
3.2 JAVS BASIC CONSTRAINTS

BASIC processing is capable of handling quite large source text files. Unusually large programs may have to be processed by several successive executions of BASIC, each operating on a separate file of START-TERM texts. In this event it is essential that JAVSTEXT directives are inserted in the source code prior to BASIC processing.

The following implementation constraints are the current ones which must be observed during BASIC processing:

1. Each module placed on the same library must have a unique module name for a given JAVSTEXT name. For this purpose only the first eight characters of any name are used. The first six characters should be unique (a compiler restriction).

2. COMPOOL texts should only be processed through BASIC.

3. If any errors are detected in JOVIAL source by BASIC, one or more statements on the library may be flagged as not parsed. The errors found in BASIC are summarized in Appendix B.

4. Hollerith and transmission type items must not exceed 120 characters.

5. Only one COMPOOL for each execution of BASIC can be placed on the library, when processing both COMPOOL and executable text. If one or more COMPOOLS are referenced by an executable text, the COMPOOL(S) need not be processed by BASIC, but the program text must contain a JAVSTEXT directive which indicates the presence of COMPOOL references.
1. A module should not GOTO a statement, directly or indirectly through a SWITCH, which is in another module.

2. A TEST statement must not appear in the range of a single factor FOR statement sequence.

3. Function calls with side effects (i.e., two successive calls to the function which produce different results) are not permitted in FOR, IF, IFEITH, and ORIF statements.

4. A declaration statement containing an END cannot appear as the last statement preceding the TERM statement of a program, procedure, or close.

5. The maximum depth of nesting of control or compound statements (IFEITH, ORIF, IF, FOR, BEGIN) is 100.

6. The maximum number of DD-paths which can begin at a statement is 100.

7. The maximum number of statements on a single DD-path is 100.

8. CLOSE invocations which appear as switch points in SWITCH declarations are treated as null switch points.

9. SWITCH invocations which appear as switch points (nested switches) in SWITCH declarations are treated as null switch points.

10. Invocations of externally declared switches are treated as RETURNS.

11. The first three factor FOR statement in a parallel FOR is assumed to be the controlling FOR. If there are no three factor FOR statements in the parallel FOR, the first FOR statement is chosen as the controlling FOR for the purpose of constructing interstatement pointers in the JAVS statement descriptor blocks.

12. All programs, close and procedures must contain a BEGIN-END set of statements surrounding the executable code.

13. CLOSE declarations within a FOR statement may not use a TEST statement to reference the active FOR variable.
3.4 JAVS INSTRUMENT CONSTRAINTS

1. The first three factor FOR statement in a parallel FOR is assumed to be the controlling FOR and is instrumented.

2. Invocations of externally declared switches are not instrumented.

3. The maximum number of nested IF statements is 100.

4. Subscript expressions in SWITCH invocations are limited to 72 characters.

5. Item names and switch names are limited to 30 characters in length.

6. The maximum number of variables allowed in a single TRACE directive is 18.

7. The maximum number of variables in TRACE directives is limited to 999 per module.

8. The maximum number of nested ITEITH and/or ITEITH/ORIF statements is 100.

9. The maximum DD-path number is 9999.

10. The maximum length of the TEXT in an ASSERT directive is 72 characters.

11. The functional modifiers BIT, CHAR, MANT, NENT, POS and BYTE may not appear in the numeric formulas for FOR statements; i.e., no side-effects are allowed in the initial value formula.

12. FOR variable may not appear in TRACE or EXPECT directives.

13. A declaration statement containing an END cannot appear as the last statement preceding the TERM statement of a program, procedure or close.
3.3 JAVS ASSIST CONSTRAINTS

1. Maximum of 100 DD-paths per reaching set path.
2. Maximum of 100 outways per decision.
3. Maximum of 1200 DD-paths per analyzed module for reaching set.
4. Maximum of 2400 statements per analyzed module for reaching set.
5. Maximum of 200 statements in reaching set.
6. Maximum 100 JAVSTEXTs specified for cross-reference mapping.
3.6 TEST EXECUTION CONSTRAINTS

There is no probe test data recorded on AUDIT until after the first invocation of PROBI has been made. (PROBI invocations must be inserted manually or by using the INSTRUMENT, STARTTEST and INSTRUMENT, STOPTEST commands.) The PROBI invocation which indicates end-of-test must be executed to properly terminate data recording on AUDIT. When this end-of-test invocation is made, a message is issued.

There is a limitation of 100 instrumented modules in the workspace for the JAVS data collection routines. This limitation may be changed by recompiling the routines.

There is a limitation of 2000 instrumented DD-paths in the workspace for the JAVS data collection routines. This limitation may be changed by recompiling the routines.
3.7 JAVS DEPENDENCE CONSTRAINTS

1. Maximum 100 modules in each of the groups in group dependency analysis.

2. Maximum of 115 modules in sum of group A and group C in group dependency analysis.

3. Maximum bandwidth of 5 specified in BAND analysis.
3.8 JAVS ANALYZER CONSTRAINTS

In a single execution of ANALYZER, either MODTRACE or DDTRACE option may be used, but not both.

Since ANALYZER does not reposition the file AUDIT at the conclusion of processing, two consecutive executions of ANALYZER in the same command set process different test cases on AUDIT.

Maximum of 100 test cases is analyzed.

If the test termination PROBI module is not invoked, ANALYZER will process the test case data on the AUDIT file up to the start of the last test case. When ANALYZER attempts to process the last test case on AUDIT and hits an end-of-file before the required 4H(TERM), written on AUDIT by the test termination call to PROBI, the error message "Illegal record on AUDIT file" will be issued. Thus, if there is only one test case on the AUDIT file and no occurrence of 4H(TERM), ANALYZER will have no summary data to process.
JAVS commands

JAVS processing is implemented by a command language (see Sec. 1.2). This section describes the eight basic types of commands. Each JAVS command starts with a keyword. The keywords are intended to be descriptive, concise, while maintaining flexibility through their processing options. Table 4.1 shows the relationship between the keywords and basic types of commands. A partial list is given below for process option, execution, and print/punch commands. Complete lists are given for the remaining types of commands.

Section 5 contains a complete description for each JAVS command. The commands in Sec. 5 are presented alphabetically.

- **Library commands** - (Sec. 4.1)
  
  OLD LIBRARY = <name>.
  CREATE LIBRARY = <name>.
  ALTER LIBRARY = <name>.
  DESCRIBE = <ON/OFF>.
  MERGE.

- **Startup command** - (Sec. 4.2)
  
  START.

- **Module Selection commands** - (Sec. 4.3)
  
  JAVSTEXT = <name>.
  MODULE = <name>.
  FOR LIBRARY.
  FOR JAVSTEXT.
  FOR MODULE = <name-1>,...,<name-n>.
  END FOR.

- **Process Option commands** (an incomplete list) - (Sec. 4.4)
  
  BASIC,COMMENTS = OFF.
  INSTRUMENT,MODE = FULL.
### TABLE 4.1

**KEYWORDS FOR COMMANDS**

<table>
<thead>
<tr>
<th>COMMAND TYPE</th>
<th>KEYWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>OLD LIBRARY CREATE LIBRARY ALTER LIBRARY DESCRIBE MERGE</td>
</tr>
<tr>
<td>Startup</td>
<td>START</td>
</tr>
<tr>
<td>Module Selection</td>
<td>JAVSTEXT MODULE FOR LIBRARY FOR JAVSTEXT FOR MODULE END FOR</td>
</tr>
<tr>
<td>Process Option and</td>
<td>BASIC STRUCTURAL INSTRUMENT ASSIST DEPENDENCE</td>
</tr>
<tr>
<td></td>
<td>ANALYZER</td>
</tr>
<tr>
<td>Print/Punch</td>
<td>PRINT PUNCH</td>
</tr>
<tr>
<td>Termination</td>
<td>END</td>
</tr>
<tr>
<td>Macro*</td>
<td>BUILD LIBRARY PROBE DOCUMENT TEST</td>
</tr>
</tbody>
</table>

*Available only with JAVS overlay version.*
- Process execution commands (an incomplete list) - Sec. 4.5
  BASIC.
  STRUCTURAL.
  INSTRUMENT.
  ASSIST, CROSSREF, LIBRARY.
  DEPENDENCE, GROUP, LIBRARY.
  ANALYZER, SUMMARY.

- Standard print and punch commands (incomplete list) - Sec. 4.6
  PRINT, MODULE.
  PUNCH, JAVSTEXT = <name>.
  PRINT, DDPATHS.

- Run termination command - Sec. 4.7
  END.

- Macro command - Sec. 4.8
  BUILD LIBRARY (=name).
  PROBE (,options).
  DOCUMENT (,options).
  TEST (,options).
4.1 LIBRARY COMMANDS

Three of the library commands (OLD LIBRARY, CREATE LIBRARY, and ALTER
LIBRARY) define the name and status of the library (or libraries) during the
current run. DESCRIBE command sets a flag to print the library manager
statistics at the end of the run. The MERGE command causes the contents of
the library specified in the OLD LIBRARY command to be written unto the library
specified by CREATE or ALTER LIBRARY command.

4.1.1 Library Description Commands

- **OLD LIBRARY = <name>**. This option specifies whether LIBOLD
  is to be used in the current run, as is normally the case
  in all except BASIC and STRUCTURAL. The absence of this
  command indicates that no old library will be used. LIBOLD
  is a read-only library.

- **CREATE LIBRARY = <name>**. This option specifies that LIBNEW (a
  "new library") will be created during the current run, as
  is normally the case in BASIC. When this option is speci-
  fied, LIBNEW is assumed to be empty at the beginning of a
  run. LIBNEW is used as a read/write library.

- **ALTER LIBRARY = <name>**. This option specifies that LIBNEW is
  a previously generated library to be modified during the
  current run, as is normally the case in STRUCTURAL. The
  name of LIBNEW should be the same one used when it was
  first created. LIBNEW is used as a read/write library.

These three library description commands specify two libraries during
JAVS processing. There are five combinations of library commands for specifying
JAVS libraries.

1. **CREATE LIBRARY = <name>**.

   The only processing step for which LIBNEW alone is specified is BASIC.
   The overlay version of JAVS allows this library command to be used in multiple
   step processing without additional library definition. The overlay particulars
   are described along with other machine dependent information in the JAVS User's
   Guide.

2. **OLD LIBRARY = <name>**.

   LIBOLD by itself is used whenever no new information is to be added to
   the library. It can be used in any processing step except BASIC. Processing
   steps ASSIST, DEPENDENCE and ANALYZER use this library; INSTRUMENT can
   use LIBOLD and save the instrumented file (LPUNCH) on disk, tape, or punched
   cards. This method of not storing the instrumented source in the library (by
not using ALTER or CREATE LIBRARY) is useful if the library is already large before instrumentation.

Although LIBNEW must be used for processing subsequent to STRUCTURAL, there may be instances where a preliminary STRUCTURAL run, using only LIBOLD, is desirable. JAVS printout reflects any database modifications made during a run; however, the modifications are stored only on LIBNEW (the read/write library). Thus LIBOLD remains unchanged.

3. ALTER LIBRARY = <name>.

This command modifies LIBNEW and saves the altered library. STRUCTURAL and INSTRUMENT data are often saved on the database library in this manner.

4. OLD LIBRARY = <name-1>.
CREATE LIBRARY = <name-1>.

This set of library commands allows the creation of a new library by copying the contents of LIBOLD for the specified modules onto LIBNEW and then updating LIBNEW during the current run. LIBOLD remains unchanged.

5. OLD LIBRARY = <name-1>.
ALTER LIBRARY = <name-2>.

This set of commands differs from the previous combination in several respects: (1) LIBSEW must be a previously created library, (2) for the specified modules already on LIBNEW, data from the current run is added to LIBNEW, and (3) modules previously only on LIBOLD are copied onto LIBNEW and updated by the current run. LIBOLD remains unchanged.

Either of the last two sets of library description commands can be used in any processing step when a library merge (see Sec. 4.1.3) is desired. They can also be in STRUCTURAL and INSTRUMENT when an unchanged library, LIBOLD, is desired as well as an updated LIBNEW.

All library description commands must precede the START command.

4.1.2 Library Access Command

- DESCRIBE = ON. or DESCRIBE = OFF. This option turns ON or OFF a sequence of printouts which describe in detail the access interface to the libraries. The default is OFF. This option is intended for JAVS maintenance. The DESCRIBE command must precede the START command.

4.1.3 Library Process Command

The MERGE command can be used (1) to add new modules to an existing library, and (2) to replace old modules with new modules having the same module name and text name. The command is MERGE.
The MERGE command merges a LIBOLD with a LIBNEW. After merging LIBNEW contains all modules on the original LIBNEW, plus all modules on the LIBOLD that do not have the same module name and text name as a module on the LIBNEW. Both LIBOLD and LIBNEW must be specified in one of the two sets of library description commands (see Sec. 4.1.1).

The MERGE command must be preceded by the START command.
4.2 STARTUP COMMAND

The startup command terminates the library description and indicates the start of processing. The command is

\[
\text{START.}
\]

and causes the report in Fig. 4.1 to be printed. See command END in Sec. 5 for a description of the items in this report.

Figure 4.1.
4.3 MODULE SELECTION COMMANDS

Many JAVS commands require specification of the particular module upon which computations are to be performed. (Some of the standard print commands require this, for example.) If both LIBNEW and LIBOLD have been declared, the module selector will first look in LIBNEW for the module and, if not found, will look in LIBOLD. Only the version on LIBNEW will be selected if a module with the same name appears on both libraries. If two or more versions of a module appear on a library, the last one entered on the library will be selected.

A module is identified by its START-TERM text name and its module name. The text name is taken from the JAVSTEXT directive which appears at the beginning of the START-TERM text sequence. If there is no JAVSTEXT directive, the name "NOJAVS" is automatically supplied by BASIC. The module name is taken from the JOVIAL statement which identifies the module (e.g., program, *PROC, CLOSE, COMPOOL). Only the first eight characters of the name on the JOVIAL statement are used for the module name. The first eight characters of the name on the JAVSTEXT are used as the text name (Sec. 1.4). The first six characters of each JAVSTEXT on the library must be unique.

4.3.1 START-TERM Text Selection

Some JAVS commands require specification of the START-TERM text sequence upon which computations are to be performed. START-TERM texts are known to JAVS by their names. The following command makes the "current text" be the named one:

\[\text{JAVSTEXT} = \text{<text-name>}.\]

All subsequent commands (if they refer to a specific text) are applied to the specified text. There can be any number of JAVSTEXT= commands. A new JAVSTEXT= command replaces and supersedes a previous JAVSTEXT= command.

4.3.2 Single Module Selection

Modules are known to JAVS by their names and the text to which they belong. The following makes the "current module" within the "current text" be the named one:

\[\text{MODULE} = \text{<name>}.\]

All subsequent commands (if they refer to a specific module) are applied to the specified module within the "current text." If no "current text" has been specified, the module selector will locate the module using only its name. There can be any number of MODULE= commands.

*Some JOVIAL compilers require that the main program be specified with a PROGRAM statement.
4.3.3 Multiple Module Iteration

In many applications, the user will want to repeat a command for a number of different modules. The three forms of command iteration structure are described below.

4.3.3.1 Selected Module Iteration.

The following sequence selects a number of modules, by name, and iterates a block of commands (which cannot contain another iteration) once for each specified module:

\[
\text{FOR}\text{ Module } = \langle\text{name-1}, \langle\text{name-2}, \ldots, \langle\text{name-n}\rangle.}
\]

(any set of commands)

END FOR.

4.3.3.2 ALL—Modules Iteration for Specified JAVSTEXT (START—TERM Text).

The following sequence selects each known module within the "current text" and iterates a block of commands (which cannot contain another iteration) once for each known module of that text:

\[
\text{FOR JAVSTEXT.}
\]

(any set of commands)

END FOR.

4.3.3.3 ALL—Modules Iteration for Library.

The following sequence selects each known module on the library and iterates a block of commands (which cannot contain another iteration) once for each known module:

\[
\text{FOR LIBRARY.}
\]

(any set of commands)

END FOR.

4.3.4 Module Selection Constraint

The maximum number of modules which may be specified by a single iteration is 150 (see Sec. 3).
4.4 PROCESS OPTION COMMANDS

Processing steps BASIC, STRUCTURAL, INSTRUMENT and ANALYZER have option commands which define the action taken when the process execution command is given. The option commands follow the START command and precede the appropriate execution command (see Sec. 4.5).

4.4.1 BASIC Option Commands

The BASIC options are specified by the following commands (with the default value assumed in case the option command is not given prior to the appearance of the BASIC execution command):

- BASIC,CARD IMAGES = ON/OFF.  DEFAULT = ON.
- BASIC,COMMENTS = ON/OFF.  DEFAULT = ON.
- BASIC,DEFINES = ON/OFF.  DEFAULT = ON.
- BASIC,ERRORS = ON/OFF/LIMIT/TRACE.  DEFAULT = ON.
- BASIC,SYMBOLS = ON/OFF/PARTIAL.  DEFAULT = PARTIAL.
- BASIC,TEXT = PRESET/COMPUTE/BOTH/JAVSTEXT.  DEFAULT = COMPUTE.

BASIC options which have been selected in the command sequence remain in effect until they are reset by a later command. See Sec. 5 for a complete definition and example of each BASIC option command.

The user will note that module selection commands do not apply until after a module has been added to a library during BASIC processing by the BASIC verb. The name assigned to a module when it is added to a library is the first eight characters of the program name, procedure name, or close name of the module. The first six characters of the names should be unique.

It is important that the command sequence involving one or more occurrences of the BASIC execution command be terminated with the END command so that LIBNEW is closed properly. The user may employ the standard print and punch commands (Sec. 5) after the BASIC command and after module or JAVSTEXT specification (Sec. 4.3).

4.4.2 STRUCTURAL Option Command

The STRUCTURAL print option is specified by the following command (with the specified default value assumed in case the command is not given prior to the appearance of the STRUCTURAL verb):

- STRUCTURAL,PRINT = SUMMARY/DEBUG.  (DEFAULT = SUMMARY)
The STRUCTURAL option which has been selected in the command sequence remains in effect until it is reset by a later option command. Each instance of the STRUCTURAL execution command in a command sequence assumes that a module has been selected, and that the STRUCTURAL execution command will be applied only once to this module.

See Sec. 5 for a definition and example of the STRUCTURAL print option.

### 4.4.3 INSTRUMENT Option Commands

The INSTRUMENT options are specified by the following commands (with the specified default value assumed in case the command is not given prior to the appearance of the INSTRUMENT execution command):

- **INSTRUMENT,MODE = INVOCATION/DDPATHS/DIRECTIVES/FULL.**
  
  DEFAULT = DDPATHS.

- **INSTRUMENT,PROBE,DDPASS = <probe-name>.**
  
  DEFAULT = PROBE*

- **INSTRUMENT,PROBE,MODULE = <invocation-name>.**
  
  DEFAULT = PROBM*

- **INSTRUMENT,PROBE,TEST = <test-name>.**
  
  DEFAULT = PROBI*

INSTRUMENT options which have been selected in the command sequence remain in effect only during INSTRUMENT processing of the current module. Thus if the same option is requested for a group of modules, the INSTRUMENT option and INSTRUMENT execution command should be placed within the module selection iteration (see Sec. 4.3.3).

### 4.4.4 ANALYZER Option Commands

The ANALYZER options are specified by the following commands. There are few default options for ANALYZER, since most option commands produce one or more reports.

Unlike the other processing steps, ANALYZER has its own module selection which does not use a JAVSTEXT specification. If there are several modules with the same name, but with different JAVSTEXT names, coverage reports should be specified by the option:

*PROBE is an arbitrary name used for the DD-path data collection routine.
PROBM is an arbitrary name used for the module-invocation-and-return data collection routine.
PROBI is an arbitrary name used for the test-beginning-and-test-end data collection routine.
PROBD is an arbitrary name for the description writing routine.
ANALYZER, ALL MODULES.

If not all of the modules were instrumented, they will be noted as "not traced."

The ANALYZER options are of the following format:

ANALYZER, MODULE = <name-1>, ..., <name-n>.
ANALYZER, ALL MODULES.
ANALYZER, SUMMARY.
ANALYZER, HIT.
ANALYZER, NOTHIT.
ANALYZER, MODLST.
ANALYZER, MODTRACE.
ANALYZER, DDPTTRACE.
ANALYZER, TIME.
ANALYZER, DDPATHS.
ANALYZER, CASES = <number>. DEFAULT = 10.
ANALYZER, FACTOR = <percent increase>. DEFAULT = 0.
ANALYZER, ALL.

A description and example for each option is given in Sec. 5.
4.5 PROCESS EXECUTION COMMANDS

For the four processing steps described in the option commands, the process execution commands are:

BASIC.
STRUCTURAL.
INSTRUMENT.
ANALYZER.

Each of these commands causes execution of the processing step on the specified modules (except for BASIC, where modules are not specified) according to the previously specified options or their default values.

The ASSIST command set contains only execution commands. Module selection must be made for these commands.

ASSIST,REACHING SET,<specifications>.
ASSIST,CROSSREF,<specifications>.
ASSIST,PICTURE[,<options>].
ASSIST,STATEMENTS.

The DEPENDENCE command set contains only execution commands; some commands require module specification (see Sec. 4.3.2) and some commands operate on the entire library. (See Sec. 5 for a detailed description of each command.) All DEPENDENCE commands produce a report based on intermodule dependencies.

The DEPENDENCE module commands are:

DEPENDENCE,PRINT,INVOKES.
DEPENDENCE,BANDS = <number>. (DEFAULT = 2)
DEPENDENCE,TREE.
The DEPENDENCE library commands are:

DEPENDENCE, GROUP MODULES = <name-1>, ..., <name-n>.

DEPENDENCE, GROUP, LIBRARY

DEPENDENCE, GROUP, AUXLIB.

DEPENDENCE, SUMMARY.
4.6 STANDARD PRINT AND PUNCH COMMANDS

4.6.1 PRINT

The standard print commands provide the means to generate formatted renditions of JAVS internal tables. These print commands are universal; i.e., they can be used in any processing step. Ordinarily, these commands are used after STRUCTURAL to print the key tables generated during BASIC and STRUCTURAL. The standard print commands are of the form:

PRINT, <table-name>.

PRINT, JAVSTEXT=<name>[, options].

where <table-name> is DDP, DDPATHS, DMT, MODULE, SE, SDE, SLT, STB or SYMTAB.

For a PRINT table to be accepted, a module must have been selected with a module selection command (see Sec. 4.3.2). The PRINT JAVSTEXT commands are used to display the START-TERM text as it is presented to the JOVIAL compiler. The options specify instrumented or uninstrumented source text.

4.6.2 PUNCH

The standard punch commands provide the means to generate reformatted versions of the source text in preparation for the JOVIAL compiler. The source text is written onto file LFENCE (see Table 1.1 in Sec. 1.1.3). The standard punch commands are of the form:

PUNCH, MODULE.

PUNCH, JAVSTEXT=<name>[, options].

The PUNCH MODULE command requires module selection to be made prior to the command. The JAVSTEXT punch command is used to prepare the complete START-TERM text for compilation. The options specify whether the text is to be output to LFUNCH in its instrumented or uninstrumented form.

A complete description of each standard print and punch command is given in Sec. 5.
4.7 RUN TERMINATION COMMAND

A JAVS run terminates on the END command, which provides for correctly closing any files and writing a wrapup summary. If the wrapup report, shown in Fig. 4.2, is not printed, then LIBNEW was not closed properly and cannot be used in subsequent runs.

The run termination command is

END.

---

**Figure 4.2.**

---

**JAVS WRAPUP...**

**KNOWN MODULE DESCRIPTOR BLOCKS...**

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>TEST</th>
<th>PARENT</th>
<th>PROC PROBE</th>
<th>EXEC FIRST WORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EXEMPL</td>
<td>YES</td>
<td>EXEMPL</td>
<td>10</td>
<td>112</td>
</tr>
<tr>
<td>2</td>
<td>EXEMPL</td>
<td>YES</td>
<td>EXEMPL</td>
<td>10</td>
<td>112</td>
</tr>
<tr>
<td>3</td>
<td>EXEMPL</td>
<td>YES</td>
<td>EXEMPL</td>
<td>10</td>
<td>112</td>
</tr>
<tr>
<td>4</td>
<td>EXEMPL</td>
<td>YES</td>
<td>EXEMPL</td>
<td>10</td>
<td>112</td>
</tr>
<tr>
<td>5</td>
<td>EXEMPL</td>
<td>YES</td>
<td>EXEMPL</td>
<td>10</td>
<td>112</td>
</tr>
</tbody>
</table>

**LIBRARY INFORMATION—RUN OF 3731**

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>DATE</th>
<th>LAST</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 TEST</td>
<td>3731</td>
<td>3731</td>
<td>17740</td>
</tr>
</tbody>
</table>

---

**BEST AVAILABLE COPY**
This section contains a complete list of JAVS commands, in alphabetical order, along with the JAVS processing step in which each command is used. The term "universal" indicates that the command can be used in any processing step.*

Following the list of commands is a description, accompanied with sample output and command sets, of each JAVS command.

*The overlay version of JAVS allows all commands to be "universal."
JAVS COMMANDS (DEFAULTS UNDERLINED)

ALTER LIBRARY = <libname>.
ANALYZER.
ANALYZER,ALL.
ANALYZER,ALL MODULES.
ANALYZER,CASES = <number>.
ANALYZER,DDPATHS.
ANALYZER,DDTRACE.
ANALYZER,FACTOR = <percent-increase>.
ANALYZER,HIT.
ANALYZER,MODLIST.
ANALYZER,MODTRACE.
ANALYZER,MODULE = <name-1>,<name-2>,...,<name-n>.
ANALYZER,NOTHIT.
ANALYZER,SUMMARY.
ANALYZER,TIME.
ASSIST,CROSSREF,JAVSTEXT = <text-name-1>,<text-name-2>,...,<text-name-n>.
ASSIST,CROSSREF,LIBRARY.
ASSIST,PICTURE.
ASSIST,PICTURE{,CONTROL}{,NOSWITCH}.
ASSIST,REACHING SET,<number-to>{,<number-from>}
{,PICTURE{,ITERATIVE}}.
ASSIST,STATEMENTS.

BASIC.
BASIC,CARD IMAGES = ON/OFF.
BASIC,COMMENTS = ON/OFF.
BASIC,DEFINES = ON/OFF.
BASIC,ERRORS = ON/OFF/LIMIT/TRACE.
BASIC,Symbols = ON/OFF/PARTIAL.
BASIC,TEXT = COMPUTE/BOTH/PRESET/JAVSTEXT.

*BUILD LIBRARY (= <library name>).

CREATE LIBRARY = <libname>.

*Can be used only with the overlay version.

5-2
JAVS COMMANDS (DEFAULTS UNDERLINED)

DEPENDENCE,BANDS.
DEPENDENCE,BANDS = <number>.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,MODULES = <name—1>,<name—2>,...,<name—n>.
DEPENDENCE,PRINT,INVOKES.
DEPENDENCE,SUMMARY.
DEPENDENCE,TREE.
DESCRIBE = ON/OFF.

*DOCUMENT,JAVSTEXT={<text-name>},{MODULE=<name—1>,...}.

END.
END FOR.

FOR JAVSTEXT.
FOR LIBRARY.
FOR MODULE = <name—1>,<name—2>,...,<name—n>.

INSTRUMENT.
INSTRUMENT,MODE = INVOCATION/DDPATHS/DIRECTIVES/FULL.
INSTRUMENT,PROBE,DDPATH = <probe—name>.
INSTRUMENT,PROBE,MODULE = <invocation—name>.
INSTRUMENT,PROBE,TEST = <test—name>.
INSTRUMENT,STARTTEST = <modname>,<textname>,<stmt. no.> {,<TESNAM>}{,<TFLAG>},
INSTRUMENT,STOPTEST = <modname>,<textname>,
<stmt. no.>.

JAVSTEXT = <text—name>.

MERGE.

MODULE = <name>.

OLD LIBRARY = <libname>.

* Can be used only with the overlay version.

5-3
<table>
<thead>
<tr>
<th>JAVS COMMANDS (DEFAULTS UNDERLINED)</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINT,DP.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,DPATHS.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,DNT.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,JAVTEXT = &lt;text-name-1&gt;, INSTRUMENTED = ALL.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,JAVTEXT = &lt;text-name&gt;, INSTRUMENTED = &lt;name-1&gt;,</td>
<td>(Universal)</td>
</tr>
<tr>
<td>&lt;name-2&gt;, ..., &lt;name-n&gt;.</td>
<td></td>
</tr>
<tr>
<td>PRINT,JAVTEXT = &lt;text-name&gt;.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,MODULE.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,SB.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,SB3.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,SLT.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PRINT,STB.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>*PROBE{,JAVTEXT = &lt;text-name&gt;{,MODULE = &lt;name-1&gt;, ...}}.</td>
<td>INSTRUMENT,</td>
</tr>
<tr>
<td></td>
<td>(Universal)</td>
</tr>
<tr>
<td>*PROBE,STARTTEST = &lt;modname&gt;, &lt;textname&gt;, &lt;stmt. no.&gt;</td>
<td>INSTRUMENT,</td>
</tr>
<tr>
<td>{,TESNAM{,TFLAG}.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>*PROBE,STOPTEST = &lt;modname&gt;, &lt;textname&gt;, &lt;stmt. no.&gt;.</td>
<td>INSTRUMENT,</td>
</tr>
<tr>
<td></td>
<td>(Universal)</td>
</tr>
<tr>
<td>PUNCH,JAVTEXT = &lt;text-name&gt;.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PUNCH,JAVTEXT = &lt;text-name&gt;, INSTRUMENTED = ALL.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>PUNCH,JAVTEXT = &lt;text-name&gt;, INSTRUMENTED = &lt;name-1&gt;,</td>
<td>(Universal)</td>
</tr>
<tr>
<td>&lt;name-2&gt;, ..., &lt;name-n&gt;.</td>
<td></td>
</tr>
<tr>
<td>PUNCH,MODULE.</td>
<td>(Universal)</td>
</tr>
<tr>
<td>START.</td>
<td>(Startup)</td>
</tr>
<tr>
<td>STRUCTURAL.</td>
<td>STRUCTURAL</td>
</tr>
<tr>
<td>STRUCTURAL,PRINT = SUMMARY/DEBUG.</td>
<td>STRUCTURAL</td>
</tr>
<tr>
<td>*TEST{,MODULE = &lt;name-1&gt;, &lt;name-2&gt;, ..., &lt;name-n&gt;}.</td>
<td>ANALYZER,</td>
</tr>
<tr>
<td></td>
<td>(Universal)</td>
</tr>
</tbody>
</table>

*Can be used only with the overlay version.
ALTER LIBRARY = <name>.

Description

This option specifies that LIBNEW is a previously generated library to be modified during the current run. LIBNEW is used as a read/write library.

Rule

The name of LIBNEW must be the same one used when it was first created.

Example Command Sets

(1) ALTER LIBRARY = REFMAN.
    START.
    FOR LIBRARY.
    STRUCTURAL.
    END FOR.
    END.

    This command set will modify the library created by the BASIC processing step and add the STRUCTURAL data to the library for each module on the library.

(2) ALTER LIBRARY = REFONE.
    START.
    BASIC.
    END.

    This command set augments a previously built library with the BASIC processing of additional source text.
Description

The ANALYZER execution command causes processing of the post-test analysis. If no ANALYZER report options are present, no report is produced.

Rules

For a single ANALYZER command,

(1) Either option MODTRACE or DDTRACE can be used, but not both.

(2) A maximum of 100 test cases is analyzed.

(3) The ANALYZER command does not reposition the AUDIT file; thus consecutive ANALYZER commands process different numbers of test cases, dependent upon the ANALYZER,CASES command.

Example Command Sets

(1) OLD LIBRARY = REFMAN.
START.
ANALYZER, ALL MODULES.
ANALYZER, ALL.
ANALYZER.
END.

This set of commands will produce the SUMMARY, HIT, NOTHIT, TIME, DDPATHS, MODLIST, DDTRACE post-test analysis reports for all modules on the library.

(2) OLD LIBRARY = TRFMAN.
START.
ANALYZER,CASES = 50.
ANALYZER,MODULE = EXPROGM,EXMPL1.
ANALYZER, SUMMARY.
ANALYZER, NOTHIT.
ANALYZER, DDPATHS.
ANALYZER, DDTRACE.
ANALYZER.
END.
Description

This command is equivalent to the following set of commands:

```
ANALYZER, SUMMARY.
ANALYZER, HIT.
ANALYZER, NOTHIT.
ANALYZER, TIME.
ANALYZER, DDPATHS.
ANALYZER, MODLST.
ANALYZER, DDPTRACE.
```

Example Command Sets

(1) OLD LIBRARY = REFMAN1.
START.
ANALYZER, MODULE = EXPROGM, EXMPL1.
ANALYZER, ALL.
ANALYZER.
END.

(2) OLD LIBRARY = REFMAN1.
START.
ANALYZER, ALL MODULES.
ANALYZER, CASES = 100.
ANALYZER, ALL.
ANALYZER.
END.
**Description**

The module specification commands are used to identify the modules on LIBOLD for which ANALYZER processes the summary data on the Test Execution trace file. The default is no modules specified. To specify all known modules, use the command:

```
ANALYZER,ALL MODULES.
```

To specify a subset of modules, use the command:

```
ANALYZER,MODULE = <name-1>,<name-2>,...,<name-n>.
```

The following ANALYZER options require module specification: SUMMARY, HIT, NOTHIT, TIME, MODLIST, DDPATHS, ALL.

**Example Command Sets**

1. **OLD LIBRARY = REFTEST.**
   START.
   ANALYZER,ALL MODULES.
   ANALYZER,ALL.
   ANALYZER.
   END.

2. **OLD LIBRARY = REFMAN.**
   START.
   ANALYZER,FACTOR = 20.
   ANALYZER,TIME.
   ANALYZER,ALL MODULES.
   ANALYZER.
   END.
ANALYZER, CASES = <number>.

Description

This command is used to specify the maximum number of test cases to process from the test file. The absence of this command specifies the default of 10 test cases. A test case is determined by each invocation of the JAVS data collection routine PROBI with the second parameter, TFLAG, equal to 1, 2, or 3 (see Sec. 2.6.2.2). ANALYZER reads and analyzes the data for each test case encountered on the test file. At the conclusion of the specified number of test cases, or at the end of the last test case if less than the number specified, the ANALYZER reports are produced and all further analysis of the test file is inhibited. The maximum number of test cases which ANALYZER can process is 100.

Example Command Sets

(1) OLD LIBRARY = REFMAN.
    START.
    ANALYZER, CASES = 2.
    ANALYZER, SUMMARY.
    ANALYZER, MODULE = EXMPL1.
    ANALYZER.
    END.

This command set will produce the post-test summary report for only the first two test cases for module EXMPL1, even if there are more than two test cases on the test file.

(2) OLD LIBRARY = REF.
    START.
    ANALYZER, CASES = 100.
    ANALYZER, ALL MODULES.
    ANALYZER, ALL.
    ANALYZER.
    END.

This command set raises the number of allowable test cases from the default of 10 to the maximum limit of 100.
## Description

For each specified module, this command causes ANALYZER to produce a source listing for the statements which begin a DD-path, showing the DD-path description and an execution count reflecting DD-path coverage for all test-cases encountered on the trace file. The report is similar in format to the report from PRINT, DDPATHS, with execution counts appended. DD-path coverage is summarized at the end of the report. If a specified module is not invoked during the test, this report is suppressed.

### Sample Output

**MODULE DDPATHS COVERAGE LISTING**

**MODULE EXAMPLE --> JAVASCRIPT --> EXPERIMENT --> PARENT MODULE --> EXPERIMENT**

<table>
<thead>
<tr>
<th>NO.</th>
<th>LVL</th>
<th>STATEMENT</th>
<th>DDPATHS GENERATED</th>
<th>COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>IF LIMIT 20 100</td>
<td><strong>DD-PATH 1</strong> IS PROCEDURE ENTRY</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td><strong>DD-PATH 2</strong> IS TRUE BRANCH</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td><strong>DD-PATH 3</strong> IS FALSE BRANCH</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td><strong>DD-PATH 4</strong> IS TRUE BRANCH</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td><strong>DD-PATH 5</strong> IS FALSE BRANCH</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td><strong>DD-PATH 6</strong> IS TRUE BRANCH</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td><strong>DD-PATH 8</strong> IS SWITCH OUTWAY</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td><strong>DD-PATH 9</strong> IS SWITCH OUTWAY</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td><strong>DD-PATH 10</strong> IS SWITCH OUTWAY</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td><strong>DD-PATH 11</strong> IS SWITCH OUTWAY</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td><strong>DD-PATH 12</strong> IS TRUE BRANCH</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td><strong>DD-PATH 13</strong> IS FALSE BRANCH</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td><strong>DD-PATH 14</strong> IS TRUE BRANCH</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td><strong>DD-PATH 15</strong> IS FALSE BRANCH</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td><strong>DD-PATH 16</strong> IS TRUE BRANCH</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td><strong>DD-PATH 17</strong> IS LOOP ON FOR AGAIN</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>2</td>
<td><strong>DD-PATH 18</strong> IS ESCAPE FOR LOOP</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>2</td>
<td><strong>DD-PATH 19</strong> IS LOOP ON FOR AGAIN</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>2</td>
<td><strong>DD-PATH 20</strong> IS ESCAPE FOR LOOP</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**TOTAL DDPATHS**: 20  
**DDPATHS EXECUTED**: 8  
**PERCENT EXECUTED**: 45
**Example Command Sets**

(1) OLD LIBRARY = REFMAN.
START.
ANALYZER, ALL MODULES.
ANALYZER, DDPATHS.
ANALYZER, MOLST.
ANALYZER.
END.

This command set will produce only the two cumulative coverage reports, DD-path coverage and statement coverage, from the set of test cases on the trace file.

(2) OLD LIBRARY = REFMAN.
START.
ANALYZER, MODULE = EXAMPL1.
ANALYZER, DDPATHS.
ANALYZER, NOTHIT.
ANALYZER.
END.

This command set will produce the DD-path cumulative coverage report and the test-by-test report of DD-paths not executed for module EXAMPL1.
This command causes a printout of the Test Execution trace data at the DD-path level with the module invocation and return trace data interspersed. The trace data must have been recorded at the DD-path level; i.e., the TFLAG parameter in the PROBI call must be 3. Each line of output represents a module invocation or return, the start of an execution, or the beginning or end of a test case or test set. Each DD-path line shows the DD-path number, its description and outcome, and the relevant source text. (The FOR statement is considered to be the relevant statement in iteration.)

Rule

For a single ANALYZER command, either option DDTRACE or MODTRACE can be used, but not both.

Sample Output Excerpt

```plaintext
+----------+----------------+---------+---------
| TEST-CASE | SAMPLE 2 2000-18/SW/95 | DD-PATH | TRUE BRANCH |
| ** INVOKE MODUL 15 ENTRY (EXPRESS ) | | DD-PATH 1 | PROCEDURE ENTRY |
| STMT 1 | IF (EXPRESS ) | | DD-PATH 3 | FALSE BRANCH |
| STMT 9 | FOR LIMIT 3 | | DD-PATH 4 | TRUE BRANCH |
| STMT 12 | TRUE BRANCH | | DD-PATH 5 | FALSE BRANCH |
| ** RETURN FROM MODUL 15 ENTRY (EXPRESS ) | | DD-PATH 6 | TRUE BRANCH |
| STMT 21 | IF (EXPRESS ) | | DD-PATH 16 | ESCAPE FOR LOOP |
| STMT 18 | FOR LIMIT 1 | | DD-PATH 20 | ESCAPE FOR LOOP |
| STMT 19 | FOR LIMIT 1 | | DD-PATH 7 | ESCAPE FOR LOOP |
| STMT 20 | IF (EXPRESS ) | | DD-PATH 8 | ESCAPE FOR LOOP |
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
ANALYZER, ALL MODULES.
ANALYZER, DDTRACE.
ANALYZER, DDPATHS.
ANALYZER.
END.
```

This command set will produce the DD-path trace report and the cumulative DD-path coverage report for all modules on the library.

5-12
ANALYZER,FACTOR = <percent>.

Description

This command is used in conjunction with the ANALYZER,TIME option. It allows the user to establish a value for the estimated CPU time overhead of executing the inserted probe statements in instrumented modules.

In the report produced by the command

ANALYZER,TIME.

both the actual time including the inserted probe statements and the estimated time using the <percent> specified is shown. Absence of the ANALYZER,FACTOR option command indicates no overhead in the time; i.e., the default is 0 percent.

Example Command Sets

(1) OLD LIBRARY = REFSAM.
    START.
    ANALYZER,FACTOR = 20.
    ANALYZER,TIME.
    ANALYZER,ALL.
    ANALYZER,ALL MODULES.
    ANALYZER.
    END.

(2) OLD LIBRARY = REFSAM.
    START.
    ANALYZER,ALL MODULES.
    ANALYZER,FACTOR = 10.
    ANALYZER,TIME.
    ANALYZER.
    END.
ANALYZER,HIT.

Description

This command causes ANALYZER to produce a list of DD-paths for each module and the number of times each path has been traversed. Two such lists are made: one for each test case and one for all test cases analyzed. Only the modules specified by an ANALYZER module selection option and encountered on the AUDIT file will be represented in this report.

Sample Output

DD-paths traversed in selected modules for test-case sample 2 2920 28/04/79

Module JASTEST DD-paths (traversals)

<table>
<thead>
<tr>
<th>EXRRQNS</th>
<th>EXRSPNS</th>
<th>11</th>
<th>71</th>
<th>21</th>
<th>31</th>
<th>41</th>
<th>51</th>
<th>61</th>
<th>71</th>
<th>81</th>
<th>91</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>Example</td>
<td>11</td>
<td>11</td>
<td>21</td>
<td>31</td>
<td>41</td>
<td>51</td>
<td>61</td>
<td>71</td>
<td>81</td>
<td>91</td>
</tr>
</tbody>
</table>

Single Test Case

DD-paths traversed in selected modules for all test-cases

Module JASTEST DD-paths (traversals)

<table>
<thead>
<tr>
<th>EXRRQNS</th>
<th>EXRSPNS</th>
<th>11</th>
<th>71</th>
<th>21</th>
<th>31</th>
<th>41</th>
<th>51</th>
<th>61</th>
<th>71</th>
<th>81</th>
<th>91</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>Example</td>
<td>11</td>
<td>11</td>
<td>21</td>
<td>31</td>
<td>41</td>
<td>51</td>
<td>61</td>
<td>71</td>
<td>81</td>
<td>91</td>
</tr>
</tbody>
</table>

Multiple Test Cases

Example Command Set

OLD LIBRARY = REFMAN.
START.
ANALYZER, ALL MODULES.
ANALYZER,HIT.
ANALYZER.
END.
Description

This command causes ANALYZER to print a source listing of each specified module which shows the number of times each executable statement was exercised for all testcases. The DD-paths are listed at the right of the statement of origin. Next is a count of the times each executable statement was exercised; executable statements not exercised are flagged with an asterisk. To the far right are control indicators for statement types which affect normal processing sequence. Statement coverage is summarized at the end of the module listing. If a specified module is not invoked during the test, this report is suppressed.

Sample Output

```
MODULE STATEMENT LISTING

MODULE <EXPROH >. JAVTEST .EXPROH>. PARENT MODULE <EXPROH>.

<table>
<thead>
<tr>
<th>NO.</th>
<th>LVL</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>#. JAVTEST. EXPROH. COMPILE 1 EXECUTE #.</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>STMT 1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>STMT 2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>STMT 3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>STMT 4</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>STMT 5</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>STMT 6</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>STMT 7</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>STMT 8</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>STMT 9</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>STMT 10</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>STMT 11</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>STMT 12</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>STMT 13</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>STMT 14</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>STMT 15</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>STMT 16</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>STMT 17</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>STMT 18</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>STMT 19</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>STMT 20</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>STMT 21</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>STMT 22</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>STMT 23</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>STMT 24</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>STMT 25</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>STMT 26</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>STMT 27</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>STMT 28</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>STMT 29</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>STMT 30</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>STMT 31</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td>STMT 32</td>
</tr>
</tbody>
</table>

EXECUTABLE STATEMENTS 14
STATEMENTS EXECUTED 14
PERCENT EXECUTED 100
```

BEST AVAILABLE COPY

5-15
Example Command Set

OLD LIBRARY = REFMAN.
ANALYZER, MODULE = EXPROGM, EXMPI.
ANALYZER, MODLST.
ANALYZER.
END.
ANALYZER, MODTRACE.

Description

This command causes a printout of the Test Execution trace data at the module invocation and return level. If the trace data was not recorded on the file during Test Execution (see Sec. 2.6.2.2), no report is produced. Each line of output represents either a module invocation or the beginning or end of a test case or test set. The level of invocation for the modules being traced is shown to the left; the module invocations are indented to show nesting level.

Rule

For a single execution of ANALYZER, either option MODTRACE or DDPTRACE can be used, but not both.

Sample Output

<table>
<thead>
<tr>
<th>TEST-CASE</th>
<th>SAMPLE</th>
<th>1044</th>
<th>08/09/74</th>
</tr>
</thead>
<tbody>
<tr>
<td>I II</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXAMPLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAMPLE 3</td>
<td></td>
<td>1044</td>
<td>08/09/74</td>
</tr>
<tr>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXAMPLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAMPLE 4</td>
<td></td>
<td>1044</td>
<td>08/09/74</td>
</tr>
<tr>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXAMPLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST-SET TERMINATION</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sample Command Sets

(1) OLD LIBRARY = REFMAN.
START.
ANALYZER, MODTRACE.
ANALYZER.
END.

(2) OLD LIBRARY = REFMAN.
START.
ANALYZER, MOD TRACE.
ANALYZER, ALL MODULES.
ANALYZER, SUMMARY.
ANALYZER.
END.
The module specification commands are used to identify the modules on LIBOLD for which ANALYZER processes the summary data on the Test Execution trace file. The default is no modules specified. To specify all known modules use the command:

```
ANALYZER, ALL MODULES.
```

To specify a subset of modules use the command:

```
ANALYZER, MODULE = <name-1>, <name-2>, ..., <name-n>.
```

The following ANALYZER options require module specification: SUMMARY, HIT, NOTHIT, TIME, ALL.

**Rule**

As many as 24 modules can be specified using this command.

**Example Command Sets**

1. OLD LIBRARY = REFTST.
   START.
   ANALYZER, MODULE = EXPROC, EXMPL1.
   ANALYZER, ALL.
   ANALYZER.
   END.

2. OLD LIBRARY = REFMAN.
   START.
   ANALYZER, FACTOR = 20.
   ANALYZER, TIME.
   ANALYZER, MODULE = EXMPL1.
   ANALYZER.
   END.
ANALYZER, NOTHIT.

Description

This command causes ANALYZER to list any DD-path which was not traversed in each test case and all test cases for the specified modules. If a module was not invoked during any test cases, not probed during INSTRUMENT, or not specified for ANALYZER processing, it will not be referenced in this report.

Sample Output

Example Command Sets

(1) OLD LIBRARY = REFMAN.
    START.
    ANALYZER, MODULE = EXPROGM, EXMPL1.
    ANALYZER, NOTHIT.
    ANALYZER.
    END.

(2) OLD LIBRARY = REFMAN.
    START.
    ANALYZER, ALL MODULES.
    ANALYZER, SUMMARY.
    ANALYZER, NOTHIT.
    FOR LIBRARY.
    PRINT, DDP.
    END FOR.
    END.

This command set will provide the post-test analysis coverage summary, the list of DD-paths not executed in any of the test cases, and the list of statements on each DD-path for all modules in the library.
Description

When this option is specified, a report is produced which summarizes the testing coverage of each of the specified modules for each test case encountered on the saved trace file and for the cumulative of all tests completed in the run. Included in the report is the list of modules selected but not invoked during any of the tests in the run. The cumulative coverage shown on the last line is adjusted to reflect all selected modules.

Sample Output Excerpt

<table>
<thead>
<tr>
<th>MODULE</th>
<th>COVERED</th>
<th>% COVERED</th>
<th>TEST CASE(S)</th>
<th>DATE</th>
<th>COVERED</th>
<th>% COVERED</th>
<th>COVERED</th>
<th>% COVERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>100</td>
<td></td>
<td>04/29/78</td>
<td>2</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>80</td>
<td>80</td>
<td></td>
<td>04/29/78</td>
<td>3</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>50</td>
<td>50</td>
<td></td>
<td>04/29/78</td>
<td>4</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>20</td>
<td>20</td>
<td></td>
<td>04/29/78</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>10</td>
<td></td>
<td>04/29/78</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The cumulative coverage shown on the last line is adjusted to reflect all selected modules.

5-20
Example Command Sets

(1) OLD LIBRARY = REFMAN.
    START.
    ANALYZER, ALL MODULES.
    ANALYZER, SUMMARY.
    ANALYZER.
    END.

(2) OLD LIBRARY = REFMAN.
    START.
    ANALYZER, MODULE = EXPROGM, EXMPL1.
    ANALYZER, SUMMARY.
    ANALYZER.
    END.
This command causes ANALYZER to print a summary of the time spent in execution within the specified modules for all tests. Only complete invocation-return cycles are included in the time reported (i.e., the increment of time added for each cycle to the cumulative execution time for a module is the time interval logged by the system clock between the invocation of the module and the next return from the module, excluding time spent in the data collection routines).

The TIME report lists all specified modules, even though some modules may not have been invoked or instrumented (denoted by NOT TRACED).

Sample Output

<table>
<thead>
<tr>
<th>MODULE NAME</th>
<th>EXECUTION TIME</th>
<th>ESTIMATED TIME</th>
<th>CYCLES</th>
<th>INVOCATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP40PL</td>
<td>171</td>
<td>143</td>
<td>1</td>
<td>NOT TRACED</td>
</tr>
<tr>
<td>EXP40PM</td>
<td>26</td>
<td>22</td>
<td>2</td>
<td>NOT TRACED</td>
</tr>
</tbody>
</table>

In this sample output, the estimated time reflects an ANALYZER, FACTOR = 20. command accompanying the TIME report command. In this case 20% of the actual execution time is subtracted from the values listed under execution time and these reduced values are listed as the estimated execution time. The execution time spent in EXPROOM (171 ms) includes the 26 ms spent in EXMPL1, since EXMPL1 is nested within EXPROOM.

Example Command Sets

(1) OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,FACTOR = 20.
ANALYZER,TIME.
ANALYZER,SUMMARY.
ANALYZER.
END.

(2) OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,TIME.
ANALYZER,CASES = 75.
ANALYZER.
END.
In the second command set, no attempt was made to estimate the time spent executing the probe invocation statements; thus the execution and estimated time value will be identical.
ASSIST,CROSSREF,JAVSTEXT = <name>,...

Description

This command produces a cross-reference listing of items, files, switch names, labels and subprogram names for all the specified JAVSTEXTs. The form of the command is:

```
ASSIST,CROSSREF,JAVSTEXT = <name-1>,<name-2>,...,<name-n>.
```

The output for this command is similar to that for the library cross-reference, except that only symbols from the specified JAVSTEXTs are included.

Rules

1. A maximum of 22 JAVSTEXTs can be specified with this command.
2. A maximum of 125 modules can be used for cross reference mapping.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
ASSIST,CROSSREF,JAVSTEXT = EXCOMPL,EXPROC.
END.
```
ASSIST,CROSSREF,LIBRARY.

Description

The outcome of this command is a cross-reference listing of names in the entire library and their usage. The names listed are all items, files, switch names, labels, and subprogram names.

Rules

1. A maximum of 100 JAVSTEXTs can be used in cross reference mapping.
2. A maximum of 125 modules can be used in cross reference mapping.

Sample Output Excerpt

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MODULE</th>
<th>USED/SET/DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>EXPROG</td>
<td>180 28</td>
</tr>
<tr>
<td>CARO</td>
<td>EXPROG</td>
<td>120 18* 21 22 23</td>
</tr>
<tr>
<td>EXPL1</td>
<td>EXPL1</td>
<td>1</td>
</tr>
<tr>
<td>EXPL2</td>
<td>EXPL2</td>
<td>2</td>
</tr>
<tr>
<td>EXPL3</td>
<td>EXPL3</td>
<td>2</td>
</tr>
<tr>
<td>FILL</td>
<td>EXPL1</td>
<td>1</td>
</tr>
<tr>
<td>I0</td>
<td>EXPL1</td>
<td>40 15*</td>
</tr>
<tr>
<td>INDEX</td>
<td>EXPL1</td>
<td>70 22* 24* 28</td>
</tr>
<tr>
<td>ITER1</td>
<td>EXPL1</td>
<td>60 16 15 22</td>
</tr>
<tr>
<td>ITER2</td>
<td>EXPL2</td>
<td>44</td>
</tr>
<tr>
<td>ITER3</td>
<td>EXPL2</td>
<td>60 100 24 26</td>
</tr>
<tr>
<td>ITER4</td>
<td>EXPL3</td>
<td>40 33 15 23</td>
</tr>
<tr>
<td>LABEL1</td>
<td>EXPL1</td>
<td>27 27 27 300</td>
</tr>
<tr>
<td>LABEL2</td>
<td>EXPL1</td>
<td>27 290</td>
</tr>
<tr>
<td>LIMIT1</td>
<td>EXPL1</td>
<td>1</td>
</tr>
<tr>
<td>LIMIT2</td>
<td>EXPL1</td>
<td>1</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>EXCOMP</td>
<td>1</td>
</tr>
<tr>
<td>MSG01</td>
<td>EXCOMP</td>
<td>1</td>
</tr>
<tr>
<td>MSG02</td>
<td>EXCOMP</td>
<td>1</td>
</tr>
<tr>
<td>MSG03</td>
<td>EXCOMP</td>
<td>1</td>
</tr>
</tbody>
</table>

Example Command Set

OLD LIBRARY = REFMAN.
START.
ASSIST,CROSSREF,LIBRARY.
END.

5-25
**Description**

There are two forms of this command:

ASSIST,PICTURE. or
ASSIST,PICTURE,<option—l>,<option—2>.

where option—l is CONTROL and option—2 is NOSWITCH.

This command produces, for the specified module, a stylized picture of
the pattern of program flow implicit in the set of sequence of DD—paths. The
available options are CONTROL and NOSWITCH. If CONTROL is present, the pic-
ture includes all control statements (e.g., GOTO, invocation) on the DD—path.
If NOSWITCH is present, the picture omits the flows emanating from GOTO-switch
statements.

Each line in the picture represents a statement at the beginning of a
DD—path or a control statement; the statement type* and number are displayed
in the center. Each DD—path is represented by either a B + E sequence or an
S. The B + E sequence shows where the DD—path begins and ends; the S indi-
cates that the DD—path begins and ends on the same statement. The statements
shown are those which are "necessary" to display all possible program flow
and, by convention, are arranged in statement order. All "forward flow"
(with respect to the statement ordering) is shown by convention on the right-
hand side of the picture; and all "reverse flow" is shown in the left-hand
side. For those statements which are at the beginning of a DD—path, the num-
er of the DD—path is specified on the far right margin. The DD—path numbers
are ordered left to right. Control statements are indicated by a C.

For very complex modules (e.g., switches with a large number of targets)
the NOSWITCH format limits the number of paths which can be displayed. If the
picture overflows on the right or left side, an informative message is printed
and processing continues.

*See Appendix C for statement type abbreviations.
ASSIST, PICTURE. (Cont.)

Sample Output

PICTURE WITH ALL DO-PATHS...
MODULE <EXMPL1 > JAVSTEXT <EXPROM > PARENT MODULE <EXPROM >

<table>
<thead>
<tr>
<th>STAT</th>
<th>TYPE</th>
<th>NO.</th>
<th>DO-PATH NUMBERS...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Example Command Sets

1. OLD LIBRARY = REFMAN.
   START.
   MODULE = EXMPL1.
   ASSIST, PICTURE, CONTROL.
   END.

   This command set produced the sample output.

2. OLD LIBRARY = REFMAN.
   START.
   FOR LIBRARY.
   ASSIST, PICTURE.
   END FOR.
   END.

   This command set would produce a picture for each module on the library
   similar to the one in the sample output, but without the control statements
   (CTCL 29 in the sample output) displayed.

3. OLD LIBRARY = REFMAN.
   START.
   JAVSTEXT = EXPROM.
   FOR JAVSTEXT.
   ASSIST, PICTURE, CONTROL, NOSWITCH.
   END FOR.
   END.

5-27
ASSIST,PICTURE. (Cont.)

In addition to the control statements’ being displayed, this command set will omit the flows and DD-path numbers emanating from GOTO-switch statements (GTSW 28, DD-paths 7 to 11 in the sample output).

(4) OLD LIBRARY = REFMAN.
START.
MODULE = EX1PL1.
ASSIST,PICTURE,NOSWITCH.
END.

The picture options need not be in order.
ASSIST, REACHING SET,<specs>.

Description

This command is used to analyze the flow required to reach a particular DD-path according to the items in the specification list. The reaching set consists of all DD-paths which flow between the beginning and ending DD-paths. There are three types of specifications (which may be present in any order but separated by commas):

<number-to>,<number-from>

ITERATIVE

PICTURE

The DD-path for the reaching set (i.e., the target of flow) is named by the (required) specification <number-to>. In the absence of a <number-from> specification, the flow is assumed to start with the first executable statement. The <number-from> specification allows the user to designate the DD-path where flow starts. The analysis begins with the first executable statement on the DD-path.

The (optional) ITERATIVE specification allows the user to control the set of DD-paths in the analysis. If ITERATIVE is not specified, all flows which include iteration are suppressed in determination of paths of control. If ITERATIVE is specified, the flows include iteration.

The (optional) PICTURE specification allows the user to control the additional reports presented. If PICTURE is not present, the output is limited to the reaching set definition. The additional picture report shown in the sample output is included as a result of the command ASSIST, REACHING SET, 17, 4, PICTURE.

Rules

(1) Maximum of 100 DD-paths per reaching set path.
(2) Maximum of 100 outways per decision.
(3) Maximum of 1200 DD-paths per analyzed module for reaching set.
(4) Maximum of 2400 statements per analyzed module for reaching set.
(5) Maximum of 200 statements in reaching set.

5-29
ASSIST, REACHING SET,<specs>. (Cont.)

Sample Output

```

ASSIST, REACHING SET,<specs>. (Cont.)

Sample Output

```

```

ASSIST, REACHING SET,<specs>. (Cont.)

Sample Output

```

```
Example Command Sets

(1) OLD LIBRARY = REFMAN.
START.
JAVA TEXT = EXPROGM.
MODULE = EXMPL1.
ASSIST, REACHING SET, 16, ITERATIVE, PICTURE.
END.

This command set will produce the reaching set definition and reaching set picture reports containing all DD-paths, including iteration, which flow between DD-path 1 and DD-path 16 of module EXMPL1.

(2) OLD LIBRARY = REFMAN.
START.
JAVA TEXT = EXPROGM.
MODULE = EXMPL1.
ASSIST, REACHING SET, 17, 4.
END.

This command set produces the first report shown in the sample output.
Description

This command produces a report describing the DD-path membership for each executable statement in the "current module." The statements are listed in statement order with their type, the DD-path numbers which begin on the statement, and the complete set of DD-paths which contain the statement. A DD-path beginning or ending is represented by a B or E.

Sample Output

<table>
<thead>
<tr>
<th>STATEMENT/DDPATH LISTING</th>
<th>MODULE &lt;EXMPL&gt; &gt; JAVSTEXT &lt;EXPROG&gt; &gt; PARENT MODULE &lt;EXPROG&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAT</td>
<td>TYPE</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>PROC</td>
</tr>
<tr>
<td>2</td>
<td>BEGIN</td>
</tr>
<tr>
<td>3</td>
<td>IF</td>
</tr>
<tr>
<td>4</td>
<td>END</td>
</tr>
<tr>
<td>5</td>
<td>ASMT</td>
</tr>
<tr>
<td>6</td>
<td>FOR</td>
</tr>
<tr>
<td>7</td>
<td>END</td>
</tr>
<tr>
<td>8</td>
<td>BEGIN</td>
</tr>
<tr>
<td>9</td>
<td>ASMT</td>
</tr>
<tr>
<td>10</td>
<td>IF</td>
</tr>
<tr>
<td>11</td>
<td>END</td>
</tr>
<tr>
<td>12</td>
<td>BEGIN</td>
</tr>
<tr>
<td>13</td>
<td>ASMT</td>
</tr>
<tr>
<td>14</td>
<td>IF</td>
</tr>
<tr>
<td>15</td>
<td>END</td>
</tr>
<tr>
<td>16</td>
<td>BEGIN</td>
</tr>
<tr>
<td>17</td>
<td>ASMT</td>
</tr>
<tr>
<td>18</td>
<td>IF</td>
</tr>
<tr>
<td>19</td>
<td>END</td>
</tr>
<tr>
<td>20</td>
<td>BEGIN</td>
</tr>
<tr>
<td>21</td>
<td>ASMT</td>
</tr>
<tr>
<td>22</td>
<td>BEGIN</td>
</tr>
<tr>
<td>23</td>
<td>IF</td>
</tr>
<tr>
<td>24</td>
<td>END</td>
</tr>
<tr>
<td>25</td>
<td>BEGIN</td>
</tr>
<tr>
<td>26</td>
<td>IF</td>
</tr>
<tr>
<td>27</td>
<td>END</td>
</tr>
<tr>
<td>28</td>
<td>BEGIN</td>
</tr>
<tr>
<td>29</td>
<td>IF</td>
</tr>
<tr>
<td>30</td>
<td>END</td>
</tr>
<tr>
<td>31</td>
<td>BEGIN</td>
</tr>
<tr>
<td>32</td>
<td>IF</td>
</tr>
<tr>
<td>33</td>
<td>END</td>
</tr>
<tr>
<td>34</td>
<td>BEGIN</td>
</tr>
<tr>
<td>35</td>
<td>IF</td>
</tr>
<tr>
<td>36</td>
<td>END</td>
</tr>
<tr>
<td>37</td>
<td>BEGIN</td>
</tr>
<tr>
<td>38</td>
<td>IF</td>
</tr>
<tr>
<td>39</td>
<td>END</td>
</tr>
<tr>
<td>40</td>
<td>BEGIN</td>
</tr>
<tr>
<td>41</td>
<td>IF</td>
</tr>
<tr>
<td>42</td>
<td>END</td>
</tr>
<tr>
<td>43</td>
<td>BEGIN</td>
</tr>
<tr>
<td>44</td>
<td>IF</td>
</tr>
</tbody>
</table>

Example Command Set

OLD LIBRARY = REPMAN.
START.
JAVSTEXT = EXPROG.
MODULE = EXMPL.
ASSIST,STATEMENTS.
END.

5-32
Description

The BASIC command causes BASIC syntax analysis to be executed. The actions performed by the BASIC command are:

1. To build LIBNEW containing a module descriptor block, a statement descriptor table, a statement table, a symbol locator table, and a symbol table for each module in each START-TERM sequence on the INPUT file.

2. To produce a report indicating the source text of each module, the module name, and a count of the syntax and semantic errors detected in the source for each START-TERM sequence.

A sample report is shown below. This report contains the following items:

- A listing of the original source text with the errors encountered in BASIC analysis interspersed.
- Summary information at the end of each module and each START-TERM sequence.

Rules

(1) See BASIC constraints in Sec. 3.2.

(2) Maximum of 250 modules can be on LIBNEW.

(3) Maximum of 150 modules in any JAVSTEXT.
Sample Output

```
##.JAVESTRL EXCOMPL PRESET##
STARTS
##.COMPOOL EXAMPLE CONTAINING PRESET OUTPUT MESSAGES ##
COMMON MESSAGES
BEGIN
ITEM MSG 1 M 16 P 18H1 JAVR TEST CASE 155
ITEM MSG 1 M 16 P 18H1 RESULT 1 T A 155
ITEM MSG 2 M 16 P 18H1 RESULT 2 B A 155
ITEM MSG 3 M 16 P 18H1 RESULT 3 C A 155
ITEM MESSAGE M 150
END

EXCOMPL (EXCOMPL) COMPLETED
********** NO ERRORS HERE FOUND BY JAVS-2 **********

##.JAVESTRL EXPROGRAM COMPUTE (EXCOMPL)##
STARTS
##.JAVESTRL SIMPLE TEST PROGRAM ##
DEFINE INT #1 = 4 #55
DEFINE INT #2 = 4 #95
ITEM ID #1 = 4
ITEM II = #7 #80
ITEM iterative = ITERA #7 #80
ITEM iterative = ITERA #80
ITEM iterative = ITERA #80
ITEM CARDS = 4 4
FILE READER = #9 A VCKI VEOF TAPES
FILE PRINTER = #9 A VCKI VEOF TAPES
MONITOR ID, iterative, iterative
MESSAGE = #9S15
OUTPUT PRINTER MESSAGE 45
B involvement READER CARDS
IF READER 4 A VEOF 4
BEGIN
BYTE $0,-byte
ID = BYTE $0,-byte
BYTE $0,-byte
ITERA = BYTE $0,-byte
BYTE $0,-byte
ITERA = BYTE $0,-byte
CLOE EXPLOR
BEGIN
ITERA = 15
ITERA = 15
END ##.EXPLOR##

EXPLOR (EXPROGRAM) COMPLETED
```

Example Command Sets

(1) CREATE LIBRARY = REFMAN.
START.
BASIC.
END.

This command set will cause library REFMAN to be built containing the syntax analysis for the input source text. All BASIC default options are used in this command set, and the sample output will be produced.
(2) CREATE LIBRARY = REPMAN.
START.
BASIC,CARD IMAGES = OFF.
BASIC.
FOR LIBRARY.
PRINT,MODULE.
END FOR.
END.

This command set will perform syntax analysis on the input source text, create the library, print an abbreviated report, and then print the reformatted listing for each module on the newly created library.
BASIC,CARD IMAGES = <option>.

Description

BASIC,CARD IMAGES = ON/OFF. (DEFAULT = ON.)

This command allows user control over the inclusion of the source text listing in the report produced by BASIC processing. The effect of the option command

BASIC,CARD IMAGES = OFF.

on the BASIC output is shown below; only summary information is provided.

Sample Output

EXCOMPL1 (EXCOMPL1) COMPLETED
********* NO ERRORS WERE FOUND BY JAVS-2 *********
EXMPL2 (EXPROGM) COMPLETED
EXMPL3 (EXPROGM) COMPLETED
EXMPL1 (EXPROGM) COMPLETED
EXPROGM (EXPROGM) COMPLETED
********* NO ERRORS WERE FOUND BY JAVS-2 *********

Example Command Set

CREATE LIBRARY = REFMAN.
START.
BASIC,CARD IMAGES = OFF.
BASIC.
END.
BASIC,COMMENTS = <option>.

Description

BASIC,COMMENTS = ON/OFF. (DEFAULT = ON.)

This command allows user control over the treatment of comments in the module source text. The default action is to include all comments in the library. Comments may be excluded from the statement text table of all modules by the command

BASIC,COMMENTS = OFF.

If comments are included, more space is required on the library and all subsequent processing of the statement table requires more computer time. Comments should be excluded from the library if they are imbedded within JOVIAL statements.

Sample Command Set

CREATE LIBRARY = REFMAN.
START.
BASIC,COMMENTS = OFF.
BASIC.
END.
BASIC,DEFINES = <option>.

Description

BASIC,DEFINES = ON/OFF. (DEFAULT = ON.)

This command allows the user control over BASIC usage of DEFINE variables. The default option is to replace the occurrence of each DEFINE variable with its definition just as the JOVIAL compiler does. Subsequent BASIC processing utilizes the results of that definition both in statement analysis and in the text stored for each statement on the library. It is essential that the default option be used if the modules are to be processed by any other JAVS standard processing step.

Replacement of DEFINE variables by their definition may be suppressed with the command:

BASIC,DEFINES = OFF.

This command is normally used only when the following commands are also used:

BASIC,ERRORS = OFF.

BASIC,TEXT = JAVSTEXT.

Text so processed may be written to LPRINT in the structured format with the command:

PUNCH,JAVSTEXT = <text-name>.

This is useful in using JAVS to punch a copy of reformatted source text. To get a reformatted listing of the source text, the PRINT,JAVSTEXT command can be used.
BASIC,ERRORS = <option>.

Description

BASIC,ERRORS = ON/OFF/LIMIT/TRACE. (DEFAULT = ON.)

This command allows the user control over the syntax and semantic errors detected in the JOVIAL source by BASIC. The default option prints a numbered error message as each error is detected.* These error messages are interspersed in the list of source text lines (CARD IMAGES = ON.). Since BASIC processing scans the JOVIAL source text only once, not all source errors are detected.

All error messages may be suppressed by the command

BASIC,ERRORS = OFF.

The number of errors may be limited by the command

BASIC,ERRORS = LIMIT.

For this option, when the number of errors exceeds 100, BASIC terminates processing immediately and a fatal JAVS error results.

In addition to the error message itself, the origin of the error in the JAVS code can be displayed with the command

BASIC,ERRORS = TRACE.

This is useful in JAVS system maintenance.

Sample Command Set

CREATE LIBRARY = REFMAN.
START.
BASIC,CARD IMAGES = OFF.
BASIC,ERRORS = OFF.
BASIC.
END.

*See Appendix B for a description of BASIC error messages.
BASIC,SYMBOLS = <option>.

Description

BASIC,SYMBOLS = ON/OFF/PARTIAL. (DEFAULT = PARTIAL.)

This command allows the user control over the symbols entered in the
detailed symbol table. The default action is to enter only those symbols
essential to other JAVIS processing steps. These symbols, called control sym-
bols, are the names of modules, statement labels, and switches.

Other JOVIAL identifiers and constants may be included in the symbol
table with the command

BASIC,SYMBOLS = ON.

BASIC performs a limited analysis on the properties of each symbol and stores
type-dependent information about each in the symbol table. More space is
required on the library for the additional symbol table entries and all sub-
sequent processing of the symbol table requires more computer time.

The symbol table may be omitted from the library with the command

BASIC,SYMBOLS = OFF.

This option is meaningful only if the library is never processed by
STRUCTURAL and subsequent JAVIS processing steps (e.g., when just BASIC pro-
cessing is used to reformat the source text).

Sample Command Set

CREATE LIBRARY = REFMAN.
START.
BASIC,SYMBOLS = ON.
BASIC.
FOR LIBRARY.
PRINT,STB.
END FOR.
END.

This command set will put all symbols on the newly created library and
then print the symbol table for each module.
**Description**

BASIC,TEXT = COMPUTE/BOTH/PRESET/JAVSTEXT. (DEFAULT = COMPUTE.)

This command allows the user control over the type of analysis performed by BASIC on the sets of START-TERM text appearing on the file INPUT. BASIC processes a sequence of START-TERM texts until an end-of-file is detected on READER. Three types of text sequences are permitted:

1. One or more START-TERM texts of executable (COMPUTE) source.
2. A single COMPOOL START-TERM text followed by one or more START-TERM texts of executable (COMPUTE) source.
3. One or more COMPOOL START-TERM texts.

The default action is to process type 1. The text presented under the COMPUTE option is assumed to be a sequence of START-TERM texts which are executable. Any particular START-TERM text may normally require a COMPOOL for proper compilation; BASIC will properly process the text in the absence of its COMPOOL under the COMPUTE option provided a JAVSTEXT directive (Sec. 1.4) which includes the related-text-name parameter appears at the beginning of the START-TERM text.

Type 2 text sequences (one COMPOOL followed by one or more executable texts) are processed with the command:

```
BASIC,TEXT = BOTH.
```

With this option, when LIBOLD is used, the COMPOOL must be included on READER, even if no change has been made to its source.

Type 3 text sequences (one or more COMPOOL texts) are processed with the command:

```
BASIC,TEXT = PRESET.
```

BASIC will not separate source text into modules if the following command is used:

```
BASIC,TEXT = JAVSTEXT.
```

Since the unit of processing for most other JAVS processing steps is an executable module, this command prohibits performing module analyses on the text. (See BASIC,DEFINES = command description for usage.) All errors normally detected by BASIC are ignored with this option.

*See Table 1.1 on page 1-4.*
Example Command Set

CREATE LIBRARY = REFMAN.
START.
BASIC,TEXT = BOTH.
BASIC.
END.

This command set will build a library containing a single COMPOOL and one or more START-TERM texts.
BUILD LIBRARY.*

Description

This macro command has two forms:

BUILD LIBRARY. or

BUILD LIBRARY = <name>.

In the first command form, the library name given to LIBNEW is TEST. In the second form, the user specifies the library name. Both command forms generate the following JAVS standard commands:

CREATE LIBRARY = <name>. (or TEST)
START.
BASIC,COMMANDS = OFF.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.

The macro command processor generates the JAVS "END" command if it is not present as the last command.

Rules

(1) See BASIC and STRUCTURAL constraints in Sec. 3.2 and 3.3.

(2) The library name should be 8 or less characters.

(3) This macro command should not be used if the source program includes a COMPOOL (constraint 2 on page 3-3).

Example Command Sets

(1) BUILD LIBRARY = REFMAN.
DOCUMENT.

This command set will create a new library, perform syntax and structural analyses and produce a series of JAVS reports for all modules in the JOVIAL source program. These reports are useful for documentation, maintenance, testing and retesting the source program.

(2) BUILD LIBRARY.
PROBE,JAVSTEXT = EXPROM.
PRINT,JAVSTEXT = EXPROM,INSTRUMENTED = ALL.

*Can be used only with the overlay version of JAVS.
This command set shows a mixture of JAVS macro and standard commands. The first macro command will create a new library called TEST and perform syntax and structural analyses on all modules in the JOVIAL source program. The PROBE macro command will instrument JAVSTEXT EXPROCM, using the default instrumentation options, and write the instrumented text onto file LPRINT. The last command will print the instrumented text.
CREATE LIBRARY = <libname>.

Description

This library command specifies that LIBNEW (a "new library") will be created during the current run, as is normally the case in BASIC. When this option is specified, LIBNEW is assumed to be empty at the beginning of a run. LIBNEW is a read/write library.

Rule

The library name should be 8 or less characters.

Example Command Set

CREATE LIBRARY = REFMAN.
START.
BASIC.
END.
Description

This command has two forms:

DEPENDEENCE, BANDS. or

DEPENDEENCE, BANDS = <number>. (DEFAULT = 2.)

The outcome of this command is a "snapshot" of the position of the selected module within the inter-module hierarchy. The sample output shows the BANDS report for the default width. To the left of the selected module is shown the structure of the calls to the module; to the right of the selected module is shown the invocation structure emanating from the module. The number of bands is the width (in each direction) of the structure displayed. Up to five bands may be displayed on this report. This report is useful in determining the extent of inter-module dependence to several levels. Modules which are called from only one other module are potential candidates to head a link for overlay purposes.

The modules listed under column -1 call the selected module directly (i.e., GTCR and OLOOK). The modules listed under column -2 call those in column -1. For example, GTCR is called by CK2K, LLK, and OLOOK.

The modules listed under column 1 are called by the selected module (e.g., BTOH, ERRXR, GTCARD, etc.). The modules listed under column 2 are called by those listed under column 1. For example, ERRXR calls BTOD, ERROR, OPUT, and SYSTEMP.

Rule

Maximum bandwidth is 5.

Sample Output

<table>
<thead>
<tr>
<th>MODULE</th>
<th>INVOCATION BANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE</td>
<td>INPUT</td>
</tr>
<tr>
<td>LEVEL</td>
<td>0</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>-2</td>
<td>-1</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>CK2K</td>
<td>GTCR</td>
</tr>
<tr>
<td>LLK</td>
<td></td>
</tr>
<tr>
<td>OLOOK</td>
<td></td>
</tr>
<tr>
<td>LOOK</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5-46
Example Command Set

OLD LIBRARY = REFMAN.
START.
DEPENDENCE, GROUP, LIBRARY.
FOR LIBRARY.
DEPENDENCE, BANDS = 5.
END FOR.
END.

S-47
Description

The outcome of this library command is a report which identifies all modules not present in the library which are called by modules in the library. The sample output shows this report. The data presented is useful in analyzing interfaces of all the modules on the library to other software. Each X mark represents an invocation by the module in the list down the page to the module in the list across the page. The modules are presented in alphabetical order.

Rule

Maximum of 100 library modules for this report.

Sample Output

```
AUXILIARY LIBRARY DEPENDENCE TABLE

- I, N
- AHEFSGGHI
- N, M
- GTRAEYOFVWDOUPUGYDUR
- V, D
- SGRTECOLSGAVBVTTHMSSA
- O, K
- TDOABBVSDLVHS2T4L7STP
- E, E, X
- X, O, W, X, T, D, R
- R, *
- *

ADOCR *
ADDOK  X  XX
BT0D *
CINT  X
CARM *
COUTK  X  X
DTRB *
EFAC *
ERRXX  X  X  X
GETB  X
GETC  X
GTCR *
LOOK  X  X  X
LUK *
MUVI *
MUV4 *
NPUT  X  X  XX  X
LOOK *
OUTSB  X  X  XX  XX  X
POM *
PVALU  X  X
SAPTR *
SONP *
TERMIN  X  X  X
```
Example Command Sets

(1) OLD LIBRARY = REFMAN.
START.
DEPENDENCE, GROUP, LIBRARY.
DEPENDENCE, GROUP, AUXLIB.
END.

(2) OLD LIBRARY = REFMAN.
START.
DEPENDENCE, GROUP, LIBRARY.
DEPENDENCE, GROUP, AUXLIB.
FOR LIBRARY.
DEPENDENCE, PRINT, INVOKES.
END FOR.
END.
**Description**

The outcome of this library command is a report which identifies the interdependencies of all known modules. The sample output shows a sample of this report. Only modules present in the library are listed. Each X mark represents an invocation by the module in the list down the page to the module in the list across the page. The modules are presented in alphabetical order.

**Rule**

Maximum of 100 library modules for this report.

**Sample Output**

```
LIBRARY DEPENDENCE TABLE

| ADFR | X | X |
| ADFTR | X | X | X |
| NBDT | X |
| NTCD | X |
| CKA | X |
| NTOK | X |
| DTCH | X |
| EFAF | X | X |
| ERRIR | X |
| GETB | X |
| GETC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
| GTEC | X |
Example Command Sets

(1) OLD LIBRARY = REFMAN.
    START.
    DEPENDENCE, GROUP, LIBRARY.
    DEPENDENCE, GROUP, AUXLIB.
    END.

(2) OLD LIBRARY = REFMAN.
    START.
    DEPENDENCE, GROUP, LIBRARY.
    DEPENDENCE, GROUP, AUXLIB.
    FOR LIBRARY.
    DEPENDENCE, PRINT, INVOKES.
    END FOR.
    END.
DEPEN DE CE

DEPEN DE GROUP, MODULES = <name>.

DEPEN DE GROUP, MODULES = <name-1>,<name-2>,...,<name-n>.

Description

DEPEN DE GROUP, MODULES = <name-1>,<name-2>,...,<name-n>.

This library command allows the user to identify a group of modules for which inter-group and intra-group computations are performed on the library.

The modules so grouped are related in some way known by the user. For example, the group may be a subset of all the modules in the library which implement a particular functional process. The list of modules named in this command is called the selected group.

The outcome of this library command is a report which identifies all interdependencies between the selected group and the remainder of the library.

The sample output shows a sample of the report generated by this command. Each X mark in the matrix represents an invocation by the module in the list down the page to the module in the list across the page. In the upper left quadrant of the matrix are all intra-group dependencies. In the upper right quadrant are all other modules called by members of the group (whether on the library or not). In the lower left quadrant are all other modules in the library which call one or more members of the group. In the lower right quadrant are the direct invocations of other library modules to all other modules called by the group. The modules are presented in alphabetical order.

Rule

Maximum of 115 modules in the invitee groups.

Sample Output

<table>
<thead>
<tr>
<th>GROUP DEPENDENCY TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-------------------------</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>ACGG00PSXABCGDEEMHILYXRMNPPRTXYYETZ</td>
</tr>
<tr>
<td>N.V</td>
</tr>
<tr>
<td>ODELURP+EFTODPHUYYHVEZ</td>
</tr>
<tr>
<td>V.O</td>
</tr>
<tr>
<td>ONTOYP+SODAARTCLXHYYVUTTTAWSYN</td>
</tr>
<tr>
<td>O,K</td>
</tr>
<tr>
<td>ATACOS+T10DNK0ACARRA 551TALL0 56W</td>
</tr>
<tr>
<td>KE00</td>
</tr>
<tr>
<td>KAAR+1 AL ET L5IU T1</td>
</tr>
<tr>
<td>E.E+KK</td>
</tr>
<tr>
<td>K X KG OR 1 R</td>
</tr>
<tr>
<td>+-------------------------</td>
</tr>
<tr>
<td>ADTOK + AXX + X X + XX +</td>
</tr>
<tr>
<td>CONTOK + A + X +</td>
</tr>
<tr>
<td>GEA + + + +</td>
</tr>
<tr>
<td>GETC + + + X</td>
</tr>
<tr>
<td>OLOOK + + XXXXXX X X XXX X XX +</td>
</tr>
<tr>
<td>OUTSR + + X + X X XX X X +</td>
</tr>
<tr>
<td>POM + + + +</td>
</tr>
<tr>
<td>S0PTR + + X +</td>
</tr>
<tr>
<td>+-------------------------</td>
</tr>
<tr>
<td>LOOK + X XXX + X X +</td>
</tr>
<tr>
<td>LUK + + X X X X X +</td>
</tr>
<tr>
<td>MUV + + + +</td>
</tr>
<tr>
<td>+-------------------------</td>
</tr>
</tbody>
</table>

5-52
DEPENDENCE, GROUP, MODULES = <name>.

Example Command Set

OLD LIBRARY = BIGLIB.
START.
DEPENDENCE, GROUP, MODULES = ADDTOK, PRM, OUTSB, CONTOK, GETB, GETC, SBPTR.
END.
Description

The outcome of this module command is a report which shows (1) the invocations of the selected module from all other known modules and (2) the invocations within the selected module to all other modules. The sample output shows a report produced by this command. For each module the statement number of the invocation and the source text for the invocation are shown.

Sample Output

```
MODULE INVOCATION SPACE...
MODULE (INPUT ) , JAVSTEAT (INPUT ) , PARENT MODULE (INPUT )

---------------------------------------------
3      PROC INPUT $ ( 1 )
---------------------------------------------

INVOCATIONS FROM WITHIN THIS MODULE
---------------------------------------------

MODULE ATOM
STMT  =  40  ATOM ( LYNCH )
STMT  =  49  ATOM ( STERT )

MODULE ERRVR
STMT  =  29  ERRVR ( 4 )

MODULE GTCARD
STMT  =  41  GTCARD ( = STATUS , CRO1 )

MODULE HOLS
STMT  =  18  HOLS ( HDEF ( $ & $ ) )

MODULE INM0B
STMT  =  46  INM0B ( 0 )

MODULE PXX
STMT  =  74  PXX ( HEL + 120 + EXCT )

MODULE TERMIN
STMT  =  49  TERMIN ( 1 )

---------------------------------------------

INVOCATIONS TO THIS MODULE FROM WITHIN LIBRARY
---------------------------------------------

MODULE GTCR
STMT  =  31  INPUT

MODULE OL0OK
STMT  =  119  INPUT
STMT  =  306  INPUT
STMT  =  363  INPUT

---------------------------------------------

5-54
```
Example Command Sets

(1) OLD LIBRARY = BIGLIB.
START.
JAVSTEXT = NPUT.
MODULE = NPUT.
DEPENDENCE, PRINT, INVOKES.
END.

(2) OLD LIBRARY = BIGLIB.
START.
DEPENDENCE, GROUP, LIBRARY.
FOR LIBRARY.
DEPENDENCE, PRINT, INVOKES.
END FOR.
END.
DEPENDECE, SUMMARY.

Description

This library command results in an analysis of the interdependencies of all known modules for interfaces to other software. The sample output shows the report produced by this command.

Sample Output

LIBRARY DEPENDENCE SUMMARY...

THE FOLLOWING PROCEDURES ARE NOT INVOKED BY ANY MODULE ON THE LIBRARY

LOOK

THE FOLLOWING PROCEDURES DO NOT INVoke ANY MODULE ON THE LIBRARY

( * PROCEDURES DO NOT INVoke ANY MODULES AT ALL )

COUNT
OUTPUT
GETR
GETC
GTOO
COUNT
STOP

Example Command Set

OLD LIBRARY = REFMAN.
START.
DEPENDENCE, SUMMARY.
FOR LIBRARY.
DEPENDENCE, PRINT, INVOKES.
END FOR.
END.
Description

This module command displays the hierarchical structure of all modules invoked from the named module. The sample output shows a report from this command. The level in the hierarchy is shown in the first column. The next column shows all modules on the first level. Each succeeding column indicates the relative position of the named module in the hierarchy. Those modules with an asterisk shown to the right are not on the library. Structures which are repeated are indicated by (ETC.). The modules are alphabetical within each level.

Sample Output (Excerpt)

```
DEPENDENCE TREE--- (# NOT IN LIBRARY)

MODULE <INPUT >, JAVTEXT <INPUT >, PARENT MODULE <INPUT >

1 BT0N  *
 1 ERRAR
 2 BT0N
 2 ERRAR  *
 2 OPUT  *
 2 SYSTEM  *
 1 GTCARD  *
 1 MOLS  *
 1 INMDA  *
 1 PPX  *
 1 TERMIN
 2 BT0N
 2 ERRAR
 2 (ETC.,)
 2 FATAL  *
 2 OPUT  *
 2 WRAPUP  *
```

Example Command Set

```
OLD LIBRARY = REFERENCES.
START.
FOR LIBRARY.
DEPENDENCE, TREE.
END FOR.
END.
```
DEScribe = <option>.

Description

This library option command has the form:

    DESCRIBE = ON/OFF. (DEFAULT = OFF.)

This option turns ON or OFF a sequence of printouts which describe the library
manager information. The default is OFF.

Example Command Set

    ALTER LIBRARY = REFMAN.
    DESCRIBE = ON.
    START.
    FOR LIBRARY.
    STRUCTURAL.
    END FOR.
    END.
**Description**

This macro command has three forms:

1. **DOCUMENT.**

2. **DOCUMENT,JAVSTEXT = <text-name>**.

3. **DOCUMENT,JAVSTEXT = <text-name>, MODULE = <name-1>,<name-2>,...<name-n>**.

Each form of the DOCUMENT macro command generates a series of JAVS standard commands which produce reports useful for program documentation, maintenance, and testing.

The expansion of each form of the DOCUMENT macro command is as follows:

1. **ASSIST,CROSSREF,LIBRARY. DEPENDENCE,GROUP,LIBRARY. DEPENDENCE,GROUP,AUXLIB. DEPENDENCE,SUMMARY. FOR LIBRARY. PRINT,MODULE. DEPENDENCE,BANDS = 5. DEPENDENCE,PRINT,INVOKES. END FOR.**

This form (DOCUMENT.) is for documenting the entire library.

2. **ASSIST,CROSSREF,LIBRARY. DEPENDENCE,GROUP,LIBRARY. DEPENDENCE,GROUP,AUXLIB. DEPENDENCE,SUMMARY. JAVSTEXT = <text-name>. FOR JAVSTEXT. PRINT,MODULE. DEPENDENCE,BANDS = 5. DEPENDENCE,PRINT,INVOKES. END FOR.**

This form (DOCUMENT,JAVSTEXT = <text-name>,) is for documenting module interdependencies for the entire library, producing a library-wide cross reference of symbols, and generating module documentary reports for the specified JAVSTEXT.

* Can be used only with the overlay version of JAVS.

5-59
(3) ASSIST,CROSSREF,LIBRARY.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,SUMMARY.
JAVTEXT = <text-name>.
FOR MODULE = <name-1>,<name-2>,...,<name-n>.
PRINT,MODULE.
DEPENDENCE,BANDS = 5.
END FOR.

This form (DOCUMENT,JAVTEXT = <text-name>, MODULE = ....) produces the same report as in (2), except that the module reports are generated only for the specified modules in the JAVTEXT.

Note

The macro command processor will generate the commands:

OLD LIBRARY = TEST.
START.

if the first JAVS command is a macro command (keywords BUILD LIBRARY,
DOCUMENT,PROBE,TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) A maximum of 100 JAVTEXTs can be used in a cross reference mapping.

(2) A maximum of 100 modules can be used in the GROUP reports.

(3) A maximum of 23 modules can be specified in the second form of the DOCUMENT macro.

(4) The DOCUMENT macro command requires that syntax and structural analyses have already been performed on the entire library.

Example Command Sets

(1) BUILD LIBRARY.
DOCUMENT.

This command set will create a new library called TEST and produce JAVS documentation reports for all modules on the library.

(2) OLD LIBRARY = REFMAN.
START.
DOCUMENT,JAVTEXT = EXPRoom,MODULE = EXMPL1,EXMPL2,EXMPL3.
FOR MODULE = EXMPL1,EXMPL2,EXMPL3.
PRINT,DDPATHS.
END FOR.

5-60
This command set uses an existing library called REFMAN to generate the librarywide module interdependency and cross reference reports and the module documentary reports for EXMPL1, EXMPL2, and EXMPL3.
Description

After the operations indicated by the contents of the command file are complete, JAVS must be "shut down" by use of the following command:

`END`

The `END` command must always be the last JAVS command. The actions which occur as a consequence of this command are important to the JAVS user in only one regard—when LISNEW is completed and is to be saved for future runs. The wrapup sequence provides necessary information about the contents of the new library. The sample output shows the standard wrapup output, after running BASIC and STRUCTURAL. This report contains a printout of the Module Descriptor Table and library information for LIBOLD(1) and LIBNEW(2).

Rule

This must be the last JAVS command.

Sample Output

```
JAVS WRAPUP...

KNOWN MODULE DESCRIPTOR BLOCKS...

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>NAME</th>
<th>TYPE</th>
<th>SCORE</th>
<th>LINKS</th>
<th>STATS</th>
<th>EXEC FST</th>
<th>FIRST</th>
<th>WORD</th>
<th>DEP</th>
<th>PARMS</th>
<th>DIRECT</th>
<th>COMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EXTERNAL</td>
<td>EXTERNAL</td>
<td>1202</td>
<td>351</td>
<td>4</td>
<td>1072</td>
<td>13</td>
<td>2</td>
<td>28</td>
<td>2</td>
<td>7</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>EXTERNAL</td>
<td>EXTERNAL</td>
<td>1203</td>
<td>356</td>
<td>4</td>
<td>1076</td>
<td>13</td>
<td>2</td>
<td>28</td>
<td>2</td>
<td>7</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>EXTERNAL</td>
<td>EXTERNAL</td>
<td>1204</td>
<td>357</td>
<td>4</td>
<td>1077</td>
<td>13</td>
<td>2</td>
<td>28</td>
<td>2</td>
<td>7</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>EXTERNAL</td>
<td>EXTERNAL</td>
<td>1205</td>
<td>358</td>
<td>4</td>
<td>1078</td>
<td>13</td>
<td>2</td>
<td>28</td>
<td>2</td>
<td>7</td>
<td>23</td>
<td>1</td>
</tr>
</tbody>
</table>

LIBRARY INFORMATION—RUN OF 5731

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>ACTION</th>
<th>CREATED</th>
<th>ALTERED</th>
<th>XREFS</th>
<th>MODULES</th>
<th>FRAGMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TEST</td>
<td>AV64</td>
<td>3731.2</td>
<td>1220.2</td>
<td>5</td>
<td>35</td>
<td></td>
</tr>
</tbody>
</table>
```

Output Description

The Module Descriptor Block (MDB) summary shown in the sample output, reading left to right, gives the following statistics:

1. Module number on the file
2. Module name
3. START-TERM text name
4. Name of parent module (i.e., the module in which this module is contained)
5. Type of module where
   
   CMPL is a COMPOOL
   
   PROG is a Program
   
   PROC is a Procedure
   
   CLSM is a CLOSE of global scope
   
   CLSP is a CLOSE of local scope
   
   TEXT is a START-TERM text which has not been separated into modules by BASIC

6. Scope of an executable module where
   
   EXT is a program or an external procedure
   
   INT is an internal procedure (i.e., within a program or external procedure)
   
   GLBL is a CLOSE within a program or external procedure
   
   LOCL is a CLOSE within an internal procedure

7. Number of lines of probed text inserted in the module by INSTRUMENT

8. Number of statements in the module

9. Number of executable statements

10. The statement number of the first executable statement

11. Word pairs which measure the amount of library space occupied by the source text

12. Number of tokens in the module

13. Number of symbols defined in module

14. Number of symbols referenced in module

The remainder of the statistics are more applicable to users who wish to follow a testing strategy of analyzing the complexity of their codes and applying testcases to more comprehensively exercise those modules which exhibit a greater tendency to be error prone. These statistics are listed below:
15. Number of other modules invoked by module
16. Number of DD-paths
17. Number of DD-path/statement block entries
18. Number of formal input parameters (-1 until STRUCTURAL)
19. Number of formal output parameters (-1 if the module is a function)
20. YES if DIRECT code is present in module; NO otherwise
21. Statement-based complexity is a summation of the complexity of all statements in the module*
22. DD-path based complexity is a summation of the complexity of all DD-paths in the module. This is used as an indicator of the structural complexity of the module.*

These statistics can be used by the tester to develop a rational approach to program verification of modules which have the greatest tendency toward error (i.e., those which are most complex).

The Library Information report shown in the sample output includes the name for each library, the type of access, the date created, the total size and other useful information.

*These items are provided for future extensions to JAVS and are not presently computed.
Description

This iteration command concludes a block of commands which are repeated for each specified module. There are three sequences of commands which select a number of modules and iterates a block of commands (which cannot contain another iteration). The three forms of command iteration are:

1. FOR MODULE = <name-1>, ..., <name-n>.
   (any set of commands)
   END FOR.

2. FOR JAVSTEXT.
   (any set of commands)
   END FOR.

3. FOR LIBRARY.
   (any set of commands)
   END FOR.

Rule

Maximum of 150 modules selected in command iteration loop.

Example Command Set

OLD LIBRARY = REFMAN.
START.
ASSIST,CROSSREF,LIBRARY.
FOR LIBRARY.
PRINT,MODULE.
ASSIST,PICTURE.
END FOR.
END.

5-65
Description

The following sequence selects each known module within the "current text" and iterates a block of commands (which cannot contain another iteration) once for each known module of that text:

```
FOR JAVSTEXT.
    (any set of commands)
END FOR.
```

Rule

Maximum of 150 modules selected in this iteration command.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROC.
FOR JAVSTEXT.
    PRINT,MODULE.
END FOR.
END.
```
FOR LIBRARY.

Description

The following sequence selects each known module on the library and iterates a block of commands (which cannot contain another iteration) once for each module:

```
FOR LIBRARY.
(any set of commands)
END FOR.
```

Rule

Maximum of 150 modules selected for this iteration command.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
DEPENDENCE, PRINT, INVOKES.
PRINT, MODULE.
END FOR.
DEPENDENCE, GROUP, LIBRARY.
END.
```
FOR MODULE = <name>...

Description

The following sequence selects a number of modules, by name, and iterates a block of commands (which cannot contain another iteration) once for each specified module:

FOR MODULE = <name-1>,...,<name-n>.
(any set of commands)
END FOR.

Rule

A maximum of 25 modules can be specified.

Example Command Set

OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR MODULE = EXPROGM,EXMPL1.
PRINT,MODULE.
PRINT,DDPATHS.
END FOR.
END.

5-68
Description

The INSTRUMENT command with all options at their default values produces the small report in the sample output. The actions performed by the INSTRUMENT command is to record probe text statements on LIBNEW.

Rules

(1) See INSTRUMENT constraints in Sec. 3.4.

(2) The INSTRUMENT options are reset to their default values following execution of this command.

Sample Output

```
JOVI  AUTOMATED VERIFICATION SYSTEM *** INSTRUMENTATION ***
OPTIONS IN EFFECT ...
                      DO-PATH    PROBE   MODULE    PROBE   TEST    PROBE
                      PROBE    PROBE    PROBE
INSTRUMENTING ENTRY POINTS, RETURNS AND DO-PATHS OF MODULE <EXPROM> OF JAVSTEX <EXPROM>
```

Example Command Sets

(1) ALTER LIBRARY = REFMAN.
    START.
    FOR LIBRARY.
    INSTRUMENT.
    END FOR.
    PUNCH, JAVSTEXT = EXPROM, INSTRUMENTED = ALL.
    END.

(2) OLD LIBRARY = REFMAN.
    START.
    JAVSTEXT = EXPROM.
    FOR JAVSTEXT.
    INSTRUMENT, MODE = FULL.
    INSTRUMENT.
    END FOR.
    PUNCH, JAVSTEXT = EXPROM, INSTRUMENTED = ALL.
    END.
Description

INSTRUMENT,MODE = INVOCATION/DDPATHS/DIRECTIVES/FULL.
(DEFAULT = DDPATHS.)

This command allows user control over the extent of probe insertion. There are two types of instrumentation: structural and performance.

Structural instrumentation may be requested at the module invocation and return level with the command:

INSTRUMENT,MODE = INVOCATION.

This is useful in developing an invocation and return trace for a set of modules.

Structural instrumentation at the DD-path level is produced with the default option of the command:

INSTRUMENT,MODE = DDPATHS.

The instrumentation for module invocation and return level is also included. This is needed to develop a complete execution trace during test execution.

JAVA performance instrumentation is produced with the command:

INSTRUMENT,MODE = DIRECTIVES.

This is used when execution-time monitor output is to be generated as the result of user-supplied JAVA computation directives present in the JOVIAL source text (see Sec. 1.3).

Probe text for both structural instrumentation and performance instrumentation is produced with the command:

INSTRUMENT,MODE = FULL.

MODE = FULL is, in effect, a combination of MODE = DDPATHS and MODE = DIRECTIVES.

Example Command Sets

(1) ALTER LIBRARY = REFMAN.
START.
FOR LIBRARY.
INSTRUMENT,MODE = INVOCATION.
INSTRUMENT.
END FOR.
END.

3-70
INSTRUMENT, MODE = <option>. (Cont.)

(2) OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXPROGM.
INSTRUMENT, MODE = INVOCATION.
INSTRUMENT.
FOR MODULE = EXMPL1, EXMPL2.
INSTRUMENT, MODE = FULL.
INSTRUMENT.
END FOR.
PUNCH, JAVSTEXT = EXPROGM, INSTRUMENTED = EXPROGM, EXMPL1, EXMPL2.
END.
INSTRUMENT, PROBE, DDPATH = <name>.

Description

INSTRUMENT, PROBE, DDPATH = <probe-name>. (DEFAULT = PROBE.)

This command is included for use only when it is necessary to change the name of the DD-path data collection routine to avoid conflict. If the default name PROBE is used in the JOVIAL source being processed, a different name must be selected (and appropriate modifications made in Test Execution).

Example Command Set

ALTER LIBRARY = REFMAN.
START.
FOR LIBRARY.
INSTRUMENT, PROBE, DDPATH = DDPATH.
INSTRUMENT.
END FOR.
END.
INSTRUMENT, PROBE, MODULE = <name>.

Description

INSTRUMENT, PROBE, MODULE = <invocation-name>. (DEFAULT = PROBM.)

This command is included for use only when it is necessary to change the name of the module invocation-and-return data collection routine to avoid conflict. If the default name PROBM is used in the JOVIAL source being processed, a different name must be selected (and appropriate modifications made in Test Execution).

Example Command Set

OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT, MODE = INVOCATION.
INSTRUMENT, PROBE, MODULE = INVOR.
INSTRUMENT.
END FOR.
PUNCH, JAVSTEXT = EXPROGM, INSTRUMENTED = ALL.
PRINT, JAVSTEXT = EXPROGM, INSTRUMENTED = ALL.
END.
INSTRUMENT,PROBE,TEST = <name>.

Description
INSTRUMENT,PROBE,TEST = <test-name>. (DEFAULT = PROBI.)

This command is included for use only when it is necessary to change the name of the test-beginning and test-end data collection routine to avoid conflict. If the default name PROBI is used in the JOVIAL source being processed, a different name must be selected (and appropriate modifications made in Test Execution).

Example Command Set

OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROM.
FOR MODULE = EXPROM,EXMPL.
INSTRUMENT,STARTTEST = EXMPL,EXPROM,0.
INSTRUMENT,STOPTEST = EXPROM,EXPROM,0.
INSTRUMENT,PROBE,TEST = TESTP3.
INSTRUMENT,MODE = FULL.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROM,INSTRUMENTED = ALL.
END.
INSTRUMENT,STARTTEST = <options>.

Description

In order to identify test cases and control the recording of data on the AUDIT file, the user must supply invocations to the data collection routine, PROBI. The invocations can be manually inserted prior to Test Execution, or they can be automatically inserted during instrumentation (see Sec. 2.6.1).

The INSTRUMENT,STARTTEST command causes JAVS to insert an invocation to PROBI for identifying a new test case. The command is of the form:

INSTRUMENT,STARTTEST = <m-name>,<t-name>,<no.>{,<TESNAM>,<TFLAG>}

where

- m-name = module name
- t-name = JAVSTEXT name
- no. = statement number
- TESNAM = test case identifier DEFAULT = 8H(CASE )
- TFLAG = test file control value DEFAULT = 2

The command options must be given in the order shown above. The user must specify the module and JAVSTEXT names in which the PROBI invocation is to be inserted. The user must also specify the statement number (using the statement number presented in a PRINT,MODULE listing) before which the invocation is to be inserted. The user may specify the statement number to be 0. This results in placing the PROBI call immediately prior to the first software probe (at the first executable statement).

TESNAM and TFLAG are the two input parameters to PROBI. If these parameters are not specified in this command, the default values will be used, causing tracing of module invocations and returns. TESNAM may be up to 8 characters; TFLAG may be 1, 2, or 3. A maximum of ten INSTRUMENT,STARTTEST commands can be used for a single INSTRUMENT verb. The module and JAVSTEXT names may be different in each command.

Example Command Sets

See the examples for the INSTRUMENT,STOPTEST command.
INSTRUMENT,STOPTEST = <options>.

Description

This command is used to automatically insert calls to data collection routine PROBI to terminate data collection on the AUDIT file. Its utilization is similar to the INSTRUMENT,STARTTEST command. The command's form is:

INSTRUMENT,STOPTEST = <m-name>,<t-name>,<no.>.

where

m-name = module name

t-name = Javstext name

no. = statement number

Only one INSTRUMENT,STOPTEST command may be used for a single INSTRUMENT verb. The PROBI call to terminate testing will be placed immediately prior to the specified statement number or, if the statement number is specified as 0, immediately following the software probe at each exit from the module.

Example Command Sets

(1) ALTER LIBRARY = REFMAN.
     JAVSTEXT = EXPROGM.
     FOR JAVSTEXT.
     INSTRUMENT,STARTTEST = EXPROGM,EXPROGM,16.
     INSTRUMENT,STARTTEST = EXPROGM,EXMPL1,6.
     INSTRUMENT,STOPTEST = EXPROGM,EXPROGM,31.
     INSTRUMENT.
     END FOR.
     PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
     END.

     This command set will instrument DD-paths, invocations and returns for all modules in JAVSTEXT EXPROGM. In addition, the calls to PROBI to initiate a new test base will be automatically inserted immediately prior to statement 16 in module EXPROGM and the first executable statement in module EXMPL1. The test termination PROBI invocation will be automatically inserted prior to statement 31 in module EXPROGM. The default TESNAM and TFLAG will be used; thus the AUDIT file will contain only a module invocation trace but coverage data at the DD-path level. The instrumented source text for JAVSTEXT EXPROGM will be written to file LPUNCH, as well as stored on LIBNEW.

(2) OLD LIBRARY = REFMAN.
     START.
     JAVSTEXT = EXPROGM.
     MODULE = EXPROGM.
INSTRUMENT, STOPTEST = <options>. (Cont.)

INSTRUMENT, MODE = INVOCATIONS.
INSTRUMENT, STARTTEST = EXPROGM, EXPROGM, 0, EXPROGM, 1.
INSTRUMENT, STOPTEST = EXPROGM, EXPROGM, 0.
INSTRUMENT.
FOR MODULE = EXMPL1, EXMPL2, EXMPL3.
INSTRUMENT, STARTTEST = EXMPL1, EXPROGM, 9, EXMPL1, 3.
INSTRUMENT.
END FOR.
PUNCH, JAVSTEXT = EXPROGM, INSTRUMENTED = ALL.
END.

This command set will instrument module EXPROGM for invocations and
returns and will instrument the remaining modules in the JAVSTEXT for DD-paths
as well. The test case initiations will be automatically inserted in module
EXPROGM at its first executable statement and in module EXMPL1 at statement 9.
The termination of all test cases will be inserted following the last execu-
table statement in EXPROGM.

The name of each test case will be the name of the module containing the
PROBI call. During Test Execution, the AUDIT file will contain no trace for
module EXPROGM, but it will have a DD-path and module invocation/return trace
for the remaining modules as soon as PROBI is invoked with TFLAG = 3. The
AUDIT file will contain module invocation/return summary data for all modules
in the JAVSTEXT and DD-path summary (coverage) data for modules EXMPL1,
EXMPL2, and EXMPL3.

The instrumented source text for JAVSTEXT EXPROGM will be written to
file LPUNCH.
Description

Some JAVS commands require specification of the START-TERM text sequence upon which computations are to be performed. START-TERM texts are known to JAVS by their names. The following command makes the "current text" be the named one:

JAVSTEXT = <text-name>.

All subsequent commands (if they refer to a specific text) are applied to the specified text. There can be any number of JAVSTEXT = commands. A new JAVSTEXT = command replaces and supersedes a previous JAVSTEXT = command.

Example Command Sets

(1) ALTER LIBRARY = REPMAN.
    JAVSTEXT = EXPROC.
    FOR JAVSTEXT.
    INSTRUMENT.
    END FOR.
    END.

(2) ALTER LIBRARY = REPMAN.
    JAVSTEXT = EXCOMPL.
    MODULE = EXCOMPL.
    PRINT,MODULE.
    JAVSTEXT = EXPROC.
    FOR JAVSTEXT.
    STRUCTURAL.
    PRINT,MODULE.
    END FOR.
    END.
**Description**

The MERGE command can be used (1) to add new modules to an existing library, and (2) to replace old modules with new modules having the same module name and same text name. The command is

```
MERGE.
```

The MERGE command merges a LIBOLD with a LIBNEW. After merging, LIBNEW contains all modules on the original LIBNEW, plus all modules on the LIBOLD that do not have the same module name and same text name as a module on the LIBNEW. Both LIBOLD and LIBNEW must be specified in the library commands (see Sec. 4 and 4.1).

**Rule**

Maximum of 250 modules in the combined library.

**Example Command Sets**

1. OLD LIBRARY = REFMAN.
   CREATE LIBRARY = NEWLIB.
   START.
   BASIC.
   MERGE.
   FOR LIBRARY.
   PRINT,MODULE.
   END FOR.
   END.

   This command set will create a new library containing the syntax analysis for the source text on file READER.* After the library is built, the contents of the old library (LIBOLD), excluding any modules with the same module and text names as those on LIBNEW, will be copied onto the new library. This command set can be used for replacing a module on a library with a modified version of that module. In this case LIBNEW will contain the same module names as LIBOLD contains, but LIBNEW will have the modified module(s). Following the library merge, the above command set will print all of the modules on LIBNEW.

2. OLD LIBRARY = REFMAN.
   ALTER LIBRARY = OLIBE.
   START.
   MERGE.
   FOR LIBRARY.
   STRUCTURAL.
   END FOR.
   END.

*See Table 1.1 on page 1-4.
MERGE. (Cont.)

This command set assumes that syntax analysis has been performed on libraries REFMAN and OLIBE. All of the modules in REFMAN (on LIBOLD), not already contained in OLIBE, will be copied to LIBNEW. Structural analysis will be performed on OLIBE. REFMAN remains unchanged.

(3) OLD LIBRARY = LIBE1.
ALTER LIBRARY = LIBE2.
START.
MERGE.
END.

This command set assumes that at least syntax analysis has been performed on both libraries. LIBOLD will be copied onto LIBNEW (LIBE2), excluding any modules in LIBE1 which already exist in LIBE2. No other processing is performed.
MODULE = <name>.

Description

Modules are known to JAVS by their names and the text to which they belong. The following makes the "current module" within the "current JAVSTEXT" be the named one:

```
MODULE = <name>.
```

All subsequent commands (if they refer to a specific module) are applied to the specified module within the "current text." If no current JAVSTEXT has been specified, the module selector will locate the module using only its name.

Rule

If there are duplicate module names, in different JAVSTEXTs, the module selector will choose the last one on the library.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXMPL1.
ASSIST,PICTURE.
MODULE = EXMPL2.
PRINT,MODULE.
END.
```

This command set will produce the control flow picture for module EXMPL1 and will print the source text for EXMPL2.
OLD LIBRARY = <name>.

Description

This command specifies that LIBOLD is to be used during the current run. The name identifying the library should be that same as when the library was created. LIBOLD is a read-only library.

Rule

If the name is different, an informative message is printed, but processing continues.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
DEPENDENCE,GROUP,LIBRARY.
FOR LIBRARY.
DEPENDENCE,PRINT,INVOKES.
PRINT,MODULE.
END FOR.
END.
```
Description

This command produces a detailed listing of the JAVS information describing each DD-path within the current module.

The properties of each DD-path shown are:

- The DD-path number
- The first statement on the DD-path
- The last statement on the DD-path
- The number of edges (i.e., non-consecutive statement segments) which comprise the DD-path
- The complexity of the DD-path*
- The highest level for the DD-path*
- Parallelism indicator*
- The number of times the DD-path has been tested*
- The number of statements on the DD-path
- DS Table index of the first entry in the list of statements on the DD-path
- The list of statements on the DD-path in execution order.

*The RADC version of JAVS does not compute a value for this entry.
Sample Output

DD-PATH TABLE LISTING

MODULE (EXAMPLE)  "JASTEXT" "EXPROM"  "PARENT MODULE" "EXPROM"

<table>
<thead>
<tr>
<th>NO.</th>
<th>ST.</th>
<th>EDG COM</th>
<th>IND.</th>
<th>LVL</th>
<th>DD-PATH</th>
<th>ST.</th>
<th>INDEX</th>
<th>STATEMENTS ON DD-PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>21</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>21</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>23</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td>25</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>23</td>
<td>78</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>29</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>33</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>34</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>36</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>36</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>33</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>33</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>23</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>23</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>34</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>34</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>34</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>34</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>41</td>
<td>41</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

Example Command Set

OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT,MODULE.
PRINT,DDP.
PRINT,DDPATHS.
END FOR.
END.
PRINT, DDPATHS:

Description

For the current module, this command produces a detailed listing of the source text for the statements which begin a DD-path with the DD-path description included. The report is similar in format to the report from the PRINT, MODULE command. Only those statements which begin a DD-path or are crucial to the interpretation of a DD-path (e.g., FOR, SWITCH) are displayed.

Sample Output

<table>
<thead>
<tr>
<th>NO.</th>
<th>LVL</th>
<th>STATEMENT</th>
<th>DDPATHS GENERATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>PROC EXMPL (LIMIT1 + LIMIT2)</td>
<td>** DD-PATH 1 IS PROCEDURE ENTRY</td>
</tr>
</tbody>
</table>
| 9   | 1   | IF LIMIT1 GE 100 | ** DD-PATH 2 IS TRUE BRANCH  
|     |     |              | ** DD-PATH 3 IS FALSE BRANCH |
| 12  | 1   | FOR J = 1 TO LIMIT1 | ** DD-PATH 4 IS TRUE BRANCH  
| 13  | 2   | BEGIN          | ** DD-PATH 5 IS FALSE BRANCH |
| 18  | 2   | FOR J = 1 TO LIMIT2 | ** DD-PATH 6 IS TRUE BRANCH  
| 19  | 3   | BEGIN          | ** DD-PATH 7 IS TRUE BRANCH |
| 21  | 3   | IFEITH J LE 3 | ** DD-PATH 8 IS SWITCH OUTWAY 1 |
| 23  | 3   | ORIG 1'S      | ** DD-PATH 9 IS SWITCH OUTWAY 2 |
| 25  | 3   | END           | ** DD-PATH 10 IS SWITCH OUTWAY 3 |
| 27  | 3   | SWITCH PICK # (LABEL1 + LABEL2) | ** DD-PATH 11 IS SWITCH OUTWAY 4 |
| 28  | 3   | GOTO PICK (6 INOES - 1) | ** DD-PATH 12 IS TRUE BRANCH  
|     |     |              | ** DD-PATH 13 IS FALSE BRANCH |
| 30  | 3   | LABEL.        | ** DD-PATH 14 IS TRUE BRANCH  
|     |     |              | ** DD-PATH 15 IS FALSE BRANCH |
| 32  | 3   | ORIG RESULT EQ 4 | ** DD-PATH 16 IS TRUE BRANCH  
| 34  | 3   | ORIG 1'S      | ** DD-PATH 17 IS LOOP ON FOR AGAIN |
| 36  | 3   | END           | ** DD-PATH 18 IS ESCAPE FOR LOOP |
| 39  | 3   |              | ** DD-PATH 19 IS LOOP ON FOR AGAIN |
| 41  | 2   | END           | ** DD-PATH 20 IS ESCAPE FOR LOOP |

5-85
PRINT, DDPATHS: (Cont.)

Example Command Set

OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROC.
FOR JAVSTEXT.
PRINT, DDPATHS.
END FOR.
END.
PRINT,DMT.

Description

This command produces a listing of the internal dependent module table (DMT) descriptions. The table contains an entry for each invocation contained in the current module. Pointers to other internal tables (e.g., the SB and SLT) are included. This output is intended for JAVS system maintenance only.

Sample Output

DEPENDING MODULE TABLE LISTING

EXTERNAL SLT START OF WRD-PR OF SERIAL
NO. INVOKED NUMBER INVOCATION INVOCATION INVOCATION

1 EXMPI 24 24 1 1
2 EXMPL2 27 3 1

Example Command Set

OLD LIBRARY = REFMN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXPROGM.
PRINT,DMT.
END.

5-87
PRINT, JAVSTEXT = <name>.

Description

This command produces a listing of the source statements in the named START-TERM text as they were originally stored on the library (see Sec. 1.4). The sample output is after STRUCTURAL processing. The salient features are:

- The statement number assigned by JAVS
- The level of nesting of the statement
- The label(s) and text of the statement indented to show nesting level (e.g., FOR, IF, and IFEITH hierarchy)
- The numbers of the DD-paths which are identified and assigned by JAVS
- Statement type for most control statements
**SAMPLE OUTPUT (EXCERPT)**

```
----

PRINT, JAVSTEXT = <name>. (Cont.)

JAVSTEXT *EXPROG* LISTING

<table>
<thead>
<tr>
<th>NO.</th>
<th>LVL</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>#. JAVSTEXT <em>EXPROG</em> COMPARE ( EXEMPL ) #</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>START</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td># JAVIAL SIMPLE TEST PROGRAM #</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>ITEM ID = A $</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>ITEM ITEMS = 1 2 4 5 $</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>ITEM ITEMS = 1 2 4 5 $</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>ITEM ITEMS = 4 5 $</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>ITEM ITEMS = 5 $</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>OVERLAY ITEMS = ITEMS $</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>OVERLAY ITEMS = ITEMS $</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>ITEM CARDS = NO. S</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>FILE READER = G R &amp; W EOIF TAPES S</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>FILE PROCESS = G R &amp; W EOIF TAPES S</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>MONITOR ID = ITEMS = ITEMS S</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>MESSAGE = MESSAGES S</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>OUTPUT PRINT MESSAGES S</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>I/O</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>INPUT READER CARDS</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>IF READER MSG</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>BEGIN</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>BYTE (S 0 4 5)</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>BYTE (S 0 4 5)</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>EXAMPLES</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>CLOSE EXAMPLES S</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>MAIN CLOSE #</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>IF ITEMS = 0 S</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>GOTO EXAMPLES S</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>GOTO 90 S</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>END</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td># Example Set</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>STOP S</td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>PROC EXAMPLES ( LIMIT )</td>
</tr>
<tr>
<td>34</td>
<td>1</td>
<td>BEGIN</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
<td>ITEM LIMIT = 2 4 5 S</td>
</tr>
<tr>
<td>36</td>
<td>1</td>
<td>ARRAY FILL = 100 1 2 4 5 S</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>ITEM RESULT = 2 4 5 S</td>
</tr>
<tr>
<td>38</td>
<td>1</td>
<td>ITEM INDEXS = 2 4 5 S</td>
</tr>
<tr>
<td>39</td>
<td>1</td>
<td># example RESULT #</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>IF LIMITS = 0 S</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
<td>RESULT = 0 $</td>
</tr>
</tbody>
</table>

**Example Command Set**

OLD LIBRARY = REFMAN.
START.
PRINT, JAVSTTEXT = EXPROG.
END.

5-89
PRINT, JAVSTEXT = <name>, INSTRUMENTED = <option>.

Description

This command has two forms:

PRINT, JAVSTEXT = <text-name>, INSTRUMENTED = ALL. or
PRINT, JAVSTEXT = <text-name>, INSTRUMENTED = <name-1>, <name-2>, ..., <name-n>.

The command produces a listing of the instrumented source (i.e., results of INSTRUMENT) in the named START-TERM text. The first form is used when all modules of the text are listed with instrumentation inserted. The second form is used when specified modules in the text are to have instrumentation inserted and all other modules in the text are not instrumented. Any module which has not been processed by INSTRUMENT will be shown in its original form. The modules must be listed in the same order in which they appear on the library. The salient features of the output are:

- The statement number within the module of the original source statement assigned by JAVS
- The level of nesting of the statement or an indicator (?) if the statement shown is a probe (i.e., generated by INSTRUMENT)
- The label(s) and text of the statement indented to show nesting level.

The code generated by the instrumentation does not affect the logic of the user's program. In some instances (as with the last statement 11 in the sample output), a P is placed by the user's own code. This happens when the instrumentation code modification, stored in the library, is identical to the user's statement.

Rule

Specified modules must be in the same order in which they appear on the library.

The instrumented text consists of the source text interspersed with the probe text generated by INSTRUMENT.

5-90
Sample Command Set

ALTER LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
PRINT,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
Description

This command produces a detailed listing of the source text of the current module. The sample output is after STRUCTURAL processing. The salient features are:

- The statement number assigned by JAVS
- The level of nesting of the statement
- The label(s) and text of the statement indented to show nesting level (e.g., FOR, IF, and IFEITH hierarchy)
- The numbers of the DD-paths which are identified and assigned by JAVS
- Statement type for most control statements
Sample Output

MODULE STATEMENT LISTING

MODULE (EAMPI) -> JAVESN(EKPRAM) , PARENT MODULE (EAPRAM)

<table>
<thead>
<tr>
<th>NO.</th>
<th>LVL</th>
<th>STATEMENT</th>
<th>DD-PATHS</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>PRINT EAMPI</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>ITEM LIMIT 1 24 5 $</td>
<td>( 2- 3 ) IF</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>ITEM LIMIT 1 24 5 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>ARRAY FILL 100 1 24 5 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>ITEM RESULT 1 24 5 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>ITEM INDS 1 24 5 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>#. TRAC1 RESULT #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>IF LIMIT 60 100 $</td>
<td>( 2- 3 ) IF</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>RESULT 4 $</td>
<td>FOR3</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>FOR I = 1 * 1 + LIMIT 1 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>BEGIN</td>
<td>FOR3</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>#. EXPECT RESULT 1 .5 #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>FILL IS 1 - 1 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>RESULT 4 RESULT IS 1 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>#. ASSERT RESULT 10 #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>FOR J = 1 * 1 + LIMIT 2 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>BEGIN</td>
<td>FOR3</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td># CLOSE #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>IF EITHE4 10 3 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>4</td>
<td>INDS = J $</td>
<td>( 4- 5 ) IF</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>ORIF I $</td>
<td>( 6 )</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>4</td>
<td>INDS 4 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>3</td>
<td>ORIF I $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>IF EITHE4 END #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>3</td>
<td>#. IFHE4 END</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>4</td>
<td>SWITCH PICK x 1 LABEL2 LABEL1 LABEL1 $</td>
<td>( 13 ) 11----</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>3</td>
<td>GOTO P1CK (5 INDS = 1 $)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>3</td>
<td>LABEL2.</td>
<td>GO TO EAMPI $</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>LABEL2.</td>
<td>( 12 ) 13---- IF</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>3</td>
<td>IEITHE4 RESULT 6 $</td>
<td>( 14 ) 15--- ORIF</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>ORIF RESULT EQ 4 $</td>
<td>( 16 )</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>MESSAG = 503 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>4</td>
<td>ORIF I $</td>
<td>( 16 )</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>MESSAG = 504 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>4</td>
<td>ME$</td>
<td>( 16 )</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>4</td>
<td>END #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>3</td>
<td>OUTPUT P1CK MESSAG $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>3</td>
<td>END #</td>
<td>( 17 ) 18--- ISO</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td># J #</td>
<td>( 14 ) 20---</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>2</td>
<td>END #</td>
<td>( 19 ) 20---</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>3</td>
<td># J #</td>
<td>( 14 ) 20---</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>3</td>
<td># OFF TRAC1 RESULT #</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>1</td>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example Command Set

OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT,MODULE.
END FOR.
END.

5-93
Description

This command produces a detailed listing of how the statements within the current module are stored in the statement block (SB) on the library. Each statement is shown in token and word-pair form. The upper line contains the text of the statement and the lower line indicates the token classification. This output is intended for JAVS system maintenance only.

Sample Output

```
MODULE (EXPROM) JAVSTEXT (EXPROM) PARENT MODULE (EXPROM)

STATEMENT 1
## JAVA TEST EXEC COMP UT ( EXEC HPL ) ##
DELI ITEM CONT ENDT ITEM CONT ENDT ITEM CONT ENDT ITEM CONT ENDT ITEM CONT ENDT
STATEMENT 2
START ITEM
KEYW ENOT
STATEMENT 3
#
DELI
STATEMENT 4
## JAVA TEST EXEC COMP UT ( EXEC HPL ) ##
CONT CONT ENDT CONT ENDT CONT ENDT CONT ENDT CONT
STATEMENT 5
ITEM 10 ITEM 4 ITEM
KEYW ENOT 1 ABBR DELI
STATEMENT 6
ITEM 10 ITEM 4 ITEM
KEYW ENOT 2 ENDT ABBR ABBR DELI
STATEMENT 7
ITEM 10 ITEM 4 ITEM
KEYW ENOT 3 ENDT ABBR ABBR DELI
```

Example Command Set

```
OLD LIBRARY = REFERMAN.
START.
JAVSTEXT = EXPROM.
MODULE = EXPROM.
PRINT,SB.
END.
```

5-94
Description

This command produces a detailed listing of the internal statement descriptors generated and saved by JAVS. This output is intended for researchers or sophisticated JAVS users only.

Sample Output

<table>
<thead>
<tr>
<th>STATEMENT DESCRIPTOR LISTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE &lt;EXMP1&gt; &gt; JAVSTEXT &lt;EXPROGM&gt; &gt; PARENT MODULE &lt;EXPROGM&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NO.</th>
<th>TYPE</th>
<th>CLASS</th>
<th>CODE</th>
<th>SF6 NUM</th>
<th>W0-PPS</th>
<th>SB</th>
<th>NO. LABELS</th>
<th>W0-PPS</th>
<th>INDEX</th>
<th>INTRA- MTH PTR</th>
<th>INTRA- MTH PTR</th>
<th>COMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROC</td>
<td>100</td>
<td>38</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>BEGIN</td>
<td>100</td>
<td>39</td>
<td>7</td>
<td>2</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>ITEM</td>
<td>1</td>
<td>40</td>
<td>8</td>
<td>2</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>ITEM</td>
<td>1</td>
<td>41</td>
<td>0</td>
<td>0</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>END</td>
<td>1</td>
<td>42</td>
<td>9</td>
<td>0</td>
<td>61</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>ITEM</td>
<td>1</td>
<td>43</td>
<td>8</td>
<td>0</td>
<td>76</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>ITEM</td>
<td>1</td>
<td>44</td>
<td>8</td>
<td>0</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>END</td>
<td>1</td>
<td>45</td>
<td>7</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>IF</td>
<td>200</td>
<td>46</td>
<td>6</td>
<td>0</td>
<td>125</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>ASMT</td>
<td>100</td>
<td>47</td>
<td>5</td>
<td>0</td>
<td>137</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>ASMT</td>
<td>100</td>
<td>48</td>
<td>5</td>
<td>0</td>
<td>147</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>FOR3</td>
<td>300</td>
<td>49</td>
<td>10</td>
<td>0</td>
<td>157</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>BEGIN</td>
<td>100</td>
<td>50</td>
<td>2</td>
<td>0</td>
<td>177</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>OR</td>
<td>1</td>
<td>51</td>
<td>31</td>
<td>0</td>
<td>181</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>ASMT</td>
<td>100</td>
<td>52</td>
<td>10</td>
<td>0</td>
<td>203</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>ASMT</td>
<td>100</td>
<td>53</td>
<td>12</td>
<td>0</td>
<td>223</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>OR</td>
<td>1</td>
<td>54</td>
<td>9</td>
<td>0</td>
<td>247</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>FOR3</td>
<td>300</td>
<td>55</td>
<td>10</td>
<td>0</td>
<td>265</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>BEGIN</td>
<td>100</td>
<td>56</td>
<td>2</td>
<td>0</td>
<td>285</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>COMT</td>
<td>1</td>
<td>57</td>
<td>4</td>
<td>0</td>
<td>289</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>IEI</td>
<td>200</td>
<td>58</td>
<td>6</td>
<td>0</td>
<td>297</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>ASMT</td>
<td>100</td>
<td>59</td>
<td>5</td>
<td>0</td>
<td>309</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>ORIF</td>
<td>500</td>
<td>60</td>
<td>4</td>
<td>0</td>
<td>319</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>ASMT</td>
<td>100</td>
<td>61</td>
<td>5</td>
<td>0</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>END</td>
<td>100</td>
<td>62</td>
<td>7</td>
<td>0</td>
<td>337</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>COMT</td>
<td>1</td>
<td>63</td>
<td>4</td>
<td>0</td>
<td>339</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>JNSW</td>
<td>1</td>
<td>64</td>
<td>19</td>
<td>0</td>
<td>347</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>GTSW</td>
<td>100</td>
<td>65</td>
<td>11</td>
<td>0</td>
<td>355</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>GIICL</td>
<td>100</td>
<td>66</td>
<td>8</td>
<td>0</td>
<td>407</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>IEI</td>
<td>200</td>
<td>67</td>
<td>10</td>
<td>0</td>
<td>423</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>ASMT</td>
<td>100</td>
<td>68</td>
<td>6</td>
<td>0</td>
<td>443</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>ORIF</td>
<td>200</td>
<td>69</td>
<td>7</td>
<td>0</td>
<td>455</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>ASMT</td>
<td>100</td>
<td>70</td>
<td>6</td>
<td>0</td>
<td>469</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>ORIF</td>
<td>500</td>
<td>71</td>
<td>4</td>
<td>0</td>
<td>481</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>ASMT</td>
<td>100</td>
<td>72</td>
<td>6</td>
<td>0</td>
<td>486</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>END</td>
<td>100</td>
<td>73</td>
<td>1</td>
<td>0</td>
<td>501</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>37</td>
<td>COMT</td>
<td>1</td>
<td>74</td>
<td>4</td>
<td>0</td>
<td>503</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td>OUTP</td>
<td>100</td>
<td>75</td>
<td>7</td>
<td>0</td>
<td>511</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>39</td>
<td>END</td>
<td>400</td>
<td>76</td>
<td>1</td>
<td>0</td>
<td>525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>COMT</td>
<td>1</td>
<td>77</td>
<td>3</td>
<td>0</td>
<td>527</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>END</td>
<td>400</td>
<td>78</td>
<td>1</td>
<td>0</td>
<td>533</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td>COMT</td>
<td>1</td>
<td>79</td>
<td>3</td>
<td>0</td>
<td>535</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>END</td>
<td>400</td>
<td>80</td>
<td>8</td>
<td>0</td>
<td>541</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>END</td>
<td>100</td>
<td>81</td>
<td>1</td>
<td>0</td>
<td>557</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Example Command Set

OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXMP1.
PRINT,SDB.
END.

5-95
PRINT, SLT.

Description

This command produces a listing of the symbol locator table (SLT) (i.e., a list of the symbols referenced within the current module). This list includes those identifiers and constants which appear in the source text which are essential for JAVS processing. The first eight characters of the symbol are stored and pointers to other JAVS-internal tables are included. This output is intended for JAVS system maintenance only.

Sample Output

SYMBOL LOCATION TABLE LISTING

<table>
<thead>
<tr>
<th>NO.</th>
<th>SYMBOL</th>
<th>MODULE</th>
<th>JAVSTEXT</th>
<th>STB</th>
<th>FIRST</th>
<th>LAST</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LIMIT1</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>LIMIT2</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>LIMIT3</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>LIMIT4</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>LIMIT5</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>LIMIT6</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>LIMIT7</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>LIMIT8</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>LIMIT9</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>LIMIT10</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>LIMIT11</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>LIMIT12</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>LIMIT13</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>LIMIT14</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>LIMIT15</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>LIMIT16</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>LIMIT17</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>LIMIT18</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>LIMIT19</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>LIMIT20</td>
<td>EXWPL1</td>
<td>EXPROM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example Command Set

OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT, SLT.
END FOR.
PRINT, SLT.

5-96
Description

This command produces a listing of the detailed, internal symbol table (STB) descriptors for each symbol defined in the current module. Normally the list includes only those symbols essential to structural analysis of the module. A BASIC processing option causes all symbols defined in the module to be entered in the STB (see BASIC SYMBOLS = ON). The properties of each symbol and pointers to other JAVS-internal tables are included. This output is intended for JAVS system maintenance only.

Sample Output

| NO. | SYMBOL | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|-----|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1   | PICK   | 2 | LOC | LNSH | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2   | LABEL1 | 1 | LOC | LNSH | 4 | 14| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3   | LABEL2 | 1 | LOC | LNSH | 4 | 15| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Example Command Set

OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT,STB.
END FOR.
END.
The PROBE macro command has three forms:

1. PROBE.
2. PROBE,JAVSTEXT = <text name>.
3. PROBE,JAVSTEXT = <text name>, MODULE = <name-1>,<name-2>,..., <name-n>.

Each form of the PROBE macro command generates a series of JAVS standard commands which cause instrumentation to be performed.

The expansion of each form of the PROBE macro into JAVS standard commands is as follows:

1. JAVSTEXT = *NOJAVS*.
   FOR JAVSTEXT.
   INSTRUMENT.
   END FOR.
   PUNCH,JAVSTEXT = *NOJAVS*,INSTRUMENTED = ALL.

   This form (PROBE.) can be used for a library containing only one start-term sequence, if the JAVSTEXT identification (see Sec. 1.4) was not used. Instrumentation will be performed using the INSTRUMENT default options, and the instrumented source text will be written onto file LPUNCH.

2. JAVSTEXT = <text-name>.
   FOR JAVSTEXT.
   INSTRUMENT.
   END FOR.
   PUNCH,JAVSTEXT = <text name>,INSTRUMENTED = ALL.

   This form (PROBE,JAVSTEXT = <text name>.) should be used for instrumentation of a specified JAVSTEXT.

3. JAVSTEXT = <text name>.
   FOR MODULE = <name-1>,...,<name-n>.
   INSTRUMENT.
   END FOR.
   PUNCH,JAVSTEXT = <text name>,INSTRUMENTED = <name-1>,...,<name-n>.

   This form (PROBE,JAVSTEXT = <text name>, MODULE = <name-1>...) should be used if only selected modules of a given JAVSTEXT are to be instrumented.

* Can be used only with the overlay version of JAVS.
PROBE. (Cont.)

With all forms of the PROBE macro, the user can specify automatic insertion of the PROBI (test case initiation and test termination data collection routine) calls by preceding the PROBE command with:

\[
\text{PROBI,STARTTEST} = \langle\text{modname}\rangle,\langle\text{textname}\rangle,\langle\text{stmt. no.}\rangle,\langle\text{TESNAM}\rangle,\langle\text{TFLAG}\rangle.
\]

\[
\text{PROBI,STOPTEST} = \langle\text{modname}\rangle,\langle\text{textname}\rangle,\langle\text{stmt. no.}\rangle.
\]

These commands are described in Sec. 5 under their own heading. They perform the same function as the INSTRUMENT,STARTTEST and INSTRUMENT,STOPTEST commands. The existence of the PROBI commands aids the macro command processor which will insert the STARTTEST and STOPTEST commands in the generated series of commands.

Note

The macro command processor will generate the commands:

\[
\text{OLDER LIBRARY} = \text{TEST.}
\]

\[
\text{START.}
\]

if the first JAVS command is a macro command (keywords BUILD LIBRARY, DOCUMENT, PROBE, TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) See INSTRUMENT constraints in Sec. 3.4.

(2) Maximum of 150 modules selected in the first two forms of the PROBE macro command.

(3) Maximum of 23 modules selected in the third form of the PROBE macro command.

(4) Use PROBI (not INSTRUMENT,STARTTEST and STOPTEST) commands with the PROBE macro.

Example Command Sets

(1) BUILD LIBRARY.

\[
\text{PROBE, JAVSTEXT} = \text{EXPROM.}
\]

\[
\text{DOCUMENT.}
\]

\[
\text{PRINT, JAVSTEXT} = \text{EXPROM, INSTRUMENTED} = \text{ALL.}
\]

This command set will create a new library, perform syntax and structural analyses, instrument the JAVSTEXT EXPROM, document an entire library, and
print the instrumented JAVSTEXT so that the user can see where the PROBI calls should be manually placed.

(2) OLD LIBRARY = TEST.
START.
PROBI,STARTTEST = EXPROCM,EXPROCM,0.
PROBI,STARTTEST = EXMPL1,EXPROCM,9.
PROBI,STOPTEST = EXPROCM,EXPROCM,0.
PROBE,JAVSTEXT = EXPROCM.

This command set will instrument JAVSTEXT EXPROCM in the existing library TEST. In addition, the PROBI calls will be automatically inserted. The first PROBI,STARTTEST command will cause an invocation to PROBI to be inserted prior to the first executable statement in module EXPROCM. The second PROBI, STARTTEST command will cause a PROBI invocation to be placed immediately before statement 9 of module EXMPL1. These two PROBI invocations will initiate a new test case whenever the PROBI call is executed.

The PROBI,STOPTEST command will cause the test termination PROBI invocation to be placed at all exits from module EXPROCM. The PROBE macro command will instrument the JAVSTEXT EXPROCM and write the instrumented text, including the invocations to PROBI, onto file LPUNCH.

Note that the library definition and startup commands are included in this example. Even though the default library name is specified, these commands are required because PROBI is not a macro command.

5-100
PROBI,STARTTEST = <options>.

Description

This command performs the same function as:

INSTRUMENT,STARTTEST = <options>.

Refer to this heading for the command description and example command sets.

Rules

(1) This command is to be used only in conjunction with the PROBE macro command and must precede the PROBE command.

(2) Maximum of 10 PROBI,STARTTEST commands with a single PROBE macro command.

(3) Command options must be given in the order shown in the command description.

(4) Modules specified in PROBI,STARTTEST commands must be selected by the PROBE macro command.

(5) Library identification and startup commands must be included in the command set.

* Can be used only with the overlay version of JAWS.
PROBI,STOPTEST = <options>. *

Description

This command performs the same function as:

INSTRUMENT,STOPTEST = <options>.

Refer to this heading for the command description and example command sets.

Rules

(1) This command is to be used only in conjunction with the PROBE macro command and must precede the PROBE command.

(2) Only one PROBI,STOPTEST command with a single PROBE macro command.

(3) Command options must be given in the order shown in the command description.

(4) The module specified in the PROBI,STOPTEST command must be selected by the PROBE macro command.

(5) Library identification and startup commands must be included in the command set.

* Can be used only with the overlay version of JAVS.

5-102
**Description**

This command produces on LPUNCH in card image format the source statements for the named START-TERM text as they were originally stored on the library. The file can then be punched. The resulting cards are in normal JOVIAL format ready for compilation.

A listing of the cards resulting from this command for text EXPROGM is shown in the sample output. The statements are reformatted to a uniform structure with indentation.

**Rule**

File LPUNCH must be specified in the job control stream.

**Sample Output Excerpt**

```plaintext
## JVSTEX EXPROGM COMPUTE / EXCOMPL 1 ##
START
## JOVIAL SIMPLE TEST PROGRAM ##
ITEM ID = 4 S
ITEM ITER 1 2A S
ITEM ITER2 1 2A S
ITEM ITERA M = 5 S
ITEM ITERB M = 5 S
OVERLAY ITER I = ITER A S
OVERLAY ITER2 = ITER A S
ITEM CARD = 80 S
FILE READER M = 80 VEOF TAPES 5
FILE PRINTER M = 128 VEOF TAPES 5
MONITOR ID = ITERA, ITERB S
MESSAGE = MSG, S
OUTPUT PRINT MESSAGE S
##
BG.
INPUT READER CARD S
IF READER NO VEOF S
BEGIN
BYTE (0, 4 S) ( ID ) = BYTE (0, 4 S) ( CARD ) 5
BYTE (6, 4 S) ( ITERA ) = BYTE (6, 4 S) ( CARD ) 5
BYTE (6, 4 S) ( ITER2 ) = BYTE (6, 14, 4 S) ( CARD ) 5
EXAMPLE ( ITERA ) = ITER2 S
CLOSE EXAMPLE 5
## MAIN CLOSE ##
BEGIN
ITERA = 5 S
ITER2 = 15 S
END
## EXAMPLE ##
IF ITERA GT 100 S
GOTO EXAMPLE S
```

**Example Command Set**

```
OLD LIBRARY = REFMAN.
START.
PUNCH, JVSTEX = EXPROGM.
END.
```

5-103
PUNCH, JAVTEXT = <name>,
INSTRUMENTED = <option>.

Description

This command has two forms:

PUNCH, JAVTEXT = <text-name>, INSTRUMENTED = ALL, or
PUNCH, JAVTEXT = <text-name>, INSTRUMENTED = <name-1>, <name-2>, ..., <name-n>.

The command produces on LPUNCH in card image format the source statements for the named START-TERM text as instrumented by INSTRUMENT.* The file can then be punched. The resulting cards are in normal JOVIAL format ready for compilation.

The first form is used when all modules of the text are punched with instrumentation inserted. The second form is used when the specified modules in the text are to have instrumentation inserted and all other modules in the text are not instrumented. Any module which has not been processed by INSTRUMENT will be punched in uninstrumented, reformatted form. The modules must be listed in the same order in which they appear on the library.

A listing of the cards resulting from this command for text EXPROGM is shown in the sample output. The statements are reformatted to a uniform structure with indentation.

Rules

(1) Specified modules must be listed in the same order in which they appear in the library.

(2) File LPUNCH must be specified in the job control stream.

*The instrumented text consists of the source text interspersed with the probe text generated by INSTRUMENT.
Example Command Set

OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
PUNCH, MODULE.

Description

This command causes the module to be written on LPUNCH in card image format. The file can then be punched. The resulting cards are in normal JOVIAL format ready for insertion in a text for compilation. If the module is the only one in the START-TERM text, no additional manipulation of the punched text is necessary, since the START and TERM statements are included.

A listing of the cards resulting from this command for the module EXEMPL is shown in the sample output. The statements are reformatted to a uniform structure with indentation.

Rule

File LPUNCH must be specified in the job control stream.

Sample Output

```
** JAVESTT EXPROC COMPUTE (EXCOMPL)**
START
** JOVIAL SIMPLE TEST PROGRAM**
ITEM ID = 4 $ ITEM ITERA = 1 $ ITEM ITERA = 2 $ ITEM ITERA = 4 $ ITEM ITERA = 5 $ ITEM ITERA = 6 $ ITEM CARDS = 40 $ FILE HEADER = PLET (OK END TAPES 5 $ FILE PUNCH = PLET (OK END TAPES 5 $ MONITOR ID = ITERA + ITERA $ MESSAG $ 151 $ OUTPUT PRINTS MESSAG $ $60.
INPUT READER CARD $ IF READER NO VIECF $ RESIN BYTE (0 0 4 5 ) ( ID ) = BYTE (0 0 4 5 ) ( CARD ) $ BYTE (0 0 4 5 ) ( ITERA ) = BYTE (5 0 4 5 ) ( CARD ) $ BYTE (0 0 4 5 ) ( ITERA ) = BYTE (5 0 4 5 ) ( CARD ) $ EXEMPL ( ITERA ) = ITERA $ ** EXEMPL $ IF ITERA 03 00 $ GOTO EXAMPL $ GOTO 85 $ END
** IF $ STOP $** EXEMPL **
TERM $```

Example Command Set

OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PUNCH, MODULE.
END FOR.
END.

5-106
**STRUCTURAL.**

**Description**

The STRUCTURAL command with the print option at its default value produces the report output shown in the sample output. The actions performed by the STRUCTURAL command are:

1. To build tables describing the graphical characteristics of the specified module and add them to a library
2. To produce the report shown in the sample output

**Rule**

See STRUCTURAL constraints in Sec. 3.3.

**Sample Output**

```
JOVIAL AUTOMATED VERIFICATION SYSTEM *** SECONDARY MODULE ANALYSIS ***
MODULE EXMPL1 TEST EXPROGM.
MODULE DEPENDENCE TABLE CONSTRUCTED.
STATEMENT DESCRIPTOR BLOCKS UPDATED.
DD-PATH TABLE CONTAINS 20 ENTRIES.
```

**Example Command Set**

```
ALTER LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
STRUCTURAL.
END FOR.
END.
```
STRUCTURAL, PRINT = <option>.

Description

STRUCTURAL, PRINT = SUMMARY/DEBUG. (DEFAULT = SUMMARY.)

This command allows user control over the amount of information printed in the report produced during STRUCTURAL processing. Debug output which is useful to JAVS maintenance activities is generated with the command:

STRUCTURAL, PRINT = DEBUG.

Example Command Set

ALTER LIBRARY = REPMAN.
START.
FOR LIBRARY.
STRUCTURAL, PRINT = DEBUG.
STRUCTURAL.
END FOR.
END.

5-108
Description

The TEST macro command has two forms:

(1) TEST.

(2) TEST,MODULE = <name-1>,<name-2>,...,<name-n>.

The TEST macro command generates a series of JAVS ANALYZER commands for post-test analysis.

The expansion of the two forms of TEST is as follows:

(1) ANALYZER,ALL MODULES.
    ANALYZER,SUMMARY.
    ANALYZER,NOTHIT.
    ANALYZER,MODTRACE.
    ANALYZER.

(2) ANALYZER,MODULE = <name-1>,...,<name-n>.
    ANALYZER,SUMMARY.
    ANALYZER,NOTHIT.
    ANALYZER,MODTRACE.
    ANALYZER.

Additional post-test analysis reports can be produced by preceding the TEST macro with ANALYZER process option commands. In this case, library identification and startup commands must be included in the command set.

Note

The macro command processor will generate the commands:

OLD LIBRARY = TEST.
START.

if the first JAVS command is a macro command (keywords BUILD LIBRARY, DOCUMENT, PROBE, TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) The ANALYZER,DDPTRACE standard command cannot be used in conjunction with the TEST macro.

*Can be used only with the overlay version of JAVS.
TEST. (Cont.)

(2) ANALYZER option commands must precede the TEST macro, if they are used.

(3) See ANALYZER constraints in Sec. 3.3.

Example Command Sets

(1) TEST.
   FOR LIBRARY.
   PRINT, DDF.
   END FOR.

(2) OLD LIBRARY = REFMAN.
   START.
   ANALYZER, MODLST.
   ANALYZER, DDFPATHS.
   TEST, MODULE = EXPRGM, EXMPL1.
APPENDIX A

JOVIAL KEYWORDS RECOGNIZED BY JAVS
The JOVIAL keywords recognized by JAVS are:

ABS  |  LQ  
ALL  |  LS  
AND  |  MANT  
ARRAY |  MODE  
ASSIGN |  MONITOR (2,3)  
BEGIN |  NEXT  
BIT  |  NOT  
BYTE  |  NO  
CHAR  |  NODSEN  
CLOSE  |  ODD  
COMMON  |  OPEN  
COMPOOL (1)  |  OR  
DEFINE  |  ORIF  
DIRECT  |  OUT (1)  
END  |  OUTPUT  
ENT  |  OVERLAY  
ENTRY  |  POS  
EQ  |  PROC  
FALSE (1)  |  PROGRAM  
FILE  |  REM  
FOR  |  REMQUO  
GOTO  |  RETURN  
GQ  |  SHUT  
GR  |  START  
IF  |  STOP  
IFEITH  |  STRING  
IN (1)  |  SWITCH  
INPUT  |  TABLE  
IO (1)  |  TERM  
ITEM  |  TEST  
JOVIAL  |  TRUE (1)  
LOC  |  WAIT (1)

NOTES: (1) Honeywell JOVIAL keyword; (2) CDC JOVIAL keyword; (3) JOGIT JOVIAL keyword.
APPENDIX B
ERROR MESSAGES
### 3.1 Universal Command Error Messages

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library opened read/write with improper password</td>
<td>Library name for ALTER library must be the same as when created. Change library name and rerun.</td>
</tr>
<tr>
<td>No information on read only library</td>
<td>Provide already created library profile or remove old library command.</td>
</tr>
<tr>
<td>Procedure invoked with module equal to zero</td>
<td>Specify a module on the library and rerun.</td>
</tr>
<tr>
<td>Too many error messages</td>
<td>Make correction for preceding error messages and rerun.</td>
</tr>
<tr>
<td>No JAVSTEXT specified</td>
<td>Specify JAVSTEXT and rerun.</td>
</tr>
</tbody>
</table>
B.2 BASIC ERROR MESSAGES

The error messages* resulting from BASIC source code analysis are:

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The first symbol in the JOVIAL source program is not a START or a CLOSE. A START is assumed and processing continues.</td>
</tr>
<tr>
<td>1</td>
<td>A single prime which is not a legal symbol in a JOVIAL source program has been encountered. The prime is ignored.</td>
</tr>
<tr>
<td>2</td>
<td>The constant has an illegal format. The resulting constant may not compile.</td>
</tr>
<tr>
<td>3</td>
<td>A comment is terminated by a dollar sign. The first symbol following the dollar sign is considered to be the beginning of the next statement or declaration.</td>
</tr>
<tr>
<td>4</td>
<td>The define information overflows one of the tables provided for it. As a result, this define directive and all subsequent define directives may be ignored.</td>
</tr>
<tr>
<td>5</td>
<td>The preset data constant conflicts with the definition of the item with which it is associated. The constant is processed as if it were legal.</td>
</tr>
<tr>
<td>6</td>
<td>The symbol encountered in a preset data list or as a single preset data constant is neither a constant nor an END. If the illegal symbol occurs as the first constant in a list or in the place of a single preset data constant, the symbol is changed to the constant zero. If the illegal symbol occurs within a preset data list, it is ignored. If the next symbol in the list is a constant or an END, the processing of the preset data list is continued. If not, the processing is discontinued and the next symbol is considered to be part of the next statement or declaration.</td>
</tr>
<tr>
<td>7</td>
<td>An illegal character has been found in the JOVIAL source program. The character is treated as a blank.</td>
</tr>
<tr>
<td>8</td>
<td>An end-of-file has been encountered on the input device, but no TERM $ has been found. A TERM $ is assumed and processing continues.</td>
</tr>
<tr>
<td>10</td>
<td>The symbol which begins the statement or declaration is not legal at the beginning of a statement or declaration. The JOVIAL source program is scanned until a symbol legal at the beginning of a statement or declaration is encountered.</td>
</tr>
</tbody>
</table>

* BASIC error messages differ from JOVIAL compiler error messages. Many errors can be treated as warnings.
Error Number | Explanation
--- | ---
11 | A statement or a declaration which is not legal within a table declaration has been encountered. The processing of the table is terminated. If there are no other declarations for the table, it is removed from the dictionary.* Then the current statement or declaration is processed normally.
12 | An E- or A-factor for a constant contains more than the number of numerals allowed for it. The number is truncated on the left.
13 | The name is missing from a close, program, or switch declaration. The declaration is ignored.
14 | A simple item has been defined by a status constant. An incomplete definition for the item is left in the dictionary.
15 | A name other than a sequence designator, simple item, or procedure has been used without being previously declared. This error message is printed only when no COMPOOL has been specified. The name is entered in the dictionary with the standard mode definition, unless it is a table name.
16 | There is a syntax error in the switch declaration. The rest of the declaration is ignored.
17 | The name appearing in an OPEN or SHUT is not a FILE name.
18 | The FILE name does not appear in the proper place in an I/O statement.
19 | A constant is succeeded by a loop variable, abbreviation, primitive or name with no intervening blanks. A blank is assumed.
20 | A procedure declaration has been encountered within another procedure declaration. The previous procedure is terminated.
21 | There is a syntax error in the heading of a procedure declaration. The rest of the symbols up to the first dollar sign are ignored.

*BASIC makes use of an internal dictionary from which JAVS tables are built.
<table>
<thead>
<tr>
<th>Error Number</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>A constant has been found in a subordinate overlay declaration (overlay declaration within a table declaration). The rest of the overlay declaration is ignored.</td>
</tr>
<tr>
<td>23</td>
<td>A syntax error has been found in an overlay declaration. The rest of the statement is ignored.</td>
</tr>
<tr>
<td>24</td>
<td>The external file name is not a constant or a symbol.</td>
</tr>
<tr>
<td>25</td>
<td>A syntax error has been found in a define directive. The rest of the directive is ignored.</td>
</tr>
<tr>
<td>26</td>
<td>The name is missing from a simple item, table item, or array declaration. The rest of the declaration is ignored.</td>
</tr>
<tr>
<td>27</td>
<td>A STRING statement appears outside a TABLE statement sequence.</td>
</tr>
<tr>
<td>28</td>
<td>The dimension information for an array overflows the dimension table. The rest of the array declaration is ignored.</td>
</tr>
<tr>
<td>30</td>
<td>Two contiguous symbols in the statement are not legal placed next to one another. The left symbol has been ignored.</td>
</tr>
<tr>
<td>31</td>
<td>A possible missing dollar sign has been detected. A dollar sign has been inserted, the statement counter incremented, and the processing of the JOVIAL source program is continued.</td>
</tr>
<tr>
<td>32</td>
<td>A symbol has been detected as illegal inside of the current level of bracketers (parentheses, brackets, exponentiation brackets, or absolute value brackets). The statement is processed as though no error were found.</td>
</tr>
<tr>
<td>33</td>
<td>A constant followed by an equals sign has been encountered outside of procedure parentheses or switch parentheses. For example:</td>
</tr>
<tr>
<td></td>
<td>IF 10 = AA $</td>
</tr>
<tr>
<td>34</td>
<td>A period followed by a right parenthesis has been encountered outside of a 'LOC or procedure parentheses, or a period followed by a comma or an equals sign has been encountered outside of procedure parentheses. If a name preceded the period, it has been treated like a statement name. For example:</td>
</tr>
<tr>
<td></td>
<td>AA = AA-(CC+DD.) $</td>
</tr>
<tr>
<td>Error Number</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>35</td>
<td>When the end of the statement was reached, the count of brackets in the statement had not reached zero. That is, there were more left brackets than right brackets in the statement.</td>
</tr>
<tr>
<td>36</td>
<td>A right bracket does not match the appropriate left bracket. Error messages may be printed for the remaining right brackets in the statement.</td>
</tr>
<tr>
<td>37</td>
<td>Brackets in the statement are nested more than 24 deep. The processing of the statement has been terminated and the statement counter incremented.</td>
</tr>
<tr>
<td>38</td>
<td>A dollar sign has been recognized as the left term in a pair of symbols. This dollar sign is illegal but it is assumed that it terminates the statement. The statement counter is incremented.</td>
</tr>
<tr>
<td>40</td>
<td>The accumulator designator &quot;A(&quot; has been encountered outside of a direct statement. It is processed as if there were no error.</td>
</tr>
<tr>
<td>41</td>
<td>The entity being declared has the same name as a previously declared entity with overlapping scope and domain. The declaration is processed as though no error has occurred.</td>
</tr>
<tr>
<td>42</td>
<td>The entity being declared has the same name as a previously used entity with overlapping scope and domain. The declaration is processed as though no error has occurred.</td>
</tr>
<tr>
<td>43</td>
<td>A status constant has been used incorrectly in the statement. This may result in the integer constant 0 being substituted for the status constant.</td>
</tr>
<tr>
<td>45</td>
<td>The table containing the names of COMPOOL-defined status variables has overflowed. That is, too many COMPOOL-defined status variables are being used with a total of too many constants. Subsequent usage of new variables or old variables used with new constants will elicit this same error message and will not be handled correctly.</td>
</tr>
<tr>
<td>46</td>
<td>The symbol following the FOR in a FOR-clause is not a loop variable. The clause is processed normally.</td>
</tr>
<tr>
<td>47</td>
<td>The dictionary has overflowed. That is, the JOVIAL source program contains too many constants and names. Processing of the program has been terminated.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 48           | a. A name that is not a sequence designator is being declared, and is found to be the same as a name that is already in the dictionary.  
|              | or  
<p>|              | b. A name is being used in a way that is inconsistent with its existing dictionary definition. The name is assumed to be the same as the one in the dictionary. |
| 49           | A global simple item has the same name as a procedure. The simple item is processed as though no error has occurred. |
| 50           | No dimension information has been given for an array encountered. The preset data are ignored. |
| 51           | More than three levels of preset data for an array declaration. The rest of the declaration is processed normally. |
| 52           | The preset data list for an array does not have the same number of dimensions as the array. The preset data have been ignored. |
| 53           | The count of BEGIN-END's for the preset data for an array or string is incorrect since a symbol illegal within a preset data list has been encountered. The processing of the preset data list is terminated. |
| 54           | There is a syntax error in a table declaration. The rest of the declaration is processed in as normal a manner as possible. |
| 55           | The number of like tables exceeds 26. |
| 56           | A syntax error has been encountered in a simple item, table item, or array declaration. The processing of the declaration is terminated. |
| 57           | The status list is missing from a status item declaration. The rest of the declaration is processed normally. |
| 58           | A table item declaration does not have the correct format for a defined-entry table. The information in the dictionary for the table item will be incorrect. |
| 59           | The overlay information overflows the overlay table. The rest of this overlay declaration is ignored. |</p>
<table>
<thead>
<tr>
<th>Error Number</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>A JOVIAL primitive which is not implemented in the JAVS dialect has been encountered. (See Appendix A.) The primitive is ignored.</td>
</tr>
<tr>
<td>62</td>
<td>A subsequent constant in a preset data list is not compatible with the initial constant in the list. The constant is processed normally.</td>
</tr>
<tr>
<td>64</td>
<td>No dollar sign was found at the end of a statement. A dollar sign was assumed, however.</td>
</tr>
<tr>
<td>65</td>
<td>No dollar sign was found at the end of a declaration. A dollar sign was assumed.</td>
</tr>
<tr>
<td>66</td>
<td>A TERM or end-of-file on the input device was encountered within a table declaration. Processing of the JOVIAL source program is discontinued.</td>
</tr>
<tr>
<td>67</td>
<td>A TERM or end-of-file on the input device was encountered within a procedure. ENDS are added for the procedure. Then the processing of the JOVIAL source program is discontinued.</td>
</tr>
<tr>
<td>68</td>
<td>The BEGIN-END count for the main program was not zero when the TERM or end-of-file on the input device was encountered. ENDS are added for the main program. Then the processing of the JOVIAL source program is discontinued.</td>
</tr>
<tr>
<td>70</td>
<td>A name or a constant overflows the number of characters allowed for it (120 is the maximum number of characters allowed by most compilers). The excess characters have been deleted from the name or constant.</td>
</tr>
<tr>
<td>71</td>
<td>There are too many characters in the names and constants used by the JOVIAL source program. Processing of the JOVIAL source program is discontinued.</td>
</tr>
<tr>
<td>73</td>
<td>The number of preset data constants in a list is greater than the maximum number of entries in the table. The excess number of constants has been deleted from the list.</td>
</tr>
<tr>
<td>74</td>
<td>A preset data constant (P followed by a constant) was found in an array or table item declaration. The constant was processed normally.</td>
</tr>
<tr>
<td>75</td>
<td>An illegal double period has been encountered. A double period is legal only when there are two consecutive floating constants (example: 2..2 which is the same as 2. .2). The second period is processed like a numeral.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>76</td>
<td>A status constant was not found in the status list for the status variable with which it is associated. The value of the constant is set to zero.</td>
</tr>
<tr>
<td>77</td>
<td>The statement begins with a dollar sign, and the statement counter is incremented by one.</td>
</tr>
<tr>
<td>78</td>
<td>A table declaration with no items has been encountered. The table is removed from the dictionary.</td>
</tr>
<tr>
<td>80</td>
<td>A local simple item has the same name as the procedure in which it is being used. This item has no local declaration, but it is assumed that it defines the procedure as a function.</td>
</tr>
<tr>
<td>81</td>
<td>A local procedure call references a procedure with the same name as the procedure in which it is being used. This is a recursive use of the procedure name. The two procedure names are assumed to refer to the same entity in the dictionary.</td>
</tr>
<tr>
<td>82</td>
<td>A name used in an overlay declaration or as an actual input or output parameters for a procedure has the same name as a table item. The two variables are assumed to have the same entry in the dictionary.</td>
</tr>
<tr>
<td>83</td>
<td>A name not identified as to class (i.e., may be a simple item, table item, etc.) has the same name as a procedure with overlapping scope. The variable is processed as though no error has occurred.</td>
</tr>
<tr>
<td>84</td>
<td>A table item or table has the same name as a procedure with overlapping scope. The table item or table is processed as though no error has occurred.</td>
</tr>
<tr>
<td>85</td>
<td>A local table or array has the same name as the procedure in which it is declared. The table or array is processed as though no error has occurred.</td>
</tr>
<tr>
<td>86</td>
<td>A procedure has the same name as another global variable. The procedure name is processed as though no error has occurred.</td>
</tr>
<tr>
<td>87</td>
<td>A FILE name matches a PROC name.</td>
</tr>
<tr>
<td>88</td>
<td>An impossible entry in the variable or label identification matrix has been encountered. This indicates either a compiler or machine error.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>92</td>
<td>There is a preset data constant for a formal input or output parameter for a procedure. The preset data constant is ignored.</td>
</tr>
<tr>
<td>94</td>
<td>There are too many rows in the preset data list for an array. The excess columns have been deleted.</td>
</tr>
<tr>
<td>95</td>
<td>There are too many planes in the preset data list for an array. All of the rest of the preset data for the array is ignored.</td>
</tr>
<tr>
<td>97</td>
<td>The preset data for a string statement is too long.</td>
</tr>
<tr>
<td>98</td>
<td>A BEGIN is missing in preset data.</td>
</tr>
<tr>
<td>99</td>
<td>A procedure has been used inconsistently. It has been declared and/or used both as a procedure and a function. The final dictionary definition will reflect the way in which the procedure was declared.</td>
</tr>
<tr>
<td>851</td>
<td>More than 3000 names and constants appear in JOVIAL source being processed. Program must be broken down for BASIC analysis.</td>
</tr>
<tr>
<td>853</td>
<td>A JAVSTEXT directive does not correspond to text type.</td>
</tr>
<tr>
<td>900</td>
<td>A token has been misclassified in BASIC internal processing.</td>
</tr>
<tr>
<td>901</td>
<td>More than 53 nested modules appear in JOVIAL source stream. The program must be broken down for BASIC analysis.</td>
</tr>
<tr>
<td>941</td>
<td>Module is not a program, procedure, close, COMPOOL, or JAVSTEXT.</td>
</tr>
<tr>
<td>950</td>
<td>A CLOSE statement occurs in an illegal context.</td>
</tr>
<tr>
<td>984</td>
<td>More than 250 word pairs appear in token. The token is omitted from SB.</td>
</tr>
<tr>
<td>985</td>
<td>A COMMON statement not terminated with $. A $ is assumed.</td>
</tr>
<tr>
<td>986</td>
<td>An illegal comment in the same form as a JAVS directive has been encountered. The token is not entered in the SB table.</td>
</tr>
<tr>
<td>988</td>
<td>More than 100 names appear in the statement. The statement must be broken down for BASIC processing.</td>
</tr>
<tr>
<td>990</td>
<td>The statement has an unrecognizable class value.</td>
</tr>
<tr>
<td>Error Number</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>997</td>
<td>Input parameter error.</td>
</tr>
<tr>
<td>999</td>
<td>An unknown entry type has been encountered in the internal dictionary.</td>
</tr>
</tbody>
</table>
### B.3 STRUCTURAL ERROR MESSAGES

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad statement type on stack</td>
<td>Check syntax of module. Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>BEGIN-END block contains no statement</td>
<td>Informational message only</td>
</tr>
<tr>
<td>Duplicate symbol which is not a local label</td>
<td>Duplicate symbol in STB treated as a global label. No rerun necessary</td>
</tr>
<tr>
<td>END does not match BEGIN, IFEITH or FOR</td>
<td>Check syntax of module. Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>External array has overflowed</td>
<td>Reduce number of externals to less than 100. Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>External switch call treated as return</td>
<td>Declare switch in module and rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>FOR statement not in stack</td>
<td>Rerun STRUCTURAL</td>
</tr>
<tr>
<td>FOR target pointer error</td>
<td>Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>FOR variable not found in SB</td>
<td>Check syntax of FOR statement. Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>GOTO label not found</td>
<td>Check syntax of GOTO statement and rerun BASIC and STRUCTURAL on module</td>
</tr>
<tr>
<td>Illegal statement type on stack</td>
<td>Rerun STRUCTURAL</td>
</tr>
<tr>
<td>Invalid statement in IFEITH/ORIF</td>
<td>Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>Invalid token type in switch declaration</td>
<td>Check syntax of switch declaration and rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>JOVIAL statement missing</td>
<td>Check syntax of module for DIRECT-JOVIAL statements. Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>Module number &lt;= 0</td>
<td>Specify a module on the library. Rerun STRUCTURAL</td>
</tr>
<tr>
<td>Error Message</td>
<td>Corrective Action</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Nested switch call treated as fall through case</td>
<td>Replace switch invocation in switch declaration by a label and rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>Never found PROC statement</td>
<td>JOVIAL procedure must contain PROC statement. Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>Pointer LQ 0</td>
<td>Rerun STRUCTURAL</td>
</tr>
<tr>
<td>Possible infinite loop detected</td>
<td>Rewrite module to remove infinite loop. Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>Stack overflow</td>
<td>Increase the stack size by recompiling STRUCTURAL. Rerun STRUCTURAL.</td>
</tr>
<tr>
<td>Stack underflow</td>
<td>Rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>Switch name missing</td>
<td>Check SWITCH syntax and rerun BASIC and STRUCTURAL</td>
</tr>
<tr>
<td>Test stack overflow</td>
<td>Increase the size of the test stack by recompiling STRUCTURAL. Rerun STRUCTURAL.</td>
</tr>
<tr>
<td>Too many paths begin at statement</td>
<td>Increase the size of the DD-path queue by recompiling STRUCTURAL. Rerun STRUCTURAL</td>
</tr>
<tr>
<td>Too many statements on DD-path</td>
<td>Increase the size of the DD-path statement list by recompiling STRUCTURAL. Rerun STRUCTURAL</td>
</tr>
<tr>
<td>Undefined GOTO</td>
<td>Check location of label referenced in GOTO. Rerun BASIC and STRUCTURAL</td>
</tr>
</tbody>
</table>
B.4 INSTRUMENT ERROR MESSAGES

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad module number</td>
<td>Specify a module number within the library. Rerun INSTRUMENT</td>
</tr>
<tr>
<td>BEGIN-END block contains no statements</td>
<td>Informative message</td>
</tr>
<tr>
<td>Could not match parentheses</td>
<td>Check syntax of statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT</td>
</tr>
<tr>
<td>Delimiter stack overflow</td>
<td>Increase the size of the delimiter stack by recompiling INSTRUMENT. Rerun INSTRUMENT.</td>
</tr>
<tr>
<td>Directive not recognized</td>
<td>Check syntax of directive. Rerun BASIC, STRUCTURAL and INSTRUMENT</td>
</tr>
<tr>
<td>External switch call not instrumented</td>
<td>Declare switch in module. Rerun BASIC, STRUCTURAL, and INSTRUMENT</td>
</tr>
<tr>
<td>GOTO label not found</td>
<td>Check syntax of GOTO statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT</td>
</tr>
<tr>
<td>Input string too long—truncated from head</td>
<td>Change size of internal buffers by recompiling INSTRUMENT. Rerun INSTRUMENT</td>
</tr>
<tr>
<td>Invalid statement type in chain</td>
<td>Check syntax of IFEITH statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT</td>
</tr>
<tr>
<td>Invalid switch type</td>
<td>Check syntax of switch declaration. Rerun BASIC, STRUCTURAL, and INSTRUMENT</td>
</tr>
<tr>
<td>Last statement in module is not TERM or END</td>
<td>Add a TERM or END statement to the module and rerun BASIC, STRUCTURAL and INSTRUMENT</td>
</tr>
<tr>
<td>No DD-path for statement</td>
<td>Rerun STRUCTURAL and INSTRUMENT</td>
</tr>
<tr>
<td>Number of switch labels &lt;= 0</td>
<td>Rerun STRUCTURAL and INSTRUMENT</td>
</tr>
<tr>
<td>ORIF statement number not on false branch stack</td>
<td>Rerun INSTRUMENT</td>
</tr>
<tr>
<td>Pointer out of range</td>
<td>Rerun STRUCTURAL and INSTRUMENT</td>
</tr>
<tr>
<td>Error Message</td>
<td>Corrective Action</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Stack overflow</td>
<td>Increase the stack size by recompiling INSTRUMENT. Rerun INSTRUMENT</td>
</tr>
<tr>
<td>Stack underflow</td>
<td>Rerun INSTRUMENT</td>
</tr>
<tr>
<td>String longer than text buffer</td>
<td>Increase the size of the text buffer by recompiling INSTRUMENT. Rerun INSTRUMENT</td>
</tr>
<tr>
<td>Unable to recognize type of FOR statement</td>
<td>Check syntax of FOR statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT</td>
</tr>
<tr>
<td>Variable not declared in preceding TRACE directive</td>
<td>Declare variable in TRACE directive. Rerun INSTRUMENT</td>
</tr>
</tbody>
</table>
### 3.5 ASSIST ERROR MESSAGES

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beginning DDP outside range of DD-paths</td>
<td>Incorrect command. Rerun ASSIST</td>
</tr>
<tr>
<td>Target DDP outside range of DD-paths</td>
<td>Incorrect command. Rerun ASSIST</td>
</tr>
<tr>
<td>Too many DD-paths</td>
<td>Select smaller reaching set. Rerun ASSIST</td>
</tr>
<tr>
<td>Too many DD-paths in reaching set</td>
<td>Select smaller reaching set. Rerun ASSIST</td>
</tr>
<tr>
<td>Too many essential predicates</td>
<td>Select smaller reaching set. Rerun ASSIST</td>
</tr>
</tbody>
</table>
### DEPENDENCE ERROR MESSAGES

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too many called modules from group</td>
<td>Select smaller group. Rerun DEPENDENCE</td>
</tr>
<tr>
<td>Too many calling modules to group</td>
<td>Select smaller group. Rerun DEPENDENCE</td>
</tr>
<tr>
<td>Too many outside modules called from library</td>
<td>Select group off library. Rerun DEPENDENCE</td>
</tr>
</tbody>
</table>
### App. B

#### 3.7 ANALYZER ERROR MESSAGES

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illegal record on AUDIT file</td>
<td>Check AUDIT file for 4H(TERM) in last record. If not present, test execution did not invoke the test termination PROBI module.</td>
</tr>
<tr>
<td>Unknown module specified</td>
<td>Specify only those modules on the library. Rerun ANALYZER.</td>
</tr>
<tr>
<td>Bad module value</td>
<td>Check AUDIT file for 4H(TERM) in last record. If not present, test execution did not invoke the test termination PROBI module. Check for use of previously written AUDIT file. Insert test termination PROBI call in instrumented code where it will be invoked during test.</td>
</tr>
</tbody>
</table>
B.8 OTHER JAVS SYSTEM ERRORS

Error Message

ACTMOD does not know an active module is
Bad call to PUTBIT
Bad module value
Beginning block of search is out of table range
Block does not exist for table
Changed module not yet on new library
Did not know module was active
Entry point tables for both libraries are empty
Fragment beyond range
Fragment number given as zero
GETLST called for non-variable length table
Illegal block number
Illegal core address
Illegal core fragment number
Illegal library address
Illegal library fragment number
Illegal library number
Illegal table number
Improper call to PREPT
Index beyond block size

*The error messages listed in this section are for internal JAVS errors indicative of flaws in JAVS software over which the user has no control.
Error Message

IRSTAB cannot be used with variable length table
Library opened with improper fragment size
LSETUP bad record length
MDBGET can only be used with active modules
MDB not on library
Module location table has been overflowed
Module must be active to change
Module not in core to dump
Module number not yet assigned
Module outside legal range
No more table space available
No such block
No such fragment
Not yet implemented
PUTLST called for non-variable length table
Selected block out of known range
Selected block out of table bounds
Table not known in system
Too many items in variable list
APPENDIX C

STATEMENT TYPE ABBREVIATIONS
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRY</td>
<td>ARRAY</td>
<td>INPT</td>
<td>INPUT</td>
</tr>
<tr>
<td>ASMT</td>
<td>ASSIGNMENT</td>
<td>INSW</td>
<td>INDEX SWITCH DECLARATION</td>
</tr>
<tr>
<td>ASSN</td>
<td>ASSIGN</td>
<td>INV</td>
<td>PROCEDURE INVOCATION</td>
</tr>
<tr>
<td>ASSY</td>
<td>ASSEMBLY CODE</td>
<td>ITEM</td>
<td>ITEM</td>
</tr>
<tr>
<td>BECN</td>
<td>BEGIN</td>
<td>ITRW</td>
<td>ITEM SWITCH DECLARATION</td>
</tr>
<tr>
<td>CLOS</td>
<td>CLOSE</td>
<td>JOVL</td>
<td>JOVIAL</td>
</tr>
<tr>
<td>COVTN</td>
<td>COMMON</td>
<td>LABL</td>
<td>LABEL</td>
</tr>
<tr>
<td>COMT</td>
<td>COMMENT</td>
<td>MODE</td>
<td>MODE</td>
</tr>
<tr>
<td>CONT</td>
<td>CONTINUATION</td>
<td>MNTR</td>
<td>MONITOR</td>
</tr>
<tr>
<td>CNPL</td>
<td>COMPOOL</td>
<td>OPEN</td>
<td>OPEN</td>
</tr>
<tr>
<td>DEFN</td>
<td>DEFINE</td>
<td>ORIF</td>
<td>ORIF</td>
</tr>
<tr>
<td>DIRC</td>
<td>DIRECT</td>
<td>OUT</td>
<td>OUT</td>
</tr>
<tr>
<td>DIRT</td>
<td>JAVA'S DIRECTIVE</td>
<td>OUTF</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>END</td>
<td>END</td>
<td>OVRL</td>
<td>OVERLAY</td>
</tr>
<tr>
<td>EXCH</td>
<td>EXCHANGE</td>
<td>PROB</td>
<td>INSTRUMENTATION PROBE</td>
</tr>
<tr>
<td>FILE</td>
<td>FILE</td>
<td>PROC</td>
<td>PROCEDURE</td>
</tr>
<tr>
<td>FOR1</td>
<td>1 FACTOR FOR CLAUSE</td>
<td>PROG</td>
<td>PROGRAM</td>
</tr>
<tr>
<td>FOR2</td>
<td>2 FACTOR FOR CLAUSE</td>
<td>PRST</td>
<td>PRESET LIST</td>
</tr>
<tr>
<td>FOR3</td>
<td>3 FACTOR FOR CLAUSE</td>
<td>REIT</td>
<td>RETURN</td>
</tr>
<tr>
<td>FUNC</td>
<td>FUNCTION</td>
<td>SHUT</td>
<td>SHUT</td>
</tr>
<tr>
<td>GOTO</td>
<td>GOTO LABEL</td>
<td>STRG</td>
<td>STRING</td>
</tr>
<tr>
<td>GOTOCL</td>
<td>GOTO CLOSE</td>
<td>STRT</td>
<td>START</td>
</tr>
<tr>
<td>GTSW</td>
<td>GOTO SWITCH</td>
<td>STOP</td>
<td>STOP</td>
</tr>
<tr>
<td>GTXE</td>
<td>GOTO EXTERNAL LABEL</td>
<td>SWTC</td>
<td>SWITCH</td>
</tr>
<tr>
<td></td>
<td>(RETURN)</td>
<td>TABL</td>
<td>TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TBLI</td>
<td>TABLE ITEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TERM</td>
<td>TERM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TEST</td>
<td>TEST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WAIT</td>
<td>WAIT</td>
</tr>
</tbody>
</table>

C-2
## INDEX

<table>
<thead>
<tr>
<th>Actual parameters</th>
<th>Report</th>
<th>5-54</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYZER</td>
<td>Constraints</td>
<td>3-9</td>
</tr>
<tr>
<td></td>
<td>Coverage</td>
<td>2-19</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>1-2, 2-1</td>
</tr>
<tr>
<td></td>
<td>Option commands</td>
<td>4-11, 4-12</td>
</tr>
<tr>
<td></td>
<td>Processing</td>
<td>2-20</td>
</tr>
<tr>
<td></td>
<td>Purpose</td>
<td>2-19</td>
</tr>
<tr>
<td></td>
<td>Tracing</td>
<td>2-19</td>
</tr>
<tr>
<td>ASSERT directive</td>
<td></td>
<td>1-11</td>
</tr>
<tr>
<td>ASSIST</td>
<td>Constraints</td>
<td>3-6</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>1-2, 2-1</td>
</tr>
<tr>
<td></td>
<td>Processing</td>
<td>2-9</td>
</tr>
<tr>
<td>AUDIT file</td>
<td>Declaration</td>
<td>1-4, 2-13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-14</td>
</tr>
<tr>
<td>BASIC</td>
<td>Constraints</td>
<td>3-3</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>1-2, 2-1</td>
</tr>
<tr>
<td></td>
<td>Option commands</td>
<td>4-10</td>
</tr>
<tr>
<td></td>
<td>Processing</td>
<td>2-2</td>
</tr>
<tr>
<td>Card image listing</td>
<td></td>
<td>5-36</td>
</tr>
<tr>
<td>Code expansion</td>
<td></td>
<td>2-18</td>
</tr>
<tr>
<td>Commands</td>
<td>Command language</td>
<td>4-1, 5-1</td>
</tr>
<tr>
<td></td>
<td>Keywords</td>
<td>1-5, 1-6</td>
</tr>
<tr>
<td></td>
<td>Library</td>
<td>4-2</td>
</tr>
<tr>
<td></td>
<td>Module selection</td>
<td>4-1, 4-8, 4-9</td>
</tr>
<tr>
<td></td>
<td>Overlay commands</td>
<td>5-3, 5-6</td>
</tr>
<tr>
<td></td>
<td>Print/punch</td>
<td>4-3, 4-15</td>
</tr>
<tr>
<td></td>
<td>Process execution</td>
<td>4-3, 4-10, 4-13</td>
</tr>
<tr>
<td></td>
<td>Process option</td>
<td>4-1</td>
</tr>
<tr>
<td></td>
<td>Startup</td>
<td>4-1, 4-7</td>
</tr>
<tr>
<td></td>
<td>Termination</td>
<td>5-3, 5-4</td>
</tr>
<tr>
<td></td>
<td>Universal</td>
<td>1-5</td>
</tr>
<tr>
<td>Comments (in source text)</td>
<td></td>
<td>5-37</td>
</tr>
<tr>
<td>COMPOOL</td>
<td>Data Collection routines compool</td>
<td>2-14</td>
</tr>
<tr>
<td></td>
<td>Identification</td>
<td>1-10</td>
</tr>
<tr>
<td></td>
<td>Processing</td>
<td>5-41</td>
</tr>
</tbody>
</table>
**INDEX (Cont.)**

<p>| Computation directives                                      | 1-11, 2-8 |
| Instruments                                                | 5-70     |
| Performance data                                           | 2-11     |
| Constraints                                                | 3-9      |
| ANALYZER                                                   | 3-6      |
| ASSIST                                                     | 3-3      |
| BASIC                                                      | 3-8      |
| DEPENDENCE                                                 | 3-5      |
| INSTRUMENT                                                 | 3-4      |
| STRUCTURAL                                                 | 3-7      |
| Test Execution                                             | 3-2      |
| Universal                                                  |          |
| Coverage (Test)                                            | 5-10     |
| DD-path                                                    | 5-15     |
| Statement                                                  |          |
| Cross reference                                            | 5-24     |
| Database library                                           | 1-4      |
| Definition                                                 | 2-1      |
| Utilization                                                |          |
| Data collection                                            | 2-11, 2-13, 5-72, 5-73, 5-74, 2-13 |
| Routines                                                   | 1-3      |
| PROBI                                                      | 2-13     |
| Data flow analysis                                         |          |
| ASSERT directive                                           | 1-11     |
| EXPECT directive                                           |          |
| OFFTRACE directive                                         | 1-12     |
| TRACE directive                                            | 1-12, 1-13 |
| DD-path                                                    | 2-4      |
| Definition                                                 | 2-5, 2-6 |
| Example                                                    |          |
| DEPENDENCE                                                 | 1-2, 2-1 |
| Function                                                   |          |
| DEFINE variables                                           | 5-38     |
| DOCUMENT                                                   | 5-59     |
| Error messages                                             | 8-1      |
| Example program                                            | 1-7, 1-8 |
| I-2                                                        |          |</p>
<table>
<thead>
<tr>
<th>INDEX - (Cont.)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
<td>5-22</td>
</tr>
<tr>
<td>EXPECT directive</td>
<td>1-12</td>
</tr>
<tr>
<td>Files</td>
<td></td>
</tr>
<tr>
<td>Identification</td>
<td>1-4</td>
</tr>
<tr>
<td>Formal parameters</td>
<td></td>
</tr>
<tr>
<td>Report</td>
<td>5-54</td>
</tr>
<tr>
<td>INSTRUMENT</td>
<td></td>
</tr>
<tr>
<td>Code expansion</td>
<td>2-18</td>
</tr>
<tr>
<td>Constraints</td>
<td>3-5</td>
</tr>
<tr>
<td>Directive instrumentation</td>
<td>2-8, 2-11</td>
</tr>
<tr>
<td>Function</td>
<td>1-2</td>
</tr>
<tr>
<td>Option commands</td>
<td>4-11</td>
</tr>
<tr>
<td>Processing</td>
<td>2-7</td>
</tr>
<tr>
<td>Structural instrumentation</td>
<td>2-8</td>
</tr>
<tr>
<td>Inter-module dependencies</td>
<td>2-10, 5-46, 5-48, 5-50</td>
</tr>
<tr>
<td>JAVS overlay</td>
<td>1-4, 1-5, 1-6, 2-21</td>
</tr>
<tr>
<td>JAVTEXT</td>
<td></td>
</tr>
<tr>
<td>Directives</td>
<td>1-10</td>
</tr>
<tr>
<td>Identification</td>
<td>1-10</td>
</tr>
<tr>
<td>Listing</td>
<td>5-88, 5-90</td>
</tr>
<tr>
<td>Selection</td>
<td>5-56, 5-78</td>
</tr>
<tr>
<td>JOVIAL keywords</td>
<td>A-1, A-2</td>
</tr>
<tr>
<td>Library</td>
<td></td>
</tr>
<tr>
<td>Commands</td>
<td>4-4, 4-5, 4-6</td>
</tr>
<tr>
<td>Definition</td>
<td>1-4</td>
</tr>
<tr>
<td>LIBNEW</td>
<td>1-4, 2-2, 4-4, 5-5, 5-45</td>
</tr>
<tr>
<td>LIBOLD</td>
<td>1-4, 2-3, 2-9, 2-10, 2-19, 4-4, 5-82</td>
</tr>
<tr>
<td>Macro commands</td>
<td>4-2, 4-3, 5-43, 5-59, 5-98, 5-109</td>
</tr>
<tr>
<td>Merging libraries</td>
<td>5-79</td>
</tr>
<tr>
<td>Module</td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td>1-10, 2-2</td>
</tr>
<tr>
<td>Example</td>
<td>1-7, 1-8</td>
</tr>
<tr>
<td>Listing</td>
<td>5-92</td>
</tr>
<tr>
<td>Selection</td>
<td>5-81</td>
</tr>
<tr>
<td>OFFTRACE directive</td>
<td>1-12</td>
</tr>
</tbody>
</table>
**INDEX (Cont.)**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>1-3</td>
</tr>
<tr>
<td>PROBD</td>
<td>2-15</td>
</tr>
<tr>
<td>Probes</td>
<td>2-7</td>
</tr>
<tr>
<td>PROBI</td>
<td>2-13, 2-15, 5-75</td>
</tr>
<tr>
<td>Automatic insertion</td>
<td>2-15</td>
</tr>
<tr>
<td>Example</td>
<td>5-75, 5-76, 5-101, 5-102</td>
</tr>
<tr>
<td>Program flow</td>
<td>2-4</td>
</tr>
<tr>
<td>Picture</td>
<td>5-26</td>
</tr>
<tr>
<td>Program documentation</td>
<td></td>
</tr>
<tr>
<td>DOCUMENT</td>
<td>5-60</td>
</tr>
<tr>
<td><strong>Punch</strong></td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>1-4, 2-13, 2-16</td>
</tr>
<tr>
<td>Javastext</td>
<td>5-103, 5-104</td>
</tr>
<tr>
<td>Module</td>
<td>5-106</td>
</tr>
<tr>
<td>Reaching set</td>
<td>5-29</td>
</tr>
<tr>
<td>Retesting guidance</td>
<td>2-10</td>
</tr>
<tr>
<td>Source text processing</td>
<td>2-2</td>
</tr>
<tr>
<td>Statement type abbreviations</td>
<td>C-1</td>
</tr>
<tr>
<td><strong>START-TERM</strong></td>
<td></td>
</tr>
<tr>
<td>Selection command</td>
<td>4-8</td>
</tr>
<tr>
<td>Breakdown into modules</td>
<td>2-2</td>
</tr>
<tr>
<td>Text identification</td>
<td>1-10</td>
</tr>
<tr>
<td><strong>STRUCTURAL</strong></td>
<td></td>
</tr>
<tr>
<td>Constraints</td>
<td>3-4</td>
</tr>
<tr>
<td>Function</td>
<td>1-2, 2-1</td>
</tr>
<tr>
<td>Option command</td>
<td>4-10, 4-11</td>
</tr>
<tr>
<td>Processing</td>
<td>2-3</td>
</tr>
<tr>
<td>Symbol table</td>
<td>5-40, 5-97</td>
</tr>
<tr>
<td>Test assistance</td>
<td>2-9</td>
</tr>
<tr>
<td>Test case</td>
<td></td>
</tr>
<tr>
<td>Initiation</td>
<td>2-15</td>
</tr>
<tr>
<td>Termination</td>
<td>2-15</td>
</tr>
</tbody>
</table>

1-4
INDEX (Cont.)

Test description
  PROBD  2-15

Test execution
  Compilation  1-2
  Input/Output  2-13, 2-16
  Loading  2-18
  Processing  2-18
  Purpose  2-11

Time (execution)  5-22

TRACE directive  1-12, 1-13

Tracing
  DD-paths  5-12
  Modules  5-17

Tree  5-57

Universal
  Commands  1-5
  Constraints  3-2

Wrapup  4-16, 5-62
REFERENCES


**BASE UNITS:**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Unit</th>
<th>SI Symbol</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>metre</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>mass</td>
<td>kilogram</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>second</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>electric current</td>
<td>ampere</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>thermodynamic tempera</td>
<td>kelvin</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>amount of substance</td>
<td>mole</td>
<td>mol</td>
<td></td>
</tr>
<tr>
<td>luminous intensity</td>
<td>candela</td>
<td>cd</td>
<td></td>
</tr>
</tbody>
</table>

**SUPPLEMENTARY UNITS:**

| Plane angle           | radian       | rad       |         |
| Solid angle           | steradian    | sr        |         |

**DERIVED UNITS:**

| Acceleration          | metre per second squared | ms         | time   |
| Activity of a radioac| disintegration per second| s         |        |
| Angular acceleration  | radian per second squared| rad s      | angle  |
| Angular velocity      | radian per second      | rad s      |        |
| Area                  | square metre          | m          |        |
| Density               | kilogram per cubic metre | kg·m      |        |
| Electric capacitance  | farad                | F          |        |
| Electrical conductance| siemens              | S          |        |
| Electric field strength| volt per metre     | V          |        |
| Electric inductance   | volt                  | V          |        |
| Electric potential difference| ohm            | V          |        |
| Electric resistance   | volt                  | V          |        |
| Electromotive force   | joule                 | J          |        |
| Energy                | joule per kelvin     | J          |        |
| Entropy               | joule per kelvin     | J          |        |
| Force                 | newton               | N          |        |
| Frequency             | hertz                | Hz         |        |
| Illuminance           | lumen                | lm         |        |
| Luminance             | candela per square metre | cd·m      |        |
| Luminous flux         | lumen                | lm         |        |
| Magnetic field strength| ampere per metre     | T          |        |
| Magnetic flux         | weber                | W          |        |
| Magnetic flux density | ampere               | A          |        |
| Magnetomotive force   | watt                 | W          |        |
| Power                 | joule                | J          |        |
| Pressure              | pascal               | Pa         |        |
| Quantity of electricity| coulomb             | C          |        |
| Quantity of heat      | joule                | J          |        |
| Radiant intensity     | watt per sterdian    | J          |        |
| Specific heat         | joule per kilogram-kelvin | W·m·K  |        |
| Stress                | pascal               | Pa         |        |
| Thermal conductivity  | watt per metre-kelvin| W·m·K      |        |
| Velocity              | metre per second     | m·s        |        |
| Viscosity, dynamic    | square metre per second | m·s·s     |        |
| Viscosity, kinematic  | volt                 | V          |        |
| Voltage               | cubic metre          | J          |        |
| Volume                | reciprocal metre     | J          |        |
| Wavenumber            | joule                | J          |        |

**SI PREFIXES:**

<table>
<thead>
<tr>
<th>Multiplication Factors</th>
<th>Prefix</th>
<th>SI Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 000 000 000 000</td>
<td>Tero</td>
<td>T</td>
</tr>
<tr>
<td>1 000 000 000</td>
<td>Giga</td>
<td>G</td>
</tr>
<tr>
<td>1 000 000</td>
<td>Mega</td>
<td>M</td>
</tr>
<tr>
<td>100</td>
<td>Kilo</td>
<td>k</td>
</tr>
<tr>
<td>10</td>
<td>Hecto</td>
<td>h</td>
</tr>
<tr>
<td>1</td>
<td>Deka</td>
<td>d</td>
</tr>
<tr>
<td>0.1</td>
<td>Centi</td>
<td>c</td>
</tr>
<tr>
<td>0.01</td>
<td>Milli</td>
<td>m</td>
</tr>
<tr>
<td>0.001</td>
<td>Micro</td>
<td>µ</td>
</tr>
<tr>
<td>0.000 001</td>
<td>Nano</td>
<td>n</td>
</tr>
<tr>
<td>0.000 000 01</td>
<td>Pico</td>
<td>p</td>
</tr>
<tr>
<td>0.000 000 001</td>
<td>Femto</td>
<td>f</td>
</tr>
</tbody>
</table>

* To be avoided where possible
MISSION

of

Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.