PDL/81™

Design Language Reference Guide

(Version 2.0)

Caine, Farber & Gordon, Inc. **Warren Point International Ltd.**

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure of the software described herein is governed by the terms of a license agreement or, in the absence of an agreement, is subject to restrictions stated in subparagraph (c)(1) of the Commercial Computer Software – Restricted Rights clause at FAR 52.227-19 or subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, as applicable. [Caine, Farber & Gordon, Inc.; 1010 East Union St.; Pasadena, CA 91106]

Comments or questions relating to this manual or to the subject software are welcomed and should be addressed to:

In North America:

Caine, Farber & Gordon, Inc.
1010 East Union Street
Pasadena, CA 91106
USA

Tel: (800) 424-3070 or
(818) 449-3070

Fax: (818) 440-1742

In the Rest of the World:

Warren Point International Ltd.
Babbage Road
Stevenage, Herts SG1 2EQ
England

Tel: 0438 316311

Fax: 0227 86521

Form Number: 9102-2

1 August 1988
20 February 1989
1 December 1991

Contents

Appendices

1. Introduction

PDL/81 is a software tool intended as an aid to designing and documenting a program or system of programs. The tool consists of a processor and a style library which is used to tailor the processor to the particular requirements of the document being produced. As distributed, the style library includes definitions for such document styles as:

- program designs;
- manuals and reports;
- memoranda; and
- business letters.

This manual describes the particular data base components which relate to formatting program designs. Other manuals (see Section 1.3) describe the other data base components and the methods for modifying the data base.

The original version of PDL, known as PDL/74, was first developed in 1973. It was intended exclusively for processing program design documents and displaying these documents in a predetermined style. Over the years since the first release, the large PDL user community has provided numerous suggestions for changes and improvements. Most of these suggestions came from the desire to improve the text handling capabilities of PDL/74 and the desire to have significantly more control over the detailed format of the resulting document. PDL/81 addresses these desires directly while still presenting an interface to the designer which is easy to use.

1.1 Features and Capabilities of PDL/81

PDL/81 is a tool which integrates the capabilities commonly associated with a program design language processor and those of a text processing system.

This integration is accomplished by providing an extensive set of primitive formatting operations and a definitional language which allows a format designer to compose abstract constructs from these primitive operations. As an example, a document style for program designs might contain such concepts as “data segment” and “flow segment” while a style for manuals might contain such concepts as “chapter”, “enumerated list”, and “paragraph”.

The end user of PDL/81 uses these abstract concepts without any need to understand the underlying implementation or format design methods. Thus, writing and processing program designs is as simple with PDL/81 as with PDL/74, but the local project manager has significantly more control over the layout and appearance of the resulting design document.

The primitive operations of the Format Design Language allow the format designer a very high degree of flexibility in creating document styles. Among the available capabilities are:

- Complete control over page layout including sheet dimensions and top, bottom, left, and right margins;
- Simple measurements of the cyclomatic complexity of a design;
- Tracking of requirements sections throughout a design;
- Checking that procedure definitions and invocations are consistent;
- Arbitrary running text at top and bottom of each page including security banners with document classification and sheet count;
- Definition of primary and secondary keywords for use in program designs;
- Definition of layout and characteristics of all program design segment types and the ability to create new types of segments;
- Ability to include input from alternate files;
- Automatic generation of table of contents and other such tables (e.g., table of figures, table of tables);
- Automatic generation of document indexes in various forms.

1.2 Document Styles and the PDL/81 Style Library

The document styles which are available at an installation reside in the PDL/81 '*style library*. The form of the library depends on the particular host operating system. The particular style to be used in a PDL/81 run is specified as an option when PDL/81 is invoked.

1.3 Related Publications

Other publications relating to the use of PDL/81 are:

- *PDL/81 Introduction and Invocation Guide* – a guide to invoking PDL/81 under various operating environments
- *PDL/81 Ada Design Language Reference Guide* – a guide to using PDL/81 for Ada program design
- *PDL/81 Document Language Reference Guide* – a guide to using PDL/81 for producing various documents such as manuals and reports
- *PDL/81 Format Designers Guide* – A guide to developing new types of PDL/81 design and document styles
- *PDL/81 Installation Guide* – a guide to installing PDL/81 under the various supported operating systems.

1.4 A Note to the Reader

This manual describes the distributed document style “design” which is intended to be the standard style for formatting program designs. A sample program design is presented in Section D.1.

If you don't like the results of this style, you may desire to modify the data base. Simple modifications can generally be accomplished after an examination of various data base entries. Extensive modifications, and the development of entirely new design styles, will require reference to the *PDL/81 Format Designers Guide*.

The “design” style should be considered as an example of the kind of design tool which may be defined with PDL/81. For any particular project, it may be desirable to tailor a specific definitions file by removing many of the options which are described here.

2. General Information

This chapter discusses various aspects of the PDL/81 design style which are of general interest. It includes information on the form of a design, overall operation of the processor, and the syntax of PDL/81 commands.

This chapter does not discuss how to invoke the PDL/81 processor under the various supported operating systems. Invocation is discussed in the *PDL/81 Introduction and Invocation Guide*.

2.1 Format of a Design

The PDL/81 design style accepts as input a series of source lines and produces a design document. The document can be formatted for printing on different types of output devices and different paper sizes.

A sample design is showen in Section D.1. The design document is composed of several major sections:

1. Front matter
2. Design body
3. Reports
4. Final page

which are now briefly described.

2.1.1 Front Matter

This is the first part of the design document. It begins with a *title page* which identifies the particular design. Primary information for this page comes from the *title* command (see Section 10.1) and the *date* command (see Section 10.2).

The title page is followed by the *table of contents* for the design. The table of contents lists all of the sections and subsections which make up the design along with their corresponding page numbers.

2.1.2 Design Body

The *design body* presents the actual data definitions, procedure definitions, and textual information of the design. This section is composed of various kinds of *segments* which may be structured into *groups* (see Chapter 3). The segment types are:

- *Text Segments*: which represent arbitrary commentary (see Chapter 4).
- *Data Segments*: which allow explicit definition of data items (see Section 6.3).
- *Flow Segments*: which represent the procedural flow of the design (see Chapter 7).
- *External Segments*: which allow declaration of procedures which are assumed to be defined somewhere outside of this design document (see Chapter 8).

2.1.3 Reports

The processor can be instructed to produce several reports (see Chapter 12) which provide information about the content and internal structure of the design. These reports are particularly useful in understanding a design. The possible reports are:

- *Reference Trees*: which shows all of the flow segments arranged in the form of a calling tree. There will be several trees if there are several flow roots in the design. Recursive use of flow segments will be indicated. Reference trees are further described in Section 12.1.
- *Data Index*: which lists each data item in alphabetic order and shows the points in the design where each is referenced. The data index is further described in Section 12.2.
- *Flow Segment Index*: which lists each flow segment in alphabetic order and shows the points in the design where each is referenced. The flow segment index is further described in Section 12.3.
- *Overly Complex Segment Index*: which lists each segment which has a cyclomatic complexity value greater than the selected maximum (see Section 11.1 for a discussion of complexity measurement and Section 12.4 for a discussion of the report).
- *Requirements Index*: which lists each declared requirement number and the segments that address that requirement (see Section 11.2 for a discussion of requirements tracking and Section 12.5 for a discussion of the report).
- *Calls-in-Context Index*: which shows each procedure, function, or entry pint definition along with each line that calls it (see Section 11.3 for a discussion of calls-in-context and Section 12.6 for a discussion of the report).

2.1.4 Final Page

This is the last page of the design document. Besides confirming that the design was completely processed, this page displays a number of statistics about the processing.

2.2 Invocation of PDL/81

The manner of invoking PDL/81 depends on the particular operating system being used. Refer to the *PDL/81 Introduction and Invocation Guide* for specific information.

2.3 Overall Operation

PDL/81 processes a design in two passes. During the first pass, the source is read, page breaks are determined, and a dictionary of data item and segment names is constructed. During the second pass, the source is reread, references to data items and segments are detected, and the design document is formatted. During both passes, progress is noted by displaying the current page number and processing phase on a file (which will usually be the controlling terminal).

2.4 Input Format

Input to PDL/81 consists of a sequence of source lines. Each line is terminated by a newline character. This section describes the interpretation of various special characters and character sequences within source lines. The only ASCII control codes allowed on an input line are “tab” and “newline”.

2.4.1 Tab Expansion on Input

ASCII tab characters are allowed on input lines. Each tab will be replaced by enough blanks to position the immediately following character to the next input tab stop. Input tab stops are set at columns 1, 9, 17, ...

2.4.2 Continuation of Input Lines

Any input line may be continued in one of two ways:

- The sequence “\<newline>” results in deletion of both characters, thus causing the following line to be considered part of the current line. The character “\” is known as the *escape character* and has additional uses as described in Section 2.4.3.
- The sequence “/<newline>” will be replaced by a single blank, thus causing the current and following lines to be treated as a single line with their contents separated by a blank. The character “/” is known as the *continue character*. It has special significance only when it immediately precedes a newline character – in any other context, it is just another character.

2.4.3 Special Characters

The character sequence “#{” is used to introduce a text function as described in Chapter 9. The sequence should not appear in any other context, as the results will be unexpected. If it is necessary to use the sequence for some other purpose, the “#” should be protected by an escape character as in “\#{”.

The special sequence “*” will be replaced by the so-called *bullet* character (bullet) in the printed output. The form of this special character depends greatly on the output device being used.

The escape character followed by a space is known as the *unpaddable space*. It will be replaced by a single space in the printed output, but will not be considered to mark a word break during processing.

2.5 Command Lines

If the first character of a line is a "%", the line is known as a *command line*. Command lines contain *commands* which direct various types of processing or provide various information to PDL/81.

When a command line is encountered, white space (blanks and tabs) following the "%" is skipped. If a newline is encountered, the command line is ignored. If an asterisk ("*") is encountered, the line is considered to be a *comment command*, the rest of the line is skipped, and the whole line is ignored.

If anything else is encountered, it is assumed to start a *command name* which extends to the first blank, tab, or newline. After skipping any white space, the remainder, if any, of the line is considered to be the *command argument*. Thus, for example.

```
%Title    This is a Sample
```

is a command line with a command name of "Title" and a command argument of "This is a Sample".

Commands may have multiple arguments which are separated from each other by semicolons (";"). Thus, the general form of a command line is

```
%name [argument[;argument]...]
```

where the brackets indicate optional material.

The case (upper, lower, mixed) of a command name is immaterial. Thus, for example, "Title", "TITLE", "title", or even "tiTlE" all represent the same command name.

2.6 Including Alternate Source Files

At any point in the design source, input may be switched to another source file by the command

```
%Include  file
```

where *file* is the name of the file to be included. Files included with an %Include command may contain %Include commands.

2.7 Design Body Conventions

As outlined in Section 2.1.2, the design body is composed of a number of segments. There are no restrictions on the ordering of segments. The only restriction on the number of segments is that imposed by the amount of memory available to PDL/81 while processing a design.

2.7.1 Segment Delimiting

A segment is introduced by one of the segment commands described elsewhere in this manual. These commands are:

%Text or %T	start a text segment (Chapter 4)
%TextF or %TF	start a formatted text segment (Chapter 4)
%Data or %D	start a data segment (Section 6.3)
%Segment or %S	start a flow segment (Chapter 7)
%External or %E	start an external segment (Chapter 8)

A segment is terminated by the next occurrence of a segment command, a %Group command (see Chapter 3), or the end of the design source.

2.7.2 Display of Segments

Each segment can occupy one or more pages. However, experience has shown that designs are generally much more readable and understandable if each segment is limited to a single page.

Each segment will be enclosed in a box composed of characters specific to the type of segment. The various characters are:

#	text segment
D	data segment
*	flow segment
X	external segment

If the body of a segment is empty, the box will contain a generated notice that the segment was intentionally left blank.

2.7.3 Comment Strings

The description of data segments (Section 6.3) and of flow segments (Chapter 7) will refer to syntactic constructs known as *comment strings* which are used as delimiters in certain contexts (e.g., to separate a procedure name from its "arguments").

Initially, there are two comment strings defined – the dot (".") and the left parenthesis ("("). Replacement or additional comment strings may be defined by the command

```
%CString   [string]
```

where *string* is one or two non-blank printing characters other than letters or digits. No two comment strings may begin with the same first character. If *string* is absent, all comment strings will be deleted.

For example, to add the Ada comment convention of "--" to the list of comment strings, the command

```
%CString    --
```

may be used. The sequence of commands

```
%Cstring
%CString    (
%CString    --
```

will establish just the left parenthesis and the Ada comment convention as the only comment strings.

3. Groups

A design may be broken into various sections by use of commands of the form

```
%Group   text
```

or

```
%G   text
```

where *text* is any sequence of characters to be used as the title for the group. In the design document, each group will be prefaced with a page containing the title of the group centered and boxed. The title will also appear as a subtitle on each design page within the group and will be placed in the table of contents for the design.

Examples of group declarations are

```
%Group  Pass One Processing
%G      Input Editing Phase
```

A group is terminated by the next “Group” or “G” command or by the end of the design.

4. Text Segments

Text segments are used to place blocks of commentary into a design. They are frequently used to supply such material as introductory information, table layouts, and record layouts. There are two types of text segments – *unformatted* and *formatted*. Only commands specific to text segments are described in this chapter. See Chapter 5 and Chapter 9 for a discussion of other functions and commands which are useful in text segments.

4.1 Unformatted Text Segments

An unformatted text segment is introduced by the command

```
%Text   text
```

or

```
%T   text
```

where *text* is any sequence of characters to be used as the title of the segment. The title will be displayed at the top of the segment page and will be entered in the table of contents.

The lines comprising the body of an unformatted text segment are simply input and printed as is. No automatic formatting will be performed except that lines which are too long to fit into the segment box will be split at word boundaries and printed on two or more lines. White space on input lines is kept and blank input lines will result in blank output lines.

Examples of commands to introduce an unformatted text segment are:

```
%Text   Introduction to Position Monitoring Module
%T      Other Documents Relating to this Subsystem
```

4.2 Formatted Text Segments

A formatted text segment is introduced by the command

```
%TextF   text
```

or

```
%TF   text
```

where *text* is any sequence of characters to be used as the title of the segment. The title will be displayed at the top of the segment page and will be entered into the table of contents.

The lines which comprise the segment are considered to be running text. Words are collected, regardless of the input line boundaries, and are put into the output line until a word does not fit. The output line is then printed and a new output line is started. This action is known as “breaking” the line. A break is forced by a blank input line and by the commands described in Section 4.2.1 and Chapter 5. White space on input lines is kept and blank input lines result in blank output lines.

Examples of commands introducing formatted text segments are:

```
%TextF   Standards Used in this Design
%TF      Outline of Link-Level Protocols
```

4.2.1 Lists

Within formatted text segments, three types of lists may be automatically formatted:

bullet	each list entry is prefixed by the bullet (bullet) character.
numbered	each list entry is prefixed by an automatically generated number.
verb	each list entry is prefixed by an arbitrary word or phrase. (This list is an example of a verb list.)

The same general structure is used for generating each kind of list. This structure, presented in the form of a numbered list, is:

1. a list start command specifying the type of the list
2. one or more list entries
3. a %LE (list end) command to mark the end of the list

Lists may be nested.

Each list will be automatically preceded and followed by a blank line.

4.2.1.1 Bullet Lists

A bullet list is introduced by the command

```
%BL
```

The text for the list entries should follow, separated from each other by a single blank line. The list is closed by the command

```
%LE
```

which should not be preceded by a blank line.

4.2.1.2 Numbered Lists

A numbered list is introduced by the command

```
%NL
```

The text for the list entries should follow, separated from each other by a single blank line. The list is closed by the command

```
%LE
```

which should not be preceded by a blank line.

4.2.1.3 Verb Lists

A verb list is introduced by the command

```
%VL [indent]
```

where *indent* is a decimal integer which specifies the number of characters to indent the text of the list items. In the absence of *indent*, a value of 16 will be used.

Each item in a verb list is introduced by the command

```
%Verb text
```

where *text* is the word or phrase to be displayed at the left margin. The text of the list entry follows on succeeding lines.

The last entry in a verb list should be followed by the command

```
%LE
```

to close the list.

4.3 Switching Between Formatted and Unformatted Modes

The initial formatting mode (formatted or unformatted) for a text segment is determined by the command which introduces that text segment. Within a text segment, either formatting mode may be established at any point by the commands

```
%Fill
```

which establishes formatted mode and

```
%NoFill
```

which establishes unformatted mode.

5. General Formatting Commands

This chapter describes commands which relate to the general control of formatting within a segment. These commands are most often used in text segments (see Chapter 4) but may be used in any kind of segment.

5.1 Vertical Spacing Commands

Blank lines may be inserted into a segment by the command

```
%Space   number
```

where *number* is a decimal integer giving the number of blank lines to insert. If the given number of blank lines exceeds the number of available lines remaining on the page, a new page is started instead. For example,

```
%Space   3
```

would cause three blank lines to be inserted or would cause a page eject if there were not at least three lines remaining on the page.

The command

```
%Need   number
```

where *number* is a decimal integer, will cause a page eject if at least *number* lines do not remain on the current page. If at least that many lines remain, the command has no effect. Thus,

```
%Need  5
```

will cause a new page to be started if fewer than five lines remain on the current page.

The command

```
%Eject
```

will cause a new page to be started.

5.2 Heading Commands

The heading commands allow descriptive headings to be placed within a segment. When mentioned below, the terms “centered” and “flush left” are to be taken as being relative to the textual display area within the segment box.

The command

```
%MajorHeading   text
```

will skip two lines; print *text* centered, underscored, and capitalized; and skip two lines.

The command

```
%Heading   text
```

will skip two lines; print *text* flush left, underscored, and capitalized; and skip one line.

The command

```
%SubHeading   text
```

will skip one line; print *text* flush left and underscored; and skip one line.

6. Data Item Declaration

PDL/81 allows the designer to declare certain items known as *data items*. References to these items within flow segments will be collected and may be displayed in the data item index (see Section 12.2).

6.1 Data Items

Within data segments (see Section 6.3) and flow segments (see Chapter 7), tokens consisting of letters, digits, and certain special characters are considered to be *potential data items*. A potential data item will be considered to be an actual data item if it is defined as such in an implicit (see Section 6.2) or explicit (see Section 6.3) data declaration.

Lines which begin (possibly after some leading white space) with a comment string (see Section 2.7.3) will not be examined for potential data items.

The special characters which may be part of a potential data item are initially "$", "#", "@", and "_". Thus, in the line

```
x = a$1+bb*cc
```

the potential data items are

```
x   a$1   bb   cc
```

The case (upper, lower, mixed) of letters in names of potential data items is immaterial. Thus, for example, "test", "Test", "TEST", and even "tEst" all represent the same item.

The set of these special characters may be modified by the command

```
%DSChar   char
```

where *char* is a non-blank, non-alphanumeric character to be added to the set of data item special characters. If *char* is absent, the set is made empty. All uses of the "DSChar" command should precede the first segment.

As an example, the special characters “%” and “!” can be added to the set by the commands

```
%DSChar    %
%DSChar    !
```

and the set can be defined to contain only the character “$” by

```
%DSChar
%DSChar    $
```

6.2 Implicit Data Item Declaration

When a potential data item is encountered in a flow segment, it will be declared as an implicit data item if

1. it contains a *data character*;
2. it is longer than one character; and
3. it is not declared elsewhere in the design as an explicit data item.

Initially, the underscore (“_”) is the only data character. New data characters may be added to the set of data characters by the command

```
%DChar    char
```

where *char* is a non-blank, non-alphanumeric character to be added. If *char* is absent, the set is made empty. All uses of the “DChar” command should precede the first segment.

For example, the characters “-” and “$” can be added to the set of data characters by the commands

```
%DChar    -
%DChar    $
```

and the character “!” can be defined as the only data character by

```
%DChar
%DChar    !
```

For compatibility with older versions, the command

```
%DataChar  char
```

establishes the single character *char* as the only data character. The preferred method of changing data characters is to use the “DChar” described above.

6.3 Explicit Data Item Declaration (Data Segments)

Data items are explicitly defined in *data segments*. A data segment is introduced by the command

```
%Data   text
```

or

```
%D   text
```

where *text* is a sequence of characters to be used as the title of the data segment. The title will be displayed at the top of the segment page and will be entered in the table of contents. Note that the "Data" or "D" commands do *not*, themselves, declare data items – they *introduce* segments *in which* data items are declared. Examples of these commands are:

```
%Data   Formats for Master File Records
%D      Miscellaneous Data Definitions
```

The actual data definitions occur in the *body* of a data segment. Lines beginning with a comment string (see Section 2.7.3) are considered to be comments and are not scanned for declarations. White space on source lines is kept and blank input lines will result in blank output lines.

6.3.1 Normal Declaration Mode

In the normal mode of data item declaration, the first potential data item in each line of the body is declared to be an actual data item. Anything following the data item on the line is taken as commentary. Thus, in the line

```
CType is the type of the command
```

"CType" will be declared to be a data item.

6.3.2 Special Declaration Mode

In the special declaration mode, a potential data item is declared as an actual data item *only* if it contains a data character. If the data character is the *first* character of the potential data item, it is not included as part of the name of the actual data item.

At the start of each data segment, the normal declaration mode is in effect. The special declaration mode can be established for that segment by the command

```
%SDMode
```

and the normal declaration mode can be re-established by the command

```
%NoSDMode
```

As an example, consider the line

```
Items _c1, file_count, and _open_count are counters
```

In the special declaration mode, the actual data items will be

```
c1      file_count      open_count
```

As another example, the lines

```
*********************************//*******************
*       *         *                                   *
* _code * _count *             record_text            *
*       *         *                                   *
*********************************//*******************
```

will declare “code”, “count”, and “record_text” to be actual data items.

7. Flow Segments

A *flow segment* presents, in a program-like form, the procedural flow of a portion of a design. Generally, each flow segment represents a *procedure* in the program. A flow segment is introduced by the command

```
%Segment   text
```

or

```
%S   text
```

where *text* is a sequence of characters which represents the name of the segment. The name will appear at the top of the segment page. That portion of the name up to the first comment string (see Section 2.7.3) will be placed in the table of contents and will be saved in a dictionary for indexing purposes. In saving the name in the dictionary, leading and trailing blanks are removed and each sequence of imbedded blanks is collapsed into a single blank. Some examples are:

Command	Saved Name
`%Segment System Start`	System Start
`%S Install in Data Base (Name, Type)`	Install in Data Base
`%Segment Search Dictionary (Name)`	Search Dictionary

7.1 Flow Segment Body

The body of a flow segment is composed of one or more lines. These lines are known as *statements* to emphasize the relation between a flow segment and a procedure in a programming language.

A statement may start anywhere on a line. PDL/81 will correctly format and indent each statement on output and may supply various forms of visual enhancement to the printed line. Leading blanks will be removed and each sequence of imbedded blanks will be replaced by a single blank. Blank input lines will be ig-

nored. Statements which are too wide to fit in the segment box will be automatically continued when printed.

This automatic formatting means that there is no need for the designer to do any special formatting of the flow segment input lines. In fact, each statement is normally just typed flush left on the input line and layout is left to PDL/81.

With the exception of the *special statements* discussed in Section 7.4, the contents of a statement may be anything desired. Some examples are:

```
Count = Count + 1
Increment Count
Bump Count to reflect/
the record just processed
```

Note that, since "/" is the *continue character* (see Section 2.4.2), the last two lines of this example are equivalent to:

```
Bump Count to reflect the record just processed
```

7.2 Reference Recognition

Each statement in a flow segment, except for a statement which begins with a comment string (see Section 2.7.3), will be scanned to see if it is the name of a flow segment. If a statement begins with a *keyword* (see Section 7.4), the scan begins following the keyword and any subsequent *secondary keywords*. The scanning stops at the first comment string.

In any match, leading and trailing blanks are removed, each sequence of imbedded blanks is replaced by a single blank, and the case (upper, lower, mixed) of all letters is ignored.

Lines beginning with comment strings in data segments and flow segment are normally not scanned for data item definitions or references or for flow segment references. Scanning can be specified in this case by the command

```
%CData
```

and may be inhibited by the command

```
%NoCData
```

If used, these commands should appear before the first segment.

7.3 Labels

It is occasionally desirable to place *labels* in a design. They are convenient for denoting statement sequences in DO CASE constructs (see Section 7.4.3.7) and in supplying names for DO constructs for use with UNDO and CYCLE statements (see Section 7.4.3.3 and Section 7.4.3.4).

If a colon (":") is encountered before the first blank in a statement, that statement is considered to be a *label*. The statement will be printed one indentation level to the left of the current indentation level. Anything following the colon on the same line will be treated as commentary. Some examples of labels are:

```
MainSearchLoop:
END_OF_FILE:
+,-,*:
+:
"other":
```

7.4 Special Statements

The so-called *special statements* comprise the flow-of-control statements in the PDL/81 procedural language. This section describes each of the special statements.

7.4.1 Keywords and Secondary Keywords

Each special statement begins with a *keyword* followed by a blank or the end of the input line. The keywords, grouped generally as used, are

```
IF        ELSEIF        ELSE        ENDIF
DO        UNDO          CYCLE       ENDDO      ENDO
RETURN
```

The particular keyword which starts a statement determines the indentation level for that and subsequent statements. A word is considered to be a keyword only when it is the first word of a statement.

There is also a set of so-called *secondary keywords* which are recognized as such only when immediately following a keyword or a secondary keyword. The secondary keywords are

```
IF      WHILE      FOREVER      CASE      UNTIL      FOR
NOT
```

The case (upper, lower, mixed) of keywords and secondary keywords is ignored.

The keywords and secondary keywords discussed above are those defined in the *design* style as distributed. New project-wide keywords and secondary keywords may be added by modifying the style file. Additions can also be made for a single design as described in Appendix C.

NOTE

Keywords in PDL/81 are used *only* to change the indentation level. The processor in *no way* checks for correct use. However, error messages will appear in the design document if a keyword attempts to create a negative indentation level or if a flow segment ends with a positive indentation level. These messages may be surpressed when the PDL/81 processor is installed.!

7.4.1.1 Keyword Enhancement

The form in which a keyword or secondary keyword is printed depends on the use of various commands described in this section. These commands, if used, should appear before the first segment. Initially, keywords and secondary keywords are printed in *upper* case regardless of the case in which they are entered.

The command

```
%LCase
```

causes keywords and secondary keywords to be printed in lower case regardless of the case in which they are entered.

The command

```
%SCase
```

causes keywords and secondary keywords to be printed in the same case in which they were entered.

For compatibility with older versions, the command

```
%NoLCase
```

also specifies that keywords are to be printed in the same case in which they are entered. The preferred method of accomplishing this is to use the “SCase” command described above.

The command

```
%UCase
```

causes keywords and secondary keywords to be printed in upper case regardless of the case in which they were entered. This is the default setting.

The command

```
%UScore
```

causes each keyword and secondary keyword to be underscored when printed.

The command

```
%NoUScore
```

specifies that each keyword and secondary keyword is not to be underscored when printed. This is the default setting.

Flexibility in font selection is provided by the command

```
%KWFont   n
```

where *n* is a font number:

0 base font (no special treatment except for possible conversion to upper case or lower case under control of the %UCASE or %LCASE commands).

1 underscored (same effect as obtained by the %USCORE command).

2 bold face (only if supported by your installation on the selected device).

7.4.2 The IF Construct

The IF construct consists of the keywords IF, ELSEIF, ELSE, and ENDIF. In its simplest form, it can be written as:

```
IF condition
    sequence
ENDIF
```

which implies that the statements comprising “sequence” are only to be executed if “condition” is true.

The basic form can be expanded by adding an alternate as in:

```
IF condition
    sequence-1
ELSE
    sequence-2
ENDIF
```

which implies that “sequence-1” is to be executed if “condition” is true and that “sequence-2” is to be executed if “condition” is false.

Multiple IF constructs can be nested as in:

```
IF condition-1
    sequence-1
ELSE
    IF condition-2
        sequence-2
    ELSE
        sequence-3
    ENDIF
ENDIF
```

Since nested IF constructs are quite common, an alternate form can be used as in:

```
IF condition-1
    sequence-1
ELSEIF condition-2
    sequence-2
ELSE
    sequence-3
ENDIF
```

Thus, the general form of the IF construct is

1. an IF
2. zero or more ELSEIF's
3. zero or one ELSE
4. an ENDIF

7.4.3 The DO Construct

The DO construct consists of the keywords DO, ENDDO, CYCLE, and UNDO. The word "ENDO" is considered an alternate spelling of "ENDDO". The DO construct is used to produce flow figures of the general form:

```
DO iteration or selection criteria
    statement
    statement
    ....
ENDDO
```

The iteration or selection criteria may be arbitrarily chosen. The remainder of this section presents examples of the more commonly used criteria.

7.4.3.1 The DO WHILE Construct

This form of the DO construct implies iteration as long as some given condition remains true. No iteration at all would be performed if the condition is initially false. It can be written as:

```
DO WHILE
    statement
    statement
    ....
ENDDO
```

Some examples of possible conditions are:

```
DO WHILE there is source input remaining
DO WHILE the current character is a space
DO WHILE there is room in the table
```

7.4.3.2 The DO UNTIL Construct

This construct implies iteration until some condition becomes true and, further, implies that at least one iteration will always be performed. It can be written as:

```
DO UNTIL condition
    statement
    statement
    ....
ENDDO
```

Some examples are:

```
DO UNTIL table is full
DO UNTIL last record is read
DO UNTIL source is depleted
```

7.4.3.3 The UNDO Statement

The UNDO statement is used to indicate that control should pass to the statement immediately following the ENDDO of the current DO construct, thus causing premature exit from the loop. It might be used in the following context:

```
DO WHILE source input remains
    process next source line
    IF dynamic memory is full
        UNDO
    ENDIF
ENDDO
```

An alternate form of the UNDO statement is:

```
UNDO IF condition
```

which can make the design more concise as in:

```
DO WHILE source input remains
    process next source line
    UNDO IF dynamic memory is exhausted
ENDDO
```

When DO constructs are nested, it may sometimes be necessary to indicate a premature exit from an outer loop. This is most easily shown by labelling the outer DO and writing

```
UNDO label
```

7.4.3.4 The CYCLE Statement

The CYCLE statement indicates premature transfer of control to the loop test or selection point – that is, to a point just before the ENDDO statement which terminates the loop. The most commonly used forms of the statement are:

```
CYCLE
CYCLE IF condition
CYCLE label
```

7.4.3.5 The DO FOREVER Construct

The simplest form of loop is written as:

```
DO FOREVER
    statement
    statement
    ....
ENDDO
```

which implies continuous repetition until something (either in the loop or an outside event) causes an exit. Thus the body of the loop should usually contain an UNDO or a RETURN statement.

7.4.3.6 The DO FOR Construct

This construct is used for selecting items from some list or sequence. Its general form is:

```
DO FOR selector
    statement
    statement
    ....
ENDDO
```

The actual selector can be chosen to be as meaningful as possible to the designer and reader. Examples are:

```
DO FOR each table entry
DO FOR each element in the Positions array
DO FOR all nodes in the tree
DO FOR all "interesting" entries in the dictionary
```

7.4.3.7 The DO CASE Construct

The DO CASE construct is used to select one of a group of actions according to a given selection criterion. The meaning of the construct is clearest if each action is given a label as in:

```
DO CASE selector
label-1:
    sequence-1
label-2:
    sequence-2
  ....
label-n:
    sequence-n
ENDDO
```

Examples of DO CASE statements are:

```
DO CASE of command name
DO CASE switch setting
DO CASE error message number
```

7.4.3.8 Other Possible DO Constructs

The various DO constructs described above are only examples of the most common ones. The designer is perfectly free to invent some other form to fit the needs of a particular design. For example,

```
DO in parallel
```

might introduce a collection of labelled sequences which are to be executed in parallel, with synchronization automatically assured before proceeding past the ENDDO statement. As another example,

```
DO with interrupts disabled
```

might be used to introduce a sequence in which interrupts are not allowed.

7.4.4 The RETURN Statement

Normally, a flow segment will “return” to its “caller” when control reaches the end of the segment. However, the RETURN statement can be used to indicate premature exit from a flow segment. As with UNDO and CYCLE, the form

```
RETURN IF condition
```

is often useful. Some examples of RETURN statements are:

```
RETURN
RETURN IF end has been reached
RETURN Symbol’s Value
RETURN "Illegal Reference"
```

8. External Segments

It is frequently desirable to collect references to flow segments which are not part of the current design document. These might, for example, represent operating system services or utility operations which are defined in other design documents. Such segments are known as *external flow segments* and are declared in *external segments*.

An external segment is introduced by the command

```
%External   text
```

or

```
%E   text
```

where *text* is a sequence of characters to be used as the title of the external segment. The title will be displayed at the top of the external segment page and will be entered in the table of contents. Note that the "External" or "E" commands do *not*, themselves, declare external flow segments – they introduce segments *within which* external flow segments will be declared.

Examples of these commands are:

```
%External   Basic Utilities
%External   I/O Support Services
%E   General Storage Management Functions
```

The body of an external segment consists of *statements*, each of which represents the name of a flow segment not defined elsewhere in the design document. Each statement is scanned as if it were the argument of a "Segment" command (see Chapter 7) and the resulting name is entered into the dictionary as the name of a flow segment. If a statement begins with a comment string (see Section 2.7.3), no scanning is performed.

White space on input lines is kept and blank input lines produce blank output lines.

An example of the input form of an external segment is:

```
%External   Low-Level I/O Operations
Open File (file-name)
Close File (file-id)
Read (file-id, into, max-bytes)
Write (file-id, from, count)
```

9. Text Functions

Text functions are used to insert special information into a design or to perform some kind of textual modification. They are most commonly used in text segments but may appear in any type of segment.

The general form of a text function invocation is

```
#{name[;argument;argument;...]}
```

where *name* is the name of the text function and the *arguments* depend upon the requirements of the particular function. If an argument to a text function contains any of

```
#{   {   }  ;
```

each must be preceded by an escape character ("\") as described in Section 2.4.3. The entire text function invocation must appear on a single (possibly continued) source line.

9.1 The DATE Text Function

The date on which the current run of PDL/81 was started may be obtained by

```
#{date}
```

which will be replaced by the date in the same form as it appears at the top of each page of the design. For example, the line

```
The current date is #{date}
```

will be printed as

```
The current date is 12 December 1991
```

9.2 Underscoring of Text

Keywords and secondary keywords in flow segments may be automatically underscored by use of the “UScore” command as described in Section 7.4.1.1. Other text may be underscored by

```
#{us;text}
```

which causes each non-blank character in “text” to be underscored and by

```
#{uc;text}
```

which causes each character in “text” to be underscored. For example,

```
this #{us;is under}scored and #{uc;so is this}
```

will print as

this is underscored and so is this

If bold face output is supported at your installation on the selected device, the text function

```
#{bf;text}
```

will print *text* in bold face,

```
#{bfu;text}
```

will print *text* in bold face with non-blank characters underscored, and

```
#{bfuc;text}
```

will print *text* in bold face with all characters underscored.

9.3 Tags and References

A *tag* is a symbol which can be used to mark a particular point in a design and is declared by

```
%Tag   symbol
```

where *symbol* is the name of the tag. The output page number corresponding to the location of the tag will be associated with the tag and may be retrieved by the text function

```
#{ref;symbol}
```

For example, if the command

```
%Tag  test
```

appeared at a point which was to be printed on page five of the design document, the line

```
See page #{ref;test} for a description.
```

would print as

```
See page 5 for a description.
```

10. Listing Control Commands

This chapter describes a number of commands which are used to control various aspects of the listing of the design document.

10.1 Specifying Design Titles

The title of the design may be specified by commands of the form

```
%Title   text
```

where *text* is any sequence of characters. Several "Title" commands may be used in a single design. The text of these commands will be placed, centered and boxed, double spaced, with leading and trailing blanks removed, on the cover page of the design document. In addition, the text of the first "Title" command will be capitalized and placed at the top of each design page unless a "Ptitle" seePDL/81 Design Language Reference Guide command is used.

Some examples are:

```
%Title   Fortran Compiler: Pass 3
%Title   Tree Transformation Phase
```

The "Title" commands should appear before the first segment.

10.1.1 Defining a Page Head

The command

```
%PTitle   text
```

will cause the text to be used as the running page head for the design. If this is not used, the running head will be the text of the first %Title command.

10.2 Specifying the Listing Date

Normally, the date on which the current PDL/81 run was started is the date displayed on the design title page and at the top of the other pages of the design and is the date returned by the “date” text function (see Section 9.1). The date may be changed by the command

```
%Date   string
```

where the first nine characters of *string* will be used as the date. No checking is performed on this substitute date and it will be used as is in place of the system date.

Some examples are:

```
%Date   6 May 91
%Date   6.5.91
%Date   5/6/91
%Date   91/06/05
```

If used, the “Date” command should appear before the first segment.

10.3 Specifying Security Banners

A security banner will be placed at the top and bottom of each output page by the command

```
%Security   classification
```

where *classification* is a word or phrase specifying the security classification of the design document. The command

```
%Project text
```

specifies “text” to be a project identification word or phrase to be included in the security banner. The “Project” command will be ignored in the absence of a “%Security” command. Both of these commands, if used, should appear before the first segment.

In addition to the security classification and optional project name, each security banner will contain a sequential sheet number for document control purposes. These numbers start at “one” for the title page and are incremented by one for each sheet printed. They are independent of the page numbers assigned by PDL/81 for reference purposes. The last page of the design will contain a count of the total number of sheets printed.

As an example, the security banners which appear on the sample design at the end of this manual were specified by

```
%Security   UNCLASSIFIED
%Project    PDL/81 SAMPLE DESIGN
```

10.3.1 Security Banner Style

The format of security banners may be changed to reflect various standards.

By default, banners will have the classification centered, the sheet number on the right, and the project identification on the left. This mode is known as *Security Style* 0.

Security style 1 is the same as zero except that the sheet number will appear on the left and the project identification will appear on the right on even numbered sheets. This is useful when printing duplexed designs.

Security style 2 places the project identifiaction in the center, the classification on the left (right on bottom banner), and the sheet count on the right (left on bottom banner).

The default security style may be changed by editing the style files. On a per-document basis, the command

```
%SecStyle   number
```

where the number is one of the security styles (0, 1, or 2) will set the security style in a design.

10.4 Specifying “Special” Boxes

Experience has shown that designs are often printed on serial printers since such printers are available with compressed type fonts which allow a full-width design to be printed on 8-1/2 by 11 inch paper. These printers can be very slow, however, when printing designs because of the large amount of white space which may appear between the end of a statement and the right edge of the segment box. The command

```
%SBox
```

specifies that the right edge of all boxes is not to be printed. This usually results in faster printing during the draft stage.

The command

```
%NoSBox
```

specifies that the right edge of all boxes is to be printed (the default case).

If either of these commands is used, it should appear prior to the first segment.

10.5 Specifying Line Number Printing

The PDL/81 processor does not normally display source line numbers in the design document. This can be changed by the

```
%LNO
```

which causes source line numbers to appear to the right of the segment box. Not all lines will be numbered. The display of line numbers may be stopped by

```
%NOLNO
```

10.6 Specifying Change Bars

Change bars, which appear on the right of segment boxes, can be displayed by the command

```
%MC   char
```

where *char* is a single character to be used as the change bar. If *char* is absent, the display of change bars is stopped.

For example, bracketing a section of changed design with

```
%MC   |
```

and

```
%MC
```

will cause the character "|" to be used as a change bar for that section of the design.

11. Advanced Features

This chapter describes several advanced features of the *design* style including:

- Cyclomatic complexity measurement and reporting;
- Automatic requirements tracking;
- Consistency checking;
- Flow figure enhancement;
- Maintaining design and code in the same file.

11.1 Complexity Analysis

This version supports a form of cyclomatic complexity measurement based on the work of McCabe (*A Complexity Measure*, McCabe, Thomas J., IEEE Transactions on Software Engineering, Vol SE-2, No 4, Dec 1976). It performs this measurement by assigning a complexity value to certain keywords and secondary keywords and summing these values for each flow segment – the higher the value, the more complex the segment.

The complexity for each flow segment is printed on the segment's output page and in the *Index of Flow Segments* (Section 12.3). If a segment's complexity exceeds some specified value (6, by default), a warning message is issued on the standard output and also appears on the segment's output page. An index to such overly complex segments is also printed (Section 12.4). Finally, various complexity statistics are given on the summary page at the end of the design document.

11.1.1 Complexity Measurement Commands

The command

```
%Complexity  [max]
```

specifies that complexity measurement is to be performed. If *max* is given, it should be an integer constant giving the maximum allowable complexity to use instead of the default value of 6.

The command

```
%NoComplexity
```

specifies that complexity measurement is not to be performed.

11.2 Automatic Requirements Tracking

Information about requirements are input by the command

```
%Req   r1;r2;...;rn
```

or

```
%R   r1;r2;...;rn
```

where each of the "r*i*" is a paragraph number of a requirement taken from the controlling requirements document. When used with DOD-STD 2167, this might be the *Software Requirements Specification*, DI-MCR-80025. The paragraph numbers must have the general form of section and subsection identifiers separated by decimal points. Such an identifier is a decimal integer optionally prefixed by an alphabetic string. Examples are

```
3.5.6   3.9.6.2   R4.2   4.6.R3.2
```

If a segment has associated requirements, the %R commands for the segment must immediately follow the segment command (e.g., %S, %D, %T). When a segment which references requirements is printed, the associated requirements will be displayed following the segment box.

11.2.1 Requirements Index

Optionally, an index of all requirements and their associated segments may be printed. This is accomplished by using the command

```
%RIndex
```

If this action is established as the default during installation, it may be suppressed by the command

```
%NoRIndex
```

11.3 Consistency Checking

This release provides for consistency checking of segment references in a style that is in the spirit of PDL/81. This is done by optionally producing a report known as the *Calls-In-Context List* which shows each flow segment definition and a listing of each line that calls that segment. Those calls which appear to be inconsistent in number of arguments with the definition are flagged in the report. For the purpose of this report, an argument list is assumed to be enclosed in parentheses and arguments are separated by zero-level (with respect to parentheses and single and double quotation marks).

The report is requested by the command

```
%CiC
```

If this action is established as the default during installation, it may be suppressed by the command

```
%NoCiC
```

11.4 Flow Figure Enhancement

The command

```
%KwV
```

specifies that the beginning and end of each flow figure will be connected by a series of a predetermined character. This may be turned off by the command

```
%NoKwV
```

A default character and font may be established by editing the style and/or device definition files. The character and font may be chosen on a per-device basis so that advantage may be taken of any specialized device characteristics (e.g., a line-drawing character set).

The character and font may be set on a per-design basis by the command

```
%KwVC  char[;font-expr]
```

If the font is not specified, the base font will be used.

11.5 Design and Code in the Same File

This new feature of PDL/81 allows maintaining both the design and the code for a program in the same file. Code sequences, known as *code segments*, are introduced by the command

```
%Code  [file-name]
```

where *file-name* is the name of a file to receive this code when code selection is enabled. If the file name is not specified, the code will be written to the file name given in the last preceding %Code command that had a file name or, if none such exist, to the file specified by the last preceding

```
%CodeFile  [file-name]
```

command. If no file name is in effect, code is written to the standard output.

During normal runs of PDL/81, code segments are *not* output; rather, they are completely skipped. To cause code segment selection, invoke PDL/81 with the "GetCode" number register set as in

```
pdl81 -rGetCode file
```

As a final option, a normal design run of PDL/81 can be made with code segments being displayed in the output document. This is accomplished by invoking PDL/81 with the "ShowCode" number register set as in

```
pdl81 -rShowCode file
```

If this is done, code segments may *not* contain sequences which look like invocations of Format Design Language functions.

12. Processor Reports

Several types of reports can be printed which provide information about the content and structure of the design. The designer may choose the specific reports to be included.

12.1 Segment Reference Trees

This report shows the nesting of flow segment references. A separate tree is printed for each *root segment*, which is a flow segment that is not referenced by any flow segment but which, itself, references at least one flow segment. If there are no such root segments, an arbitrary choice will be made.

When a segment is referenced recursively, its name is prefixed by an asterisk and the recursion is not further traced.

The presence or absence of this report is controlled by the command

```
%Tree
```

which specifies that the report is to be printed, and by

```
%NoTree
```

which specifies that the report is not to be printed.

A special abbreviated form of the trees can be selected by the command

```
%STree
```

In these so-called *short trees*, only the first occurrence of each subtree is printed. For subsequent occurrences, only the name of the first segment in the subtree will be printed, prefixed with a minus sign ("-").

If any of these commands are used, they should appear before the first segment. The default setting is “STree”.

12.2 Data Item Index

The data item index shows each data item which was implicitly or explicitly declared in the design and the locations in the design where each is referenced. The data item index is requested by

```
%DIndex
```

and is inhibited by

```
%NoDIndex
```

If either of these commands is used, it should appear before the first segment. The default setting is “DIndex”.

12.3 Flow Segment Index

The flow segment index lists all flow segments (both internal and external) in the design. For each, it shows the location of its definition and the segment names and locations of all references to it.

The flow segment index is requested by

```
%SIndex
```

and is inhibited by

```
%NoSIndex
```

If either of these commands is used, it should appear prior to the first segment. The default setting is “SIndex”.

12.4 Index of Overly Complex Segments

If complexity measurement is enabled (Section 11.1), this index will be printed if any segments excede the predefined maximum allowable complexity. By default, this value is 6.

The index will show the complexity, type, location, and name of each segment with too high a complexity.

12.5 Index to Requirements

If requirements tracking is enabled (Section 11.2), an index of requirements will be printed. This index will be sorted by requirement and will show the location and name of each segment that addresses that requirement.

12.6 Calls-in-Context List

When enabled (Section 11.3), this listing will show each procedure or function call together with its definition. Inconsistent usage will be flagged.

A. Error Messages

This Appendix lists error messages which may be issued during processing of a design. Error messages are displayed on the standard error file. If applicable, the message will be prefixed with the name of the current input file and the current line number within the file.

A.1 Non-Terminal Error Messages

The error messages described in this section do not cause termination of PDL/81 processing:

- COMMAND INVALID OUTSIDE OF SEGMENT – this command may only appear within a segment.
- COMMAND INVALID OUTSIDE OF TEXT SEGMENT – this command may only be used within a text segment.
- DUPLICATE DATA ITEM: <item> – the named item has been previously defined as a data item.
- DUPLICATE ENTRY POINT: <name> – the given name has previously been defined as the name of a flow segment or of an entry point.
- DUPLICATE GLOBAL NAME: <name> – the given name has already been declared as global in a Specification segment.
- DUPLICATE NAME: <name> – the given name, which appears as the argument of a “Procedure” or “Function” command, has been previously defined as the name of a flow segment.
- DUPLICATE TAG: <name> – the given name has been previously defined in another “Tag” command.
- ENDING KEYWORD WITH NO OPEN FLOW FIGURE – an ending keyword, such as ENDIF, was encountered but a flow figure is not open for it to close.
- FLOW FIGURE NOT CLOSED AT END OF SEGMENT – a flow figure is still open when the end of a segment was encountered.
- INVALID CHARACTER IN LINE – an input line contains an ASCII control character other than “tab” or “newline”.

- NAME MISSING – a name was not provided for a group or a segment.
- REQUIREMENTS MUST BE PART OF A SEGMENT – A “Req” or “R” command has been encountered but it is precedes the first segment or immediately follows a “Group” statement.
- SEGMENT TOO COMPLEX – the cyclomatic complexity of the segment is greater than the allowable maximum.
- TEXT OUTSIDE OF SEGMENT – a source line which was not a command appeared outside of a segment. A generated “Segment” command will be inserted.
- UNBALANCED BRACKETS – the number of unescaped left brackets is not the same as the number of unescaped right brackets within a call on a text function.
- UNDEFINED TAG: <name> – the given name was referenced in a “Ref” text function but did not occur also in a “Tag” command.
- UNKNOWN COMMAND – a command name on a command line is not one of those recognized by PDL/81.

A.2 Terminal Error Messages

The error messages described in this section cause immediate termination of PDL/81 processing:

- CAN’T OPEN TEMP FILE <file name> – the named temporary file cannot be opened. This usually means that disk space is not available for the file or that write access privileges are not available in the directory on which the file is to be written.
- CANNOT ALLOCATE DYNAMIC MEMORY FOR A BUFFER – Memory was needed for an input/output buffer, but insufficient memory was available.
- DYNAMIC MEMORY OVERFLOW (n) – all available dynamic memory is allocated and more is needed. The character “n” indicates the particular point in the processor where overflow was detected and is of interest only to PDL/81 processor maintenance personnel.
- MKTEMP: CANNOT GENERATE UNIQUE FILE NAME: <file name> – Names of PDL/81 temporary files are generated by the internal PDL/81 “mktemp” function. This function can generate up to 26 unique names for each invocation of PDL/81. Since names will be reused when possible, and since PDL/81 deletes temporary files after they are closed, this message usually means that a large number of temporaries were left around following a system crash. Examine the directory given in the message and delete the abandoned temporaries.
- SOURCE FILE NOT GIVEN – a source file was not specified when PDL/81 as invoked.
- UNABLE TO OPEN FILE <file name> – the named file cannot be opened for input. Possibly, it doesn’t exist.
- UNKNOWN DEVICE TYPE: <name> – the named device type was specified by an invocation option but no such device is supported.

- UNKNOWN INVOCATION OPTION: <option> – the invocation line contained an option which was not recognized by PDL/81.

A.3 Other Error Messages

The error messages described above are those which relate to processing designs using the *design* document style. Other messages may be issued but they relate to internal processing errors or system problems and should not appear when processing designs. A more complete list of such messages may be found in the {it PDL/81 Format Designers Guide}.

B. List of Commands

BL	start a “bullet” list
CIC	enable calls-in-context index printing
Code	start a Code segment
CodeFile	define a file to receive extracted code
Complexity	enable complexity measurement
CString	define comment strings
D	start a data segment
Data	start a data segment
Date	define date for printing purposes
DChar	define data characters
DIndex	print a data item index
DSChar	define data item special characters
E	start an external segment
Eject	begin a new page of output
External	start an external segment
Fill	switch to formatted mode in a text segment
G	start a group
Group	start a group
Heading	print a second level heading
Include	include source from an alternate file
KWFont	specify font in which to print keywords
KWV	enable flow figure enhancement
KWVC	define character for flow figure enhancement

LCase	print keywords in lower case
Le	end a list
LNO	start display of source line numbers
MajorHeading	print a first level heading
MC	start or stop display of change bars
Need	assure enough lines remain on a page
NL	start a numbered list
NoCIC	disable calls-in-context index
NoComplexity	disable complexity measurement
NoDIndex	do not print data index
NoFill	switch to unformatted mode in a text segment
NoKWV	disable flow figure enhancement
NOLNO	stop display of source line numbers
NoRindex	disable requirements index
NoSBox	do not print special boxes
NoSIndex	do not print a flow segment index
NoTree	do not print reference trees
NoUScore	do not underscore keywords
Project	specify name of project for security banners
PTitle	define running page title
R	specify requirements
Req	specify requirements
Rindex	enable requirements index
S	start a flow segment
SBox	use special segment boxes
SCase	print keywords in same case as entered
SecStyle	specify style of security banners
Security	specify security classification of design
Segment	start a flow segment
SIndex	print flow segment index
Space	space a given number of blank lines
STree	print short reference trees
SubHeading	print a third level heading
T	start an unformatted text segment
Tag	define a tag

Text	start an unformatted text segment
TextF	start a formatted text segment
TF	start a formatted text segment
Title	specify design titles
Tree	print reference trees
UCase	print keywords in upper case
UScore	underscore keywords
Verb	put a verb in a verb list
VL	start a verb list

C. Adding New Keywords

Project-wide keywords and secondary keywords can be added by modifying the style file for the particular design style being used. New keywords and secondary keywords for use in a single design can be defined by the functions defined in this Appendix.

C.1 Defining Primary Keywords

The “kw” function defines new keywords and has the form

```
#{kw;name;pre;post;complexity;class;code;flags}
```

where the arguments are

name	the keyword name.
pre	optionally signed integer giving the number of indentation stops to indent prior to printing the line which begins with the keyword.
post	optionally signed integer giving the number of indentation stops to indent after printing the line which begins with the keyword.
complexity	integer specifying the cyclomatic complexity associated with this keyword.
class	used in flow figure checking (Section C.3).
code	used in flow figure checking (Section C.3).
flags	flags to associate with the keyword. The only currently defined flags is “1” which suppresses reference collection for the remainder of the line which starts with the keyword.

For example, the function

```
#{kw;case;-1;1}
```

will define “case” to be a keyword which will be printed one stop to the left of the current position. Following lines will be printed one stop to the right of the posi-

tion in which the keyword is printed. With this definition, the sequence

```
do case type
case open:
...
case close:
...
case delete:
...
enddo
```

will result in

```
DO CASE type
CASE open:
    ...
CASE close:
    ...
CASE delete:
    ...
ENDDO
```

C.2 Defining Secondary Keywords

Secondary keywords are defined by the “skw” function which has the form

```
#{skw;name;complexity}
```

where the arguments are

name — the keyword name.

complexity — integer specifying the cyclomatic complexity associated with this keyword.

For example

```
#{skw;loop}
```

will define “loop” to be a secondary keyword. Note that a word may be both a keyword and a secondary keyword.

C.3 Keyword Classes and Codes

Flow figure checking is controlled by *class/code* pairs which are associated with each keyword. These pairs are defined using the “kw” text function.

The classes are used to distinguish between flow figures and are positive numbers. The codes are used to distinguish among types of keywords within a given flow figure and are in the range 0–5, inclusive. The codes are used to access the state table

	0	**1**	**2**	**3**	**4**	**5**
0	0	1	4	5	5	5
1	0	1	2	3	3	3
2	-	-	-	-	-	-
3	0	1	2	6	3	3
4	0	1	2	6	3	3
5	0	1	2	6	6	6

where the column headings correspond to the *code* of an incoming keyword, the row headings correspond to the *code* at the top of the *keyword stack*, and the table entries are *actions* to be performed.

The *codes* may be thought of as corresponding to

0 As an incoming keyword, this is one that doesn't make any change in the structure (such as *RETURN* or *UNDO*). Since these are never pushed on the stack, a *code* of 0 on the stack means the stack is empty.

1 A figure-opening keyword (such as *DO* or *IF*).

2 A figure-ending keyword (such as *ENDDO* or *ENDIF*).

3 An intermediate keyword that may only occur once following a figure-opening keyword.

4 An intermediate keyword that may occur any number of times after a *code* 2 or 3 keyword. *ELSEIF* is an example.

5 An intermediate keyword that may occur once after a *code* 2, 3, or 4 keyword and may not be followed by a *code* 3 or 4 keyword. *ELSE* is an example.

A separate *class* should be assigned to each flow figure. For example, the *IF...ENDIF* figure might be *class* 1 and the *DO...ENDDO* figure might be *class* 2.

The *actions* are

0 Do nothing.

1 Push the *class* and *code* of the incoming keyword onto the stack.

2 If the incoming *class* matches that on the top of the stack, pop the stack. Otherwise, issue an error message.

3 If the incoming *class* matches that on the top of the stack, pop the stack and push the *class* and *code* of the incoming keyword onto the stack. Otherwise, issue an error message.

4 Issue an error message.

5 Issue an error message.

6 Issue an error message.

C.4 Placement of Keyword Definitions

Uses of the “kw” and “skw” functions should appear, one per line, prior to the first segment in a design.

D. Sample PDL/81 Design

This Appendix presents a short example of a PDL/81 design. It is followed by a listing of the input source which resulted in the design listing.

D.1 Design of an Automobile Cruise Control System

This sample presents a top-level design of a mythical automobile cruise control system. It illustrates many of the PDL/81 features used in a typical design. Note that security banners are used to illustrate the distributed format. Of course, the format can be easily changed to suit the requirements of a particular program.

D.1.1 Output of PDL/81 Processor

Beginning on the next page is the actual output of the PDL/81 processor when presented with the source shown in Section D.2.

CAINE, FARBER & GORDON, INC.
1010 EAST UNION STREET
PASADENA, CALIFORNIA 91106

```
****************************************
*                                      *
* An Automobile Cruise Control Example *
*                                      *
*              21 Jan 92               *
*                                      *
*           PDL/81 X2.0.911            *
*                                      *
*               5500-PD8               *
*                                      *
****************************************
```

TABLE OF CONTENTS

```
****************
*              *
* Requirements *
*              *
****************
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  3
21 Jan 92       Requirements

     Statement of Problem


     ###############################################################################
     #                                                                             #
     #  1    Design a combined automobile cruise control and trip computer       #
     #  2    with the following characteristics:                                 #
     #  3                                                                        #
     #  4    1.      Display Fuel Consumption Rate                               #
     #  5                                                                        #
     #  6            1.1     Each time fuel is added, the driver enters the      #
     #  7                    amount added and the system computes and            #
     #  8                    displays the consumption.                           #
     #  9                                                                        #
     # 10    2.      Display Average Speed                                       #
     # 11                                                                       #
     # 12            2.1     Each time the driver requests an average speed,     #
     # 13                    the system displays the average speed since the     #
     # 14                    "trip start".                                       #
     # 15                                                                       #
     # 16            2.2     The driver may set "trip start".                    #
     # 17                                                                       #
     # 18    3.      Display Maintenance Messages                                #
     # 19                                                                       #
     # 20            3.1     Oil and oil filter change every  5000 miles.        #
     # 21                                                                       #
     # 22            3.2     Air filter change         every 10000 miles.        #
     # 23                                                                       #
     # 24            3.3     Major service             every 15000 miles.        #
     # 25                                                                       #
     # 26            3.4     Display intermittent message 250 miles before       #
     # 27                    service required.                                   #
     # 28                                                                       #
     # 29            3.5     Display continuous message 50 miles before          #
     # 30                    service required.                                   #
     # 31                                                                       #
     # 32            3.6     The driver informs the system when service          #
     # 33                    is completed.                                       #
     # 34                                                                       #
     # 35    4.      Control Cruising Speed                                      #
     # 36                                                                       #
     # 37            4.1     The system can control the throttle to maintain     #
     # 38                    cruising speed.  For simplicity, it is assumed      #
     # 39                    that the brakes will never be needed for speed      #
     # 40                    control.                                            #
     # 41                                                                       #
     # 42            4.2     The cruising speed is the speed the car is          #
     # 43                    traveling when the cruise control is activated.     #
     # 44                                                                       #
     # 45            4.3     Activation and Deactivation                         #
     # 46                                                                       #
     # 47                    4.3.1   To activate the cruise control, the         #
     # 48                            engine must be running and the car must     #
     # 49                            be traveling 30 miles per hour or           #
     # 50                            faster.                                     #
     # 51                                                                       #
     # 52                    4.3.2   The cruise control can be deactivated       #
     # 53                            at any time.                                #
     # 54                                                                       #
     #                                                                          #
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                PAGE  3.001
21 Jan 92       Requirements

     Statement of Problem   (continued)

     #                                                                          #
     # 55                    4.3.3   The cruise control is always deacti-       #
     # 56                            vated when the engine is turned off.      #
     # 57                                                                       #
     # 58            4.4     Suspension and Resumption                          #
     # 59                                                                       #
     # 60                    4.4.1   Operation of the cruise control is         #
     # 61                            temporarily suspended whenever the        #
     # 62                            driver uses the accelerator or brake.     #
     # 63                                                                       #
     # 64                    4.4.2   If the cruise control is suspended but     #
     # 65                            has not been deactivated, the driver      #
     # 66                            can tell the system to resume operation   #
     # 67                            and it will return the car to the         #
     # 68                            cruising speed.                           #
     # 69                                                                       #
     # 70            4.5     The driver can increase cruising speed by          #
     # 71                    telling the system to increase speed and then      #
     # 72                    telling it to stop increasing.                     #
     # 73                                                                       #
     # 74    5.      Calibration                                                #
     # 75                                                                       #
     # 76            5.1     The system must maintain accurate mileage.         #
     # 77                    This is done by periodic calibration.              #
     # 78                                                                       #
     # 79            5.2     To calibrate the system, the car is driven over    #
     # 80                    a measured mile and the driver informs the         #
     # 81                    system at the beginning and the end.               #
     # 82                                                                       #
     # 83            5.3     The cruise control must be off during              #
     # 84                    calibration.                                       #
     #                                                                          #
     ############################################################################
```

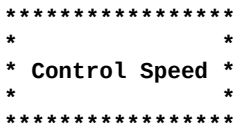
```
*********************************
*                               *
* Cruise Control and Monitoring *
*                               *
*********************************
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  5
21 Jan 92       Cruise Control and Monitoring


     Main Program Task

REF    (CX = 1)
PAGE ********************************************************************************
     *                                                                              *
   6 *  1    initialize asynchronous determine speed task                          *
     *  2    set current_state to inactive                                         *
     *  3    set within_mile to off                                                *
     *  4    DO FOREVER                                                            *
  16 *  5    |   monitor automobile                                                *
  10 *  6    |   control speed                                                     *
   8 *  7    |   measure mile                                                      *
     *  8    ENDDO                                                                 *
     *                                                                              *
     ********************************************************************************

     Requirements: 1, 2, 3, 4, 5
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  6
21 Jan 92       Cruise Control and Monitoring


     Initialize Asynchronous Determine Speed Task

REF    (CX = 0)
PAGE ********************************************************************************
     *                                                                              *
     *  1    set last_time to clock_time                                            *
     *  2    set pulse_counts_per_mile to a default value                           *
   7 *  3    SPAWN determine speed task                                             *
     *                                                                              *
     ********************************************************************************

     Requirements: 1, 2, 3, 4, 5
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  7
21 Jan 92       Cruise Control and Monitoring


     Determine Speed Task

REF    (CX = 2)
PAGE ********************************************************************************
     *                                                                              *
     *  1    DO FOR each shaft_rotation                                             *
     *  2    |   set current time to clock_time                                     *
     *  3    |   set time per rotation to (current time - last_time)                *
     *  4    |   set last_time to current time                                      *
     *  5    |   set distance per rotation to (1/pulse_counts_per_mile)             *
     *  6    |   set current_speed to (distance per rotation/time per rotation)     *
     *  7    |   add distance per rotation to cummulative_distance                  *
     *  8    |   IF cummulative_distance is greater than one mile                   *
     *  9    |   |   increment current_mileage                                      *
     * 10    |   |   subtract one mile from cummulative_distance                    *
     * 11    |   ENDIF                                                              *
     * 12    ENDDO                                                                  *
     *                                                                              *
     ********************************************************************************

     Requirements: 1, 2, 3, 4, 5
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  8
21 Jan 92       Cruise Control and Monitoring


     Measure Mile

REF    (CX = 6)
PAGE ********************************************************************************
     *                                                                              *
     *  1    IF current_state is not inactive or engine_running is false           *
     *  2    |   set within_mile to ignore                                         *
     *  3    ELSEIF within_mile is off and measure_signal is on                    *
     *  4    |   set within_mile to on                                             *
     *  5    |   reset pulse_counter                                               *
     *  6    ELSEIF within_mile is on and measure_signal is on                     *
     *  7    |   IF there has been a shaft_rotation                                *
     *  8    |   |   increment pulse_counter                                       *
     *  9    |   ENDIF                                                             *
     * 10    ELSEIF within_mile is on and measure_signal is off                    *
     * 11    |   set pulse_counts_per_mile to pulse_counter                        *
     * 12    |   set within_mile to off                                            *
     * 13    ELSEIF measure_signal is off                                          *
     * 14    |   set within_mile to off                                            *
     * 15    ENDIF                                                                 *
     *                                                                              *
     ********************************************************************************

     Requirements: 5
```


Control Speed

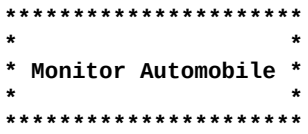
```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  10
21 Jan 92       Control Speed


     Control Speed

REF    (CX = 7)  ### SEGMENT TOO COMPLEX ###
PAGE ********************************************************************************
     *                                                                              *
     *  1    IF there is a driver_request                                          *
  11 *  2    |   get new current state                                             *
     *  3    ENDIF                                                                 *
     *  4    IF current_state is inactive or engine_running is false               *
     *  5    |   set desired_speed to zero                                         *
     *  6    |   set current_state to inactive                                     *
  14 *  7    ELSEIF pedal override                                                 *
     *  8    |   set current_state to suspended                                    *
     *  9    ELSEIF brake_engaged is true                                          *
     * 10    |   set current_state to suspended                                    *
     * 11    ELSEIF current_state is cruising                                      *
  12 * 12    |   maintain speed                                                    *
     * 13    ELSEIF current_state is accelerating                                  *
     * 14    |   IF current_speed is close to or above desired_speed               *
     * 15    |   |   set current_state to cruising                                 *
     * 16    |   ELSE                                                            *
  13 * 17    |   |   increase speed                                                *
     * 18    |   ENDIF                                                           *
     * 19    ENDIF                                                                 *
     *                                                                              *
     ********************************************************************************

     Requirements: 4
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  11
21 Jan 92       Control Speed


     Get New Current State

REF    (CX = 3)
PAGE ********************************************************************************
     *                                                                              *
     *  1    DO CASE of driver_request                                             *
     *  2    increase_speed:                                                       *
     *  3    |   set desired_speed to infinity                                     *
     *  4    |   set current_state to accelerating                                 *
     *  5    resume_speed:                                                         *
     *  6    |   IF desired_speed is not zero or infinity                          *
     *  7    |   |   set current_state to accelerating                             *
     *  8    |   ENDIF                                                             *
     *  9    maintain_speed:                                                       *
     * 10    |   IF current_speed is above 30 mph                                  *
     * 11    |   |   set desired_speed to current_speed                            *
     * 12    |   |   set current_state to cruising                                 *
     * 13    |   ELSE                                                              *
     * 14    |   |   set current_state to inactive                                 *
     * 15    |   ENDIF                                                             *
     * 16    inactivate_cruise:                                                    *
     * 17    |   set current_state to inactive                                     *
     * 18    ENDDO                                                                 *
     *                                                                              *
     ********************************************************************************

     Requirements: 4.3, 4.4, 4.5
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                 PAGE  12
21 Jan 92       Control Speed


     Maintain Speed

REF    (CX = 1)
PAGE ********************************************************************************
     *                                                                              *
     *  1    IF current_speed is much above desired_speed                          *
     *  2    |   ..deceleration is needed                                          *
     *  3    |   set throttle_position to closed                                   *
     *  4    ELSE                                                                  *
     *  5    |   set correction to current_speed - desired_speed                   *
     *  6    |   ..normal throttle position is a function of speed                 *
     *  7    |   set throttle_position the normal position plus                    *
     *  8    |   some function of correction                                       *
     *  9    ENDIF                                                                 *
     *                                                                              *
     ********************************************************************************

     Requirements: 4.1
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                 PAGE  13
21 Jan 92       Control Speed


     Increase Speed

REF    (CX = 0)
PAGE ********************************************************************************
     *                                                                              *
     *  1    ..normal throttle position is a function of speed                      *
     *  2    ..acceleration is also a function of speed                             *
     *  3    set throttle_position to normal position plus acceleration             *
     *                                                                              *
     ********************************************************************************

     Requirements: 4.1
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  14
21 Jan 92       Control Speed


     Pedal Override

REF    (CX = 1)
PAGE ********************************************************************************
     *                                                                              *
     *  1    ..pedal_deflection is an input                                         *
     *  2    ..throttle_position is an output                                       *
     *  3    IF pedal_deflection is greater than throttle_position                  *
     *  4    |   ..the driver is accelerating                                       *
     *  5    |   RETURN true                                                        *
     *  6    ELSE                                                                  *
     *  7    |   RETURN false                                                       *
     *  8    ENDIF                                                                 *
     *                                                                              *
     ********************************************************************************

     Requirements: 4.4.1
```


Monitor Automobile

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  16
21 Jan 92       Monitor Automobile


     Monitor Automobile

REF    (CX = 0)
PAGE ********************************************************************************
     *                                                                              *
  17 *  1    compute trip average                                                   *
  19 *  2    lookup maintenance schedule                                            *
  18 *  3    compute miles per gallon                                               *
     *                                                                              *
     ********************************************************************************

     Requirements: 1, 2, 3
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  17
21 Jan 92       Monitor Automobile


     Compute Trip Average

REF    (CX = 2)
PAGE ********************************************************************************
     *                                                                              *
     *  1    IF activate_trip_average                                               *
     *  2    |   set trip_mileage to current_mileage                                *
     *  3    |   set trip_clock to clock_time                                       *
     *  4    ELSEIF get_trip_average                                                *
  21 *  5    |   display number ((trip_mileage - current_mileage) / (trip_clock  *
     *       |           - clock_time))                                             *
     *  6    ENDIF                                                                  *
     *                                                                              *
     ********************************************************************************

     Requirements: 2
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  18
21 Jan 92       Monitor Automobile


     Compute Miles Per Gallon

REF    (CX = 1)
PAGE ********************************************************************************
     *                                                                              *
     *  1    IF fuel_added                                                         *
  21 *  2    |   display number ((last_fuel - current_mileage) / fuel_added)      *
     *  3    |   set last_fuel to current_mileage                                  *
     *  4    ENDIF                                                                 *
     *                                                                              *
     ********************************************************************************

     Requirements: 1
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                 PAGE  19
21 Jan 92       Monitor Automobile


     Lookup Maintenance Schedule

REF    (CX = 4)
PAGE ********************************************************************************
     *                                                                              *
     *  1    IF there is a maintenance_reset_command                                *
     *  2    |   determine maintenance_type from maintenance_reset_command          *
     *  3    |   set last_reset for maintenance_type to current_mileage             *
     *  4    ENDIF                                                                  *
     *  5    DO FOR each maintenance_type ..oil_filter, air_filter,                 *
     *       |       major_service                                                  *
     *  6    |   set miles_since_maintenance to (current_mileage - last_reset       *
     *       |           for maintenance_type)                                      *
     *  7    |   set difference to (maintenance_interval for maintenance_type -     *
     *       |           miles_since_maintenance)                                   *
     *  8    |   IF difference is greater than 250                                  *
     *  9    |   |   set maintenance_message of maintenance_type to off             *
     * 10    |   ELSEIF difference is greater than 50                               *
     * 11    |   |   set maintenance_message of maintenance_type to intermittent    *
     * 12    |   ELSE                                                               *
     * 13    |   |   set maintenance_message of maintenance_type to on              *
     * 14    |   ENDIF                                                              *
     * 15    ENDDO                                                                  *
     *                                                                              *
     ********************************************************************************

     Requirements: 3
```

Maintenance Data

```
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
D                                                                             D
D                                                                             D
D       maintenance_type is oil_filter, air_filter, major_service             D
D       ..the following items are arrays of maintenance_type                  D
D       maintenance_message                                                   D
D       maintenance_interval is 3000, 10000, 15000                            D
D       last_reset                                                            D
D                                                                             D
D                                                                             D
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
```

Requirements: 3

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  21
21 Jan 92       Monitor Automobile


     Display Number (number)

REF    (CX = 0)
PAGE ********************************************************************************
     *                                                                              *
     *  1    write the number in decimal to the display                             *
     *                                                                              *
     ********************************************************************************

     Requirements: 1.1, 2.1
```

```
**************************
*                        *
* SEGMENT REFERENCE TREES *
*                        *
**************************
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                PAGE  22.001
21 Jan 92       SEGMENT REFERENCE TREES


Main Program Task
-----------------


 LN   DEFN  SEGMENT
----  ----  -------

   1     5   Main Program Task
   2     6       Initialize Asynchronous Determine Speed Task
   3     7           Determine Speed Task
   4    16       Monitor Automobile
   5    17           Compute Trip Average
   6    21               Display Number
   7    19           Lookup Maintenance Schedule
   8    18           Compute Miles Per Gallon
   9    21               Display Number
  10    10       Control Speed
  11    11           Get New Current State
  12    14           Pedal Override
  13    12           Maintain Speed
  14    13           Increase Speed
  15     8       Measure Mile
```

```
**********************
*                    *
* INDEX TO DATA ITEMS *
*                    *
**********************
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                  PAGE  23.001
21 Jan 92       INDEX TO DATA ITEMS


INDEX TO DATA ITEMS
-------------------


PAGE  LINE  TYPE    NAME AND REFERENCES
----  ----  ----    -------------------

  17     1   ID      activate_trip_average
                        17  Compute Trip Average
                               1

  19     5   ID      air_filter
                        19  Lookup Maintenance Schedule
                               5

  10     9   ID      brake_engaged
                        10  Control Speed
                               9

   6     1   ID      clock_time
                        17  Compute Trip Average
                               3  5
                         7  Determine Speed Task
                               2
                         6  Initialize Asynchronous Determine Speed Task
                               1

   7     7   ID      cummulative_distance
                         7  Determine Speed Task
                               7  8  10

   7     9   ID      current_mileage
                        18  Compute Miles Per Gallon
                               2  3
                        17  Compute Trip Average
                               2  5
                         7  Determine Speed Task
                               9
                        19  Lookup Maintenance Schedule
                               3  6

   7     6   ID      current_speed
                        10  Control Speed
                               14
                         7  Determine Speed Task
                               6
                        11  Get New Current State
                               10  11
                        12  Maintain Speed
                               1  5

   5     2   ID      current_state
                        10  Control Speed
                               4  6  8  10  11  13  15
                        11  Get New Current State
                               4  7  12  14  17
```

INDEX TO DATA ITEMS (continued)

INDEX TO DATA ITEMS (continued)

INDEX TO DATA ITEMS (continued)

```
*************************
*                       *
* INDEX TO FLOW SEGMENTS *
*                       *
*************************
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                PAGE  24.001
21 Jan 92       INDEX TO FLOW SEGMENTS


INDEX TO FLOW SEGMENTS
----------------------


PAGE  LINE  TYPE CX NAME AND REFERENCES
----  ----  ---- -- ------------------

  18         FS   1  Compute Miles Per Gallon
                        16  Monitor Automobile
                              3

  17         FS   2  Compute Trip Average
                        16  Monitor Automobile
                              1

  10         FS   7  Control Speed
                         5  Main Program Task
                              6

   7         FS   2  Determine Speed Task
                         6  Initialize Asynchronous Determine Speed Task
                              3

  21         FS   0  Display Number
                        18  Compute Miles Per Gallon
                              2
                        17  Compute Trip Average
                              5

  11         FS   3  Get New Current State
                        10  Control Speed
                              2

  13         FS   0  Increase Speed
                        10  Control Speed
                              17

   6         FS   0  Initialize Asynchronous Determine Speed Task
                         5  Main Program Task
                              1

  19         FS   4  Lookup Maintenance Schedule
                        16  Monitor Automobile
                              2

   5         FS   1  Main Program Task

  12         FS   1  Maintain Speed
                        10  Control Speed
                              12

   8         FS   6  Measure Mile
                         5  Main Program Task
                              7

  16         FS   0  Monitor Automobile
```

INDEX TO FLOW SEGMENTS (continued)

PAGE	LINE	TYPE	CX	NAME AND REFERENCES
				5 Main Program Task
				5
14		FS	1	Pedal Override
				10 Control Speed
				7

```
**********************************
*                                *
* INDEX TO OVERLY COMPLEX SEGMENTS *
*                                *
**********************************
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                PAGE  25.001
21 Jan 92       INDEX TO OVERLY COMPLEX SEGMENTS


INDEX TO OVERLY COMPLEX SEGMENTS
--------------------------------


PAGE  TYPE  CX  SEGMENT NAME
----  ----  --  ------------

(MAXIMUM ALLOWABLE COMPLEXITY = 6)

  10   FS    7       Control Speed
```

```
**********************************
*                                *
* INDEX TO REQUIREMENTS REFERENCES *
*                                *
**********************************
```

INDEX TO REQUIREMENTS REFERENCES

```
************************
*                      *
* CALLS-IN-CONTEXT LIST *
*                      *
************************
```

```
CFG, INC.       AN AUTOMOBILE CRUISE CONTROL EXAMPLE                 PAGE  27.001
21 Jan 92       CALLS-IN-CONTEXT LIST


CALLS-IN-CONTEXT LIST
---------------------


F PAGE  NAME / CALL
- ----  -----------

    18   Compute Miles Per Gallon
    16   compute miles per gallon

    17   Compute Trip Average
    16   compute trip average

    10   Control Speed
     5   control speed

     7   Determine Speed Task
     6   spawn determine speed task

    21   Display Number (number)
    17   display number ((trip_mileage - current_mileage) / (trip_clock -
             clock_time))
    18   display number ((last_fuel - current_mileage) / fuel_added)

    11   Get New Current State
    10   get new current state

    13   Increase Speed
    10   increase speed

     6   Initialize Asynchronous Determine Speed Task
     5   initialize asynchronous determine speed task

    19   Lookup Maintenance Schedule
    16   lookup maintenance schedule

     5   Main Program Task

    12   Maintain Speed
    10   maintain speed

     8   Measure Mile
     5   measure mile

    16   Monitor Automobile
     5   monitor automobile

    14   Pedal Override
    10   elseif pedal override
```

```
*************************
*                       *
* END OF DESIGN DOCUMENT *
*                       *
*************************
```

```
STATISTICS
----------

Maximum complexity measure (CX) is 7.
1 flow segment had a complexity greater than 6.

14 flow segments.

2047 lines in definition file(s).
302 lines in source file(s).

618 dictionary entries allocated.
2204 string segments allocated;  2083 in use.

65024 bytes of dynamic memory allocated.
```

D.2 Source Listing

The input lines which resulted in the design document of the preceding section are:

```
%title An Automobile Cruise Control Example
%sindex
%dindex
%rindex
%stree
%complexity
%cic
%kwv
#{kw;spawn}
%g Requirements
%t Statement of Problem
Design a combined automobile cruise control and trip computer
with the following characteristics:

1.      Display Fuel Consumption Rate

        1.1     Each time fuel is added, the driver enters the
                amount added and the system computes and
                displays the consumption.

2.      Display Average Speed

        2.1     Each time the driver requests an average speed,
                the system displays the average speed since the
                "trip start".

        2.2     The driver may set "trip start".

3.      Display Maintenance Messages

        3.1     Oil and oil filter change every  5000 miles.

        3.2     Air filter change         every 10000 miles.

        3.3     Major service             every 15000 miles.

        3.4     Display intermittent message 250 miles before
                service required.

        3.5     Display continuous message 50 miles before
                service required.

        3.6     The driver informs the system when service
                is completed.

4.      Control Cruising Speed

        4.1     The system can control the throttle to maintain
                cruising speed.  For simplicity, it is assumed
                that the brakes will never be needed for speed
                control.

        4.2     The cruising speed is the speed the car is
                traveling when the cruise control is activated.

        4.3     Activation and Deactivation

                4.3.1   To activate the cruise control, the
                        engine must be running and the car must
                        be traveling 30 miles per hour or
```

```
                        faster.

                4.3.2   The cruise control can be deactivated
                        at any time.

                4.3.3   The cruise control is always deacti-
                        vated when the engine is turned off.

        4.4     Suspension and Resumption

                4.4.1   Operation of the cruise control is
                        temporarily suspended whenever the
                        driver uses the accelerator or brake.

                4.4.2   If the cruise control is suspended but
                        has not been deactivated, the driver
                        can tell the system to resume operation
                        and it will return the car to the
                        cruising speed.

        4.5     The driver can increase cruising speed by
                telling the system to increase speed and then
                telling it to stop increasing.

5.      Calibration

        5.1     The system must maintain accurate mileage.
                This is done by periodic calibration.

        5.2     To calibrate the system, the car is driven over
                a measured mile and the driver informs the
                system at the beginning and the end.

        5.3     The cruise control must be off during
                calibration.
%g Cruise Control and Monitoring
%s Main Program Task
%req 1;2;3;4;5

        initialize asynchronous determine speed task
        set current_state to inactive
        set within_mile to off
        do forever
                monitor automobile
                control speed
                measure mile
        enddo


%s Initialize Asynchronous Determine Speed Task
%req 1;2;3;4;5

        set last_time to clock_time
        set pulse_counts_per_mile to a default value
        spawn determine speed task

%s Determine Speed Task
%req 1;2;3;4;5

        do for each shaft_rotation
                set current time to clock_time
                set time per rotation to (current time - last_time)
                set last_time to current time
                set distance per rotation to (1/pulse_counts_per_mile)
                set current_speed to (distance per rotation/time per rotation)
                add distance per rotation to cummulative_distance
```

```
                if cummulative_distance is greater than one mile
                        increment current_mileage
                        subtract one mile from cummulative_distance
                endif
        enddo


%s Measure Mile
%req 5

        if current_state is not inactive or engine_running is false
                set within_mile to ignore
        elseif within_mile is off and measure_signal is on
                set within_mile to on
                reset pulse_counter
        elseif within_mile is on and measure_signal is on
                if there has been a shaft_rotation
                        increment pulse_counter
                endif
        elseif within_mile is on and measure_signal is off
                set pulse_counts_per_mile to pulse_counter
                set within_mile to off
        elseif measure_signal is off
                set within_mile to off
        endif


%g Control Speed
%s Control Speed
%req 4

        if there is a driver_request
                get new current state
        endif
        if current_state is inactive or engine_running is false
                set desired_speed to zero
                set current_state to inactive
        elseif pedal override
                set current_state to suspended
        elseif brake_engaged is true
                set current_state to suspended
        elseif current_state is cruising
                maintain speed
        elseif current_state is accelerating
                if current_speed is close to or above desired_speed
                        set current_state to cruising
                else
                        increase speed
                endif
        endif


%s Get New Current State
%req 4.3;4.4;4.5

        do case of driver_request
    increase_speed:
                set desired_speed to infinity
                set current_state to accelerating
    resume_speed:
                if desired_speed is not zero or infinity
                        set current_state to accelerating
                endif
    maintain_speed:
                if current_speed is above 30 mph
                        set desired_speed to current_speed
```

```
                        set current_state to cruising
                else
                        set current_state to inactive
                endif
    inactivate_cruise:
                set current_state to inactive
        enddo


%s Maintain Speed
%req 4.1

        if current_speed is much above desired_speed
                ..deceleration is needed
                set throttle_position to closed
        else
                set correction to current_speed - desired_speed
                ..normal throttle position is a function of speed
                set throttle_position the normal position plus
                        some function of correction
        endif


%s Increase Speed
%req 4.1

        ..normal throttle position is a function of speed
        ..acceleration is also a function of speed
        set throttle_position to normal position plus acceleration


%s Pedal Override
%req 4.4.1

        ..pedal_deflection is an input
        ..throttle_position is an output
        if pedal_deflection is greater than throttle_position
                ..the driver is accelerating
                return true
        else
                return false
        endif



%g Monitor Automobile
%s Monitor Automobile
%req 1;2;3

        compute trip average
        lookup maintenance schedule
        compute miles per gallon


%s Compute Trip Average
%req 2

        if activate_trip_average
                set trip_mileage to current_mileage
                set trip_clock to clock_time
        elseif get_trip_average
                display number ((trip_mileage - /
                        current_mileage) / (trip_clock - clock_time))
        endif
```

```
%s Compute Miles Per Gallon
%req 1

        if fuel_added
                display number ((last_fuel - current_mileage) / fuel_added)
                set last_fuel to current_mileage
        endif


%s Lookup Maintenance Schedule
%req 3

        if there is a maintenance_reset_command
                determine maintenance_type from maintenance_reset_command
                set last_reset for maintenance_type to current_mileage
        endif
        do for each maintenance_type ..oil_filter, air_filter, major_service
                set miles_since_maintenance to (current_mileage /
                        - last_reset for maintenance_type)
                set difference to (maintenance_interval for maintenance_type /
                        - miles_since_maintenance)
                if difference is greater than 250
                        set maintenance_message of maintenance_type to off
                elseif difference is greater than 50
                        set maintenance_message of maintenance_type to /
                                intermittent
                else
                        set maintenance_message of maintenance_type to on
                endif
        enddo

%d Maintenance Data
%req 3

maintenance_type is oil_filter, air_filter, major_service
..the following items are arrays of maintenance_type
maintenance_message
maintenance_interval is 3000, 10000, 15000
last_reset

%s Display Number (number)
%req 1.1;2.1

        write the number in decimal to the display
```

Index