Laser-Scan Ltd.

POLYGONS

Reference Manual

Science Park, Milton Road, Cambridge, England CB4 4FY tel: (0223) 420414

Document "POLYGONS Reference"		Category "REFERENCE"
Document Issue 2.5	Sunil Gupta	20-May-1992
Document Issue 2.4	Jon Barber	31-Oct-1991
Document Issue 2.3	Simon Hancock	25-Mar-1991
Document Issue 2.2	Dave Catlow	26-Oct-1988
Document Issue 2.1	Adrian Cuthbert	19-Sep-1988
Document Issue 2.0	Tim Hartnall,Martin Reid	15-Oct-1987
Document Issue 1.0	Tim Hartnall	19-Jun-1987

CONTENTS

POLYGONS reference documentation change record . . . i

PREFACE 1
  Intended audience 1
  Structure of this document 1
  Associated documents 1
  Conventions used in this document 2
  Command line data types 3

CHAPTER 1 INTRODUCTION

INTRODUCTION 1-1
POLYGONS - FEATURES 1-1
POLYGONS and Existing LSL Utilities. 1-1
POLYGONS and IFF 1-1
POLYGONS and DCL symbol $STATUS 1-2
POLYGONS and three dimensional strings. 1-2
Getting started with POLYGONS 1-2
How to specify POLYGONS command qualifier arguments 1-3
  Integer value ranges 1-3
  Floating point value ranges 1-4

CHAPTER 2 MODULE IPOLYGON

FUNCTION 2-1
FORMAT 2-2
PROMPT 2-3
PARAMETER 2-3
COMMAND QUALIFIERS 2-3
  Summary of IFF Default Parameters 2-27
RESTRICTIONS 2-28
  Summary of Parameter RESTRICTIONS 2-28
  Command Line Decoding RESTRICTIONS 2-29
DESCRIPTION 2-30
  Introduction 2-30
  IPOLYGON treatment of Polygons 2-30
  IPOLYGON treatment of the Bounding Polygon. . 2-44
  Labels and Identifiers 2-44
  Derivation of Polygon Labels 2-45
    Seed Point Identification 2-46
    Label Extraction From Seed Points 2-46
    Left/Right Coding 2-48
    Rules For Left/Right Coded Input Segments . 2-48
    Propagation Of Labels 2-49
  IPOLYGON checks on input IFF data 2-51
  IPOLYGON Production Flowline 2-52
  Program Operation 2-54
  IPOLYGON Output Options 2-57
    IPOLYGON/ASCII Output 2-58
    IPOLYGON/PIP Output 2-60
    IPOLYGON/POLYGONS Output 2-61

IPOLYGON/SEGMENTS Output 2-62
Using IPOLYGON with LITES2 2-63
Junction Check Errors 2-64
Seed Point Errors 2-64
Segment AC Check Error 2-65
Double Digitising Errors 2-65
Polygon Formation Errors 2-65
Seed Point Assignment Errors 2-66
Polygon Labelling Errors 2-66
Identifier Failure 2-67
EXAMPLES 2-68
MESSAGES (INFORMATIONAL) 2-81
MESSAGES (WARNING) 2-82
MESSAGES (ERROR) 2-84
MESSAGES (FATAL) 2-90
MESSAGES (OTHER) 2-91

CHAPTER 3 MODULE ISTSEL

FUNCTION 3-1
FORMAT 3-1
PROMPTS 3-1
PARAMETERS 3-1
COMMAND QUALIFIERS 3-2
DESCRIPTION 3-2
Program operation 3-2
The Code Definition File 3-3
EXAMPLES 3-5
MESSAGES (INFORMATIONAL) 3-7
MESSAGES (WARNING) 3-8
MESSAGES (ERROR) 3-9
MESSAGES (OTHER) 3-11

CHAPTER 4 MODULE POLMERGE

FUNCTION 4-1
FORMAT 4-1
PROMPTS 4-1
PARAMETERS 4-1
COMMAND QUALIFIERS 4-2
RESTRICTIONS 4-5
DESCRIPTION 4-6
Input and output file characteristics. 4-6
File Checking. 4-7
POLMERGE and IPOLYGON. 4-7
Polygon Merging and Elimination. 4-7
POLMERGE Tolerances. 4-8
Polygon Selection. 4-8
The Merging Process. 4-9
The Elimination Process. 4-10
The Lookup File. 4-10
EXAMPLES 4-12
MESSAGES (INFORMATIONAL) 4-19
MESSAGES (WARNING) 4-20

MESSAGES (ERROR) 4-22
MESSAGES (FATAL) 4-24
MESSAGES (OTHER) 4-25

CHAPTER 5 MODULE POLY_MESSAGES

POLYGONS library messages 5-1
MESSAGES (SUCCESS) 5-2
MESSAGES (INFORMATIONAL) 5-3
MESSAGES (WARNING) 5-4
MESSAGES (ERROR) 5-7
MESSAGES (FATAL) 5-11
MESSAGES (OTHER) 5-12

POLYGONS reference documentation change record

Version 0.0 Tim Hartnall 26-March-1987

Provisional issue of POLYGONS reference documentation.

Version 1.0 Tim Hartnall 30-March-1987

First customer issue of POLYGONS reference documentation.

Version 2.0 Tim Hartnall 06-October-1987

Major re-issue of POLYGONS reference documentation.

1. IPOLYGON chapter substantially changed to reflect the following functional changes:

   - IPOLYGON can now derive labelled closed polygons from input segments containing left/right AC (Ancillary Code) entries.

   - Polygons can now be formed from input segment data containing one arm junctions. The user may define the program action when a one arm junction is encountered. Options to issue a warning, omit, or use the segment leading to the one arm junction are provided. Used in conjunction with selective geometry processing, (within ILINK), this enables the user to form polygons from complex data sets without the need for pre-IPOLYGON data selection stages.

   - Data sets containing single point features (e.g unorientated symbols) and one arm junctions can now be processed without affecting the polygon formation process.

   - An option is now provided to generate junction structure in the left/right coded links IFF output file.

   - An option is now provided to enable the output of closed polygon features for which seed point, or labelling, assignment has failed.

   - The command interface has been completely rewritten to provide logical groupings of keyword arguments.

   In addition to reflecting these functional changes, the documentation now includes an example production flowline.

2. A new POLYGONS package library has been introduced. Messages have been rationalised for shared use with other modules within the POLYGONS package, and the library messages are now documented in a

separate chapter.

3. A whole new chapter has been added for the new POLMERGE utility.

4. The index has been greatly enlarged.

Version 2.1 Adrian Cuthbert 19-September-1988

Major re-issue of POLYGONS reference documentation.

1. IPOLYGON chapter substantially changed to reflect new functionality and rationalisation.

2. Expanded messages for the POLYGONS package library reflecting the increased functionality of the polygon formation algorithms.

Version 2.2 Dave Catlow 25-October-1988

Documentation on IPOLYGON and POLMERGE changed to reflect the redimensioning of both utilities to handle a larger number of segments and polygons.

Version 2.3 Simon Hancock 27-March-1990

1. IPOLYGON chapter changed to describe the function of ancillary code propagation.

Version 2.3 (Modified) Tim Hartnall 08-August-1990

1. IPOLYGON /SEED=AC default now correctly documented as type 82 AC.

Version 2.3 (Modified) Tim Hartnall 09-October-1990

1. IPOLYGON /SEED general description restored. This had been left out of version 2.3 documentation. Incorrect references to /PIP=LABEL_AC and /PIP=IDENT_AC changed to /PIP=CONTAIN_LABEL_AC and /PIP=CONTAIN_IDENT_AC.

Version 2.4 Jon Barber 31-October-1991

1. IPOLYGON /ABSOLUTE qualifier added.

2. New messages MDABSENT and MDDEFAULT in module POLY_MESSAGES.

Version 2.5 Sunil Gupta 20-May-1992

1. The internal limits on the number and size of polygons which can be handled by IPOLYGON and POLMERGE have been lifted. These can be modified by assigning values to the logical names LSL$POLYGONS_POLMAX and LSL$POLYGONS_AVERAGE_SIDES. The section on "Restrictions" in both the IPOLYGON and POLMERGE chapters has been modified to reflect this.

2. New POLYGON library messages: MEMORY, ALLOCD, SUGGEST, DEFPOL, DEFSID, POLVAL, SIDVAL in module POLY_MESSAGES.

Version 2.6 John Cadogan 12-March-1993

1. New message: PHANNOMOTH in module POLMERGE.

PREFACE

Intended audience

This manual is intended for users of the Laser-Scan POLYGONS package running under the VAX/VMS operating system.

Structure of this document

The POLYGONS reference manual contains the following sections:

MODULE

REPLACES - which older Laser-Scan programs it replaces.

FUNCTION - a synopsis of what the modules does

FORMAT - a summary of the module command format and command qualifiers. Default qualifier settings are indicated.

PROMPT - how it prompts the user.

PARAMETERS - description of expected command parameters.

COMMAND QUALIFIERS - description of all command qualifiers. Qualifiers are ordered alphabetically and default argument values are indicated.

RESTRICTIONS - utility restrictions and limits.

DESCRIPTION - the definitive description of the utility.

EXAMPLES - annotated examples of module useage.

MESSAGES - all classes of message are listed and described and suggested user action given. The messages are divided into sections according to message severity within which the messages are ordered alphabetically by message mnemonic.

Associated documents

The POLYGONS package is designed to be used in conjunction with the Laser-Scan STRUCTURE package component ILINK. See the "STRUCTURE Reference Manual" for details of ILINK.

Conventions used in this document

Convention	Meaning
<CR>	The user should press the carriage control key on the terminal
<CTRL/x>	The phrase <CTRL/x> indicates that the user must press the key labelled CTRL while simultaneously pressing another key, for example, <CTRL/Z>.
$ **IPOLYGON JIM<CR>**	Command examples show all user entered commands in **bold** type.
$ **IPOLYGON JIM<CR>** . . .	Vertical series of periods, or ellipsis, mean either that not all the data that IPOLYGON would display in response to the particular command is shown or that not all the data that the user would enter is shown.
file-spec...	Horizontal elipsis indicates that additional parameters, values or information can be entered.
[logical-name]	Square brackets indicate that the enclosed item is optional. (Square brackets are not, however, optional in the syntax of a directory name in a file-specification, or in the syntax of a substring specification in a VMS assisnment statement).
'integer'	An integer number is expected in the specified input or output field. (See "Command line data types" below).
'real'	A real number is expected in the specified input or output field. (See "Command line data types" below).

Convention	Meaning
FSN 'integer' ('integer')	FSN followed by two integer arguments indicates an IFF feature serial number. The integer number enclosed in round brackets is the feature internal sequence number.
00003DE7	A hexadecimal address of a location within an IFF file. IPOLYGON expresses all IFF addresses using hexadecimal radix. The address is always padded with leading zeros to a standard field width of 8 characters.

Command line data types

The utilities which comprise the POLYGONS package use the VMS Command Line Interpreter (CLI) to get and parse the program command line. IPOLYGON thus offers a VMS emulating user interface. Unfortunately the VMS Digital Command Language (DCL) does not support the real (or "floating point") data type. Many Laser-Scan IFF utilities require real value arguments for the specification of tolerances and distances etc. To meet this requirement, Laser-Scan have developed an enhanced CLI based command line decoding mechanism. This enables the interpretation of numbers as either "real" or "integer". Throughout this document the number types are differentiated by the words 'integer' for integer numbers and 'real' for real (or "floating point") numbers.

POLYGONS command line decoding operates in decimal radix.

CHAPTER 1

INTRODUCTION

INTRODUCTION

POLYGONS is the Laser-Scan IFF polygon creation and processing package. IFF stands for **I**nternal **F**eature **F**ormat and is the Laser-Scan vector file format generated by LASERAID and other Laser-Scan mapping systems and used as the data POLYGONS throughout the Laser-Scan LAMPS system. IFF files are binary and cannot be manipulated directly using a text editor. The POLYGONS package enables the user to perform a wide range of polygon creation and manipulation tasks related to the requirements of the automated mapping industry.

POLYGONS - FEATURES

The POLYGONS package consists of independent modules which together form a polygon creation and manipulation system within an automated mapping environment. The modules which form the POLYGONS package offer:

- common command syntax. Module command lines are decoded using the Command Line Interpreter as used by the VAX/VMS utilities.

- VMS format messages referenced using 32 bit condition code symbols.

- VMS DCL symbol $STATUS on image exit.

- comprehensive documentation in this reference manual using a style consistent with that used by Digital Equipment Corporation in their VMS utility manuals. The POLYGONS User Reference documentation includes an explanation of the messages output by the modules together with suggested user action.

POLYGONS and Existing LSL Utilities.

POLYGONS is designed to replace the existing Laser-Scan utilities POLCHK and CODSEG, as well as to create new facilities for the creation and manipulation of polygon data.

POLYGONS and IFF

Within the VAX/VMS system IFF files can be treated as any other file type for file management purposes. To enable the user to distinguish an IFF file from a file of another type IFF files have by default the file extension '.IFF'. To provide great flexibility in the production environment IFF files are referenced by all the POLYGONS modules using logical name LSL$IF:. (For an explanation of logical names see the VAX/VMS document set). Logical name LSL$IF: is assigned to a device and directory specification either using the VMS DEFINE command or the Laser-Scan SI utility, (see the "IFF User Guide" for details).

POLYGONS and DCL symbol $STATUS

Like VMS utilities, all POLYGONS modules generate VMS format messages and set VMS DCL symbol $STATUS on image exit. This is a valuable feature as a non-interactive process can test the success of a preceding POLYGONS module before proceeding. $STATUS will always be set to a VMS 32 bit condition code. Successful program execution will result in $STATUS being set to SS$_NORMAL. If an error occurred during POLYGONS processing, SS$_ABORT of varying severities, or a VMS System or CLI (Command Line Interpreter) condition code will be used. The user may simply test $STATUS for TRUE or FALSE within a DCL command procedure. If $STATUS is TRUE then processing was successful. If it is FALSE, an error occurred during processing. For a detailed description of the uses of $STATUS see the VAX VMS document set.

POLYGONS and three dimensional strings.

None of the POLYGONS modules handle 3 dimensional strings held in IFF ZS entries. Feature containing ZS entries will be copied unprocessed to the output file (if appropriate).

Getting started with POLYGONS

Once logged in the user must give two commands to initialise the POLYGONS package. Before the POLYGONS package can be used DCL symbols and logical names must be assigned to enable the user to invoke the modules. This is dome using a command procedure POLYGONSINI.COM which is supplied as part of the POLYGONS package. POLYGONSINI itself will be defined as a DCL symbol at your site and should be invoked thus: (see PREFACE for explanation of presentation conventions)

```
$ POLYGONSINI<CR>
```

The POLYGONSINI command invokes a command procedure which defines a DCL symbol (the module name) for each of the POLYGONS modules. After using POLYGONSINI the user need only type the symbol name to activate the module of his choice.

As an alternative to explicitly typing the POLYGONSINI command each time the user wishes to use the POLYGONS package, the POLYGONSINI command may be placed in the users login file, or in the site dependent default login file.

The second command which must be given before using the POLYGONS package is the SI command. The SI command assigns the logical name LSL$IF: (or IF: for short) to the device-directory specification which contains the IFF file(s) that are to be manipulated. For example:

```
$ SI DUA3:[BUREAU.TRIALS.DIGITISING]
```

This will assign logical name LSL$IF: to the device and directory specification DUA3:[BUREAU.TRIALS.DIGITISING]

For further details of the SI command see the IFF User Guide.

How to specify POLYGONS command qualifier arguments

POLYGONS utilities use the VMS Command Line Interpreter (CLI) to get and parse the program command line. POLYGONS utilities thus offer a VMS emulating user interface. As many POLYGONS utilities require floating point arguments to command, qualifiers Laser-Scan have developed an enhanced CLI based command line decoding mechanism. This enables the interpretation of numbers as either "real" or "integer".

The CLI allows the user to specify single and lists of integer qualifier arguments. If a list of arguments is specified, each argument must be separated by a comma and the whole list enclosed within parentheses, for example:

Single argument:

```
$ EXAMPLE/QUALIFIER=7<CR>
```

Where "EXAMPLE" is the command and /QUALIFIER is a qualifier to that command. There is one qualifier argument - 7

Argument list:

```
$ EXAMPLE/QUALIFIER=(2,5,8,9,10,11,12,13,14)<CR>
```

Where "EXAMPLE" is the command and /QUALIFIER is a qualifier to that command. There are 9 qualifier arguments within the argument list.

Integer value ranges

While developing the floating point command line data type (see Preface) it was recognised that there is a need for numeric range decoding within a VMS emulating command line. Argument ranges are specified with the syntax:

n:m

Where n is the lower limit of the range and m is the upper limit of the range (inclusive).

Such ranges are expanded in full. A maximum of 1024 arguments can be specified to any one command qualifier.

If we take our example argument list used above, i.e:

```
$ EXAMPLE/QUALIFIER=(2,5,8,9,10,11,12,13,14)<CR>
```

and now use the Laser-Scan argument range decoding mechanism:

```
$ EXAMPLE/QUALIFIER=(2,5,8:14)<CR>
```

we see that a more compact command line results but yields the same arguments. This is clearly an advantage in an IFF map processing environment where a single file could contain hundreds of attributes which the user may wish to reference via command line arguments.

Other examples are:

```
$ EXAMPLE/QUALIFIER=2:9<CR>
```

This yields 8 integer arguments: 2, 3, 4, 5, 6, 7, 8, and 9

```
$ EXAMPLE/QUALIFIER=:8<CR>
```

This yields 9 integer arguments: 0, 1, 2, 3, 4, 5, 6, 7, and 8

If when ranges are decoded, a qualifier has more than 1024 arguments the Laser-Scan LSLLIB library issues the error message:

```
%LSLLIB-E-RESPARSOVF, result of parse overflowed buffer
```

and program execution is terminated.

Floating point value ranges

Floating point value ranges are decoded in a different manner to integer value ranges. Instead of expanding the range to yield all its component integer values the command decoder merely leaves the range as a lower limit and an upper limit. Processing then takes account of any possible value lying between these limits (inclusive).

For example:

```
$ EXAMPLE/HEIGHT=(23.5:110.2)<CR>
```

Select all features having a height which lies within the range 23.5 to 110.2 inclusive.

CHAPTER 2

MODULE IPOLYGON

MODULE **IPOLYGON**

REPLACES CODSEG and POLCHK

FUNCTION

IPOLYGON is the Laser-Scan automatic **I**FF **POLYGON** creation and labelling utility. It forms the core of the Laser-Scan POLYGONS Package.

IPOLYGON is designed to be run in batch mode and all options may be specified on the command line. No user interaction is required during processing.

IPOLYGON carries out polygon formation and the determination of first order nesting based on the input geometry.

IPOLYGON offers two methods for labelling (and checking the consistency) of polygons:

- o Seed point assignment: the polygon label is extracted from the seed point data.
- o Left/Right coding: each segment has a left and right AC (Ancillary Code). The polygon label is extracted from the text part of the AC.

In addition IPOLYGON provides a unique (internally generated) identifier for each polygon.

IPOLYGON offers four methods of polygon output:

- o An IFF file containing complete closed polygons as single features.
- o An IFF file containing labelled segments with left/right codes.
- o An IFF file containing a single point feature lying in each polygon.
- o An ASCII file containing lists of those segments that make up polygons.

All output options allow the label and/or identifier for each polygon to be output to the IFF features through the use of user-specified AC (Ancillary Code) entries.

FORMAT

```
$ IPOLYGON file-spec
```

Command qualifiers

```
/[NO]ABSOLUTE

/ASCII=(      [[NO]IDENT],
              [[NO]LABEL])

/[NO]LIST     [='file-spec']

/[NO]LITES2   [='file-spec']

/[NO]LOG

/LRCODE=(     [LEFT_AC:'integer'],
              [RIGHT_AC:'integer'])

/ONEARM=(     [CONTAIN],
              [DELETE],
              [USE],
              [[NO]WARN])

/OPTIONS=(    [[NO]AREA],
              [[ANTI]CLOCKWISE],
              [IDENT_TEXT:'text-string'],
              [[NO]NEST],
              [UNDEFINED:'keyword'])

/PIP=(        [FC:'integer'],
              [[NO]IDENT],
              [CONTAIN_IDENT_AC:'integer'],
              [ITERATE:'integer']
              [[NO]LABEL],
              [CONTAIN_LABEL_AC:'integer'],
              [LAYER:'integer'],
              [OUTPUT:'file-spec'])

/[NO]PME

/POLYGONS=(   [FC:'integer'],
              [[NO]IDENT],
              [IDENT_AC:'integer'],
              [[NO]LABEL],
              [LABEL_AC:'integer'],
              [LAYER:'integer'],
              [OUTPUT:'file-spec'])

/[NO]PRINTER
```

```
          /PROPAGATE=(  [FULL],
                        [PARTIAL],
                        [[NO]CONCATENATE],
                        [LEFT_AC:'integer'],
                        [RIGHT_AC:'integer'],
                        [COVERAGE_FILE:'file-spec'])

          /SEED=(        [AC:'integer'],
                        [FC:'integer'[,....]],
                        [FILE:'file-spec'],
                        [LAYER:'integer'[,...]],
                        [PAIR:'file-spec'],
                        [SURROUND:'text-string'],
                        [USE:'keyword'])

          /SEGMENTS=(   [CONTAIN_IDENT_AC:'integer'],
                        [CONTAIN_LABEL_AC:'integer'],
                        [[NO]IDENT],
                        [[NO]JUNCTIONS],
                        [[NO]LABEL],
                        [LEFT_IDENT_AC:'integer'],
                        [LEFT_LABEL_AC:'integer'],
                        [OUTPUT:'file-spec'],
                        [RIGHT_IDENT_AC:'integer'],
                        [RIGHT_LABEL_AC:'integer'],
                        [SELECT_FC:'range;range...'])
```

PROMPT

```
          _Segment-IFF-file:       IFF-file-spec
```

PARAMETER

IFF-file-spec

- specifies the junction structured IFF file from which the polygons are to be formed. Any part of the file specification which is not supplied will be taken from the default specification 'LSL$IF:IFF.IFJ'.

COMMAND QUALIFIERS

/ABSOLUTE
/NOABSOLUTE **(default)**

- this will result in the output of absolute coordinate values to any LITES2 guidance file requested with the /LITES2 qualifier, or to any other messages output. The default action is to output the usual LITES2 coordinate values.

/ASCII=([[NO]IDENT],
        [[NO]LABEL])
/NOASCII **(default)**

- causes a list of all the segments which form each polygon to be written at the end of the /LIST text file. If /ASCII is specified without /LIST, the presence of /LIST on the command line is assumed and a text file created with the specification SYS$DISK:[]IPOLYGON.LIS;0. IPOLYGON will then behave as if an explicit /LIST qualifier was present on the command line. If /ASCII is specified without any of the keyword arguments, the following defaults are assumed:

  /ASCII=(NOIDENT,
  NOLABEL)

  Segment listings will be output for each of the polygons. Whether polygon boundaries include first-order nesting is governed by the /OPTIONS=[NO]NEST qualifier. The segment lists will be ordered clockwise or anticlockwise depending on the /OPTIONS=[ANTI]CLOCKWISE qualifier. The generation of area statistics is governed by the /OPTIONS=AREA qualifier. Area calculations will reflect the first-order nesting convention used to output the segment lists.

/ASCII=IDENT
/ASCII=NOIDENT **(default)**

- indicates that the internally generated polygon identifiers are to be written to the ASCII output file.

  By default identifiers are not written to the ASCII output file.

/ASCII=LABEL
/ASCII=NOLABEL **(default)**

- indicates that labels obtained from either seed points or left/right codes are to be written to the ASCII output file. The /ASCII=LABEL combination cannot be used unless the /SEED or /LRCODE qualifiers are present.

  Polygons for which labelling has failed will be output depending on the state of the /OPTIONS=UNDEFINED:'keyword' qualifier.

  By default labels are not written to the ASCII output file.

/LIST[='file-spec']
/NOLIST **(default)**

- by default, message and diagnostic output will be to SYS$OUTPUT. This option allows the user to redirect output to the specified text file. If the optional file-spec argument is omitted IPOLYGON directs output to a file named SYS$DISK:[]IPOLYGON.LIS. If the user supplies only a partial file-spec the missing file specification components are taken from the default specification SYS$DISK:[]IPOLYGON.LIS;0.

  If the /ASCII qualifier is present then a list of all the segments which form each polygon boundary is written at the end of the /LIST text file. If /ASCII is specified without /LIST, the presence of /LIST on the command line is assumed and a text file created with the specification SYS$DISK:[]IPOLYGON.LIS;0. IPOLYGON will then behave as if an explicit /LIST qualifier was present on the command line.

/LITES2[='file-spec']
/NOLITES2 **(default)**

- creates a LITES2 command file to take the user to potential errors. Messages generated to report a potential error are incorporated in the file. By default the LITES2 command file specification is parsed against that of the input segments IFF file but with the substitution of logical name LSL$LITES2CMD: and the extension '.LCM'. Thus if the input IFF file is called LSL$IF:TST.IFJ then the default LITES2 command file is LSL$LITES2CMD:TST.LCM.

/LOG
/NOLOG **(default)**

- this will result in supplementary messages being sent to SYS$OUTPUT. Supplementary messages are generated when a file is successfully opened, and messages indicating the progress of IPOLYGON processing are also output.

/LRCODE=([LEFT_AC:'integer'],
         [RIGHT_AC:'integer'])

- indicates that labelling information is to be extracted from ACs (Ancillary Codes) on the input segments. Each input segment must have ACs indicating which polygons lie to the left and right of it. The polygon label is taken from the string part of the specified ACs.

  /LRCODE cannot be used with the /SEED qualifier.

/LRCODE=LEFT_AC:'integer'
/LRCODE=LEFT_AC:4 **(default)**

- the label for the polygon lying to the left of a segment is obtained by taking the string part of the AC entry with the specified AC type. By default this type is 4.

/LRCODE=RIGHT_AC:'integer'
/LRCODE=RIGHT_AC:5 **(default)**

- the label for the polygon lying to the right of a segment is obtained by taking the string part of the AC entry with the specified AC type. By default this type is 5.

/ONEARM=(CONTAIN,DELETE,USE,[NO]WARN)

the purpose of the /ONEARM qualifier is to enable the user to choose how IPOLYGON reacts when it encounters an IFF junction which has only one arm. Such junctions can cause great problems to polygon formation algorithms which are designed always to 'turn right' or 'turn left' relative to the 'current' junction arm to form clockwise and anticlockwise polygons respectively.

IPOLYGON is designed to cope with any complexity of one arm junction branching and can, for example, form polygons from a mixture of dendritic drainage features and segments which do form closed polygons.

In addition to issuing simple warning messages when a one arm junction is encountered, IPOLYGON can deal with one arm junctions in one of three ways:

/ONEARM=CONTAIN

- when /ONEARM=CONTAIN is specified and a one arm junction is detected during polygon formation, IPOLYGON will go back to the junction that it left prior to arriving at the one arm junction and try a different arm. This process is repeated, possibly undoing much of the already formed polygon, until a route ahead is found, and the polygon closes. IPOLYGON will pass segments joined to one-arm junctions onto the output file. If polygon labels or identifiers are being used then an AC will indicate the segment is contained **within** the appropriate polygon.

  /ONEARM=CONTAIN is only valid with the /SEGMENTS output qualifier.

  The combination /ONEARM=CONTAIN cannot be used if /ONEARM=DELETE or /ONEARM=USE is present.

/ONEARM=DELETE **(default)**

- when /ONEARM=DELETE is specified and a one-arm junction is detected during polygon formation, IPOLYGON will go back to the junction that it left prior to arriving at the one-arm junction and try a different arm. This process is repeated, possibly undoing much of the already formed polygon, until a route ahead is found and the polygon closes. IPOLYGON will omit segments joined to one-arm junctions from the output file.

  If /ONEARM=DELETE is used with the /SEGMENTS=JUNCTIONS combination, the output junction structured file has the same junction structure as the input file. Thus segments that would normally be omitted from output are retained. These can be identified if polygon labels or identifiers are being output. Such segments will have the text "Ignored Segment".

  The combination /ONEARM=DELETE cannot be used if /ONEARM=CONTAIN or /ONEARM=USE is present.

/ONEARM=USE

- when /ONEARM=USE is specified and a one-arm junction is detected during polygon formation, IPOLYGON will track back down the other side of the free arm. This tracing round linework internal to a polygon can form a 'tree' which may (or may not) be connected to the outer boundary of the polygon. IPOLYGON will pass those segments that make up these trees onto the output file. If polygon labels or identifiers are being used then the left and right ACs will reference the same polygon.

  The 'tree' will not affect polygon area calculations (/OPTIONS=AREA), nor will it affect FPP (Fast Plotting Program) area fill.

  The combination /ONEARM=USE cannot be used if /ONEARM=CONTAIN or /ONEARM=DELETE is present.

/ONEARM=WARN
/ONEARM=NOWARN **(default)**

- warns the user of the presence of all one-arm junctions. If the /LITES2 qualifier is specified, commands are sent to the LITES2 command file to enable the user to quickly investigate the cause of the one arm junction using LITES2.

```
/OPTIONS=([[NO]AREA],
          [[ANTI]CLOCKWISE],
          [IDENT_TEXT:'text-string'],
          [[NO]NEST],
          [UNDEFINED:'keyword'])
```

- enables the user to select options that determine the method of polygon formation and the generation of unique identifiers. If /OPTIONS is not specified, or is specified without any of the keyword arguments the following defaults are assumed:

  **/OPTIONS=(NOAREA,
ANTICLOCKWISE,
IDENT_TEXT:"Polygon ",
NEST,
UNDEFINED:KEEP)**

/OPTIONS=AREA
/OPTIONS=NOAREA **(default)**

- provides output of polygon area statistics. The /OPTIONS=AREA keyword is available for use only with the /ASCII, /PIP and /POLYGONS qualifiers. Output of ASCII segment listings or IFF closed polygon boundary features will perform area calculations according to the setting of the /OPTIONS=NEST qualifier. This will only effect the calculated areas of 'doughnut' shaped polygons.

/OPTIONS=CLOCKWISE
/OPTIONS=ANTICLOCKWISE **(default)**

- specifies that the segment or coordinate listings of any closed polygon features produced as output, using the /ASCII or /POLYGONS options, will have their segments/coordinates ordered in a consistent clockwise direction.

  The default coordinate ordering is anticlockwise. (See Description Section for details of IPOLYGON treatment of nested polygons).

/OPTIONS=IDENT_TEXT:'text-string'
/OPTIONS=IDENT_TEXT:"Polygon " **(default)**

- specifies how the internally generated identifiers are to be constructed from an internally generated serial number. By default the identifier will be a string of the form 'Polygon xx' where xx represents the serial number. The default text part of the identifier can be overridden by supplying a new text part with the /OPTIONS=IDENT_TEXT:'text-string' combination.

/OPTIONS=NONEST
/OPTIONS=NEST **(default)**

- indicates IPOLYGON should not determine first-order polygon nesting. Be warned that polygon areas calculated in conjunction with the /OPTIONS=NONEST qualifier will be incorrect in the case of 'doughnut' shaped polygons. In such cases the polygon area will represent that of the whole polygon, not the polygon minus the nested polygon area.

The /OPTIONS=NONEST combination is only applicable when the /ASCII or /POLYGONS qualifiers are present, that is, those output options that output polygon boundaries as a single entity.

By default IPOLYGON will determine whether polygons are nested.

/OPTIONS=UNDEFINED:DELETE
/OPTIONS=UNDEFINED:KEEP **(default)**

- specifies the IPOLYGON action when polygon labelling fails. There is only a need for such action if the output mechanism treats polygons in their entirety, namely the /ASCII, /PIP and /POLYGONS options. Similarly labelling can only fail if some attempt is made to label the polygons, that is either the /LRCODE or /SEED qualifier is present.

  The default action is to output all polygons regardless of labelling failure. Polygons which suffered labelling failure are given the label "Undefined Polygon".

  If /OPTIONS=(UNDEFINED:DELETE) is specified, any polygons which suffer labelling failure are omitted from the relevant output files.

```
/PIP=([FC:'integer'],
      [[NO]IDENT],
      [CONTAIN_IDENT_AC:'integer'],
      [ITERATE:'integer'],
      [[NO]LABEL],
      [CONTAIN_LABEL_AC:'integer'],
      [LAYER:'integer'],
      [OUTPUT:'file-spec'])
```

/NOPIP **(default)**

- indicates that one point feature per polygon is to be output to an IFF file. The coordinates of the point features are calculated to lie within the polygon. If /PIP is specified without any of the keyword arguments the following defaults are assumed:

  ```
  /PIP=(FC:1,
        NOIDENT,
        ITERATE:1,
        NOLABEL,
        LAYER:1)
  ```

  The point-in-polygon features will be output with FC (Feature Code) 1 to layer 1. By default polygon labels and identifiers will not be copied to the output features.

/PIP=OUTPUT:'file-spec'

- specifies the IFF file that the point-in-polygon features are output to. In the absence of the /PIP=OUTPUT qualifier the output file specification is taken from that of the segment input file but with the file extension '.IFF'. This default mechanism may be overridden by use of the /PIP=OUTPUT:'file-spec' combination. Any file-spec argument given will be used as the output file specification. Any parts found to be missing from the file-spec OUTPUT will be taken from the input file-spec with a '.IFF' extension.

/PIP=LAYER:'integer'
/PIP=LAYER:1 **(default)**

- enables the user to specify the layer to be used for IFF point-in-polygon output. By default the point features will be placed in layer 1 and will be given a feature code of 1.

/PIP=FC:'integer'
/PIP=FC:1 **(default)**

- enables the user to specify the FC (Feature Code) to be used for IFF point-in-polygon output. By default the point features will be placed in layer 1 and will be given a feature code of 1.

/PIP=IDENT
/PIP=NOIDENT **(default)**

- indicates that the internally generated polygon identifiers are to be copied to the point-in-polygon features. The identifier is transferred as the string part of the AC type specified by the /PIP=CONTAIN_IDENT_AC:'integer' combination.

  The numeric field of the AC will contain the polygon area if the /OPTIONS=AREA qualifier is present.

  By default identifiers are not added to the output point-in-polygon features.

/PIP=CONTAIN_IDENT_AC:'integer'
/PIP=CONTAIN_IDENT_AC:82 **(default)**

- specifies which AC type is to be used to copy the polygon identifiers to the point-in-polygon features. The /PIP=CONTAIN_IDENT_AC:'integer' combination cannot be used unless the /PIP=IDENT combination is present.

  By default labels are copied to point-in-polygon features using AC type 82.

/PIP=ITERATE:'integer'
/PIP=ITERATE:1 **(default)**

- indicates that a maximum of 'integer' iterations should be used to place the point-in-polygon. The number of iterations should be in the range 1 to 100. All values of the iterate parameter *will* produce points that lie in the required polygons. A higher value should produce a better positioned point, albeit at the expense of increased CPU time.

/PIP=LABEL
/PIP=NOLABEL **(default)**

- indicates that labels obtained from either seed points or left/right codes are to be transferred to the point-in-polygon features. The label is transferred as the string part of the AC type specified by the /PIP=CONTAIN_LABEL_AC:'integer' combination. The /PIP=LABEL combination cannot be used unless the /SEED or /LRCODE qualifiers are present.

  The numeric field of the AC will contain the polygon area if the /OPTIONS=AREA qualifier is present.

  Polygons for which labelling has failed will be output depending on the state of the /OPTIONS=UNDEFINED:'keyword' qualifier.

  By default labels are not added to the output point-in-polygon features.

/PIP=CONTAIN_LABEL_AC:'integer'
/PIP=CONTAIN_LABEL_AC:82 **(default)**

- specifies which AC type is to be used to pass labels to the point-in-polygon features. The /PIP=CONTAIN_LABEL_AC:'integer' combination cannot be used unless the /PIP=LABEL combination is present.

  By default labels are copied to point-in-polygon features using AC type 82.

/PME
/NOPME **(default)**

- enables the PME performance monitor. The /PME qualifier is reserved for Laser-Scan use. PME is a code optimisation tool and should be invoked by LSL software personnel only.

```
/POLYGONS=([FC:'integer'],
           [[NO]IDENT],
           [IDENT_AC:'integer'],
           [[NO]LABEL],
           [LABEL_AC:'integer'],
           [LAYER:'integer'],
           [OUTPUT:'file-spec'])
```

/NOPOLYGONS **(default)**

- indicates that closed polygon boundaries are to be output to an IFF file as single features. If /POLYGONS is specified without any of the keyword arguments the following defaults are assumed:

  ```
  /POLYGONS=(FC:1,
             NOIDENT,
             NOLABEL,
             LAYER:1)
  ```

  The closed polygon boundary features will be output with FC (Feature Code) 1 to layer 1. By default polygon labels and identifiers will not be copied to the output features. Whether polygon boundaries include first-order nesting is governed by the /OPTIONS=[NO]NEST qualifier. The polygon boundaries will be ordered clockwise or anticlockwise depending on the /OPTIONS=[ANTI]CLOCKWISE qualifier.

  If the /OPTIONS=NONEST qualifier is not present then IPOLYGON will determine first-order nesting. First level nested polygons will be included in the closed polygon feature with an invisible (pen-up) move between the enclosing polygon and the nested polygon coordinates. This facilitates area shading using standard Laser-Scan plotting software.

/POLYGONS=OUTPUT:'file-spec'

- specifies the IFF file that the polygon boundary features are output to. In the absence of the /POLYGONS=OUTPUT qualifier the output file specification is taken from that of the segment input file but with the file extension '.IFF'. This default mechanism may be overriden by use of the /POLYGONS=OUTPUT:'file-spec' combination. Any file-spec argument given will be used as the output file specification. Any parts found to be missing from the file-spec OUTPUT will be taken from the input file-spec with a '.IFF' extension.

/POLYGONS=FC:'integer'
/POLYGONS=FC:1 **(default)**

- enables the user to specify the FC (Feature Code) to be used for IFF closed polygon feature output. By default the closed polygons will be placed in layer 1 and will be given a feature code of 1.

/POLYGONS=LAYER:'integer'
/POLYGONS=LAYER:1 **(default)**

- enables the user to specify the layer to be used for IFF closed polygon feature output. By default the closed polygons will be placed in layer 1 and will be given a feature code of 1.

/POLYGONS=IDENT
/POLYGONS=NOIDENT **(default)**

- indicates that the internally generated polygon identifiers are to be copied to the polygon boundary features. The identifier is transferred as the string part of the AC type specified by the /POLYGONS= IDENT_AC:'integer' combination.

  The numeric field of the AC will contain the polygon area if the /OPTIONS=AREA qualifier is present.

  By default identifiers are not added to the output polygon boundary features.

/POLYGONS=IDENT_AC:'integer'
/POLYGONS=IDENT_AC:82 **(default)**

- specifies which AC type is to be used to copy the polygon identifiers to the polygon boundary features. The /POLYGONS=IDENT_AC:'integer' combination cannot be used unless the /POLYGONS=IDENT combination is present.

  By default labels are copied to polygon boundary features using AC type 82.

/POLYGONS=LABEL
/POLYGONS=NOLABEL **(default)**

- indicates that labels obtained from either seed points or left/right codes are to be transferred to the polygon boundary features. The label is transferred as the string part of the AC type specified by the /POLYGONS=LABEL_AC:'integer' combination. The /POLYGONS=LABEL combination cannot be used unless the /SEED or /LRCODE qualifiers are present.

  If the /SEED qualifier is present then any AC (Ancillary Code) entries within the seed point feature will be copied to the closed polygon feature provided it is not to be used by IPOLYGON. This provides the user with flexible multiple attribute coding for the closed polygon. Each AC contains a numeric code field and up to 255 characters of text.

  The numeric field of the AC will contain the polygon area if the /OPTIONS=AREA qualifier is present.

  Polygons for which labelling has failed will be output depending on the state of the /OPTIONS=UNDEFINED:'keyword' qualifier.

By default labels are not added to the output polygon boundary features.

/POLYGONS=LABEL_AC:'integer'
/POLYGONS=LABEL_AC:82 **(default)**

- specifies which AC type is to be used to pass labels to the polygon boundary features. The /POLYGONS=LABEL_AC:'integer' combination cannot be used unless the /POLYGONS=LABEL combination is present.

  By default labels are copied to polygon boundary features using AC type 82.

/PRINTER
/NOPRINTER **(default)**

- queues the IPOLYGON text output for printing on SYS$PRINT: under the name given by the /LIST qualifier. If you specify /PRINTER without the /LIST qualifier, the output is directed to a file named SYS$DISK:[]IPOLYGON.LIS which is queued automatically for printing and, after printing, deleted.

```
/PROPAGATE=([FULL],
            [PARTIAL],
            [[NO]CONCATENATE],
            [LEFT_AC:'integer'],
            [RIGHT_AC:'integer'],
            [COVERAGE_FILE:'file-spec'])
```
/NOPROPAGATE **(default)**

- indicates that labelling information held in the specified ACs is to be extended after polygon formation. If some but not all of the component features of a resultant polygon P have a consistent interior identifier according to the left/right coding of a particular coverage, then this coding will be applied to all component features of the polygon. Furthermore, if full propagation is specified then this coding will be extended to any adjacent polygon separated from polygon P by a feature having no initial left or right coding with respect to the coverage in question.

/PROPAGATE=FULL
/PROPAGATE=PARTIAL **(default)**

- determines the extent of attribute code extension, as described above.

/PROPAGATE=CONCATENATE
/PROPAGATE=NOCONCATENATE **(default)**

- requires output features to contain a single attribute code, containing concatenated output texts. The AC type will be as specified for the first coverage. If this qualifier is negated or absent, each output feature will receive one attribute code from each input coverage.

/PROPAGATE=LEFT_AC:'integer'
/PROPAGATE=RIGHT_AC:'integer'

- specifies the AC types to be used in defining a coverage, in the case where a single coverage is to be propagated and no prefix is to be employed in its definition.

/PROPAGATE=COVERAGE_FILE:'file-spec'

- nominates a file defining the coverages involved in the propagation operation. A record in this file contains:

  1. coverage number, in the range 1 to 4

  2. left AC type

  3. right AC type

  4. input text prefix, delimited by quotes

  5. output text prefix, delimited by quotes

6. output left AC type

7. output right AC type

8. output AC type for contained segments

9. output AC type for polygon labels

10. output AC type for seed points

The effect of the input text prefixes is to allow a single AC pair to be used to describe different coverages, using the initial part of the text field as a modifier of the value. The input prefix is substituted by the corresponding output prefix in ACs of output segments, polygons and seed points. The AC types for output must be present, even if they will not be used, when using a coverage file: for propagation of a single coverage using /PROPAGATE=LEFT_AC and /PROPAGATE=RIGHT_AC, the AC types for created objects are governed by the appropriate qualifier on the output options, and the same defaults apply.

Comment lines, beginning !, may appear in the coverage file. The only restrictions are that the coverage numbers must begin at 1, and remain unchanged or increase by one at each record. An example of the file layout is:

```
!Cover  left right  input      output               output codes
!number code code   prefix     prefix    left  right contain polygon
seed
   1      4    5   "DISTRICT:"  ""         4     5      12       82
82
   1      4    5   "PARISH:"    ""         4     5      12       82
82
   2     1001 1002  ""          ""        1001  1002    12       82
82
```

```
/SEED=([AC:'integer',
       [FC:'integer'[,...]],
       [FILE:'file-spec'],
       [LAYER:'integer'[,...]],
       [PAIR:'file-spec'],
       [SURROUND:'text-string'],
       [USE:'keyword'])
```

/NOSEED **(default)**

- indicates that labelling information is to be extracted from seed points. The various options allow the user to specify where the seed points are to be read from, how they are to be identified and how a label can be obtained from them.

  /SEED cannot be used with the /LRCODE qualifier.

/SEED=AC:'integer'
/SEED=AC:82 **(default)**

- specifies which AC (Ancillary Code) the text is to be taken from for labelling if the /SEED=USE:AC combination has been used.

  By default seed points are expected to have an type 82 AC entry when the /SEED=USE:AC combination is used.

/SEED=FC:'integer'[,...]
/SEED=FC:1 **(default)**

- specifies the FC (Feature Code) of the polygon seed points. The seed points may either be in the junction structured input IFF file or in a separate file specified using the /SEED=FILE qualifier. All features in the input file (or in the file specified with /SEED=FILE) which have the specified FC are considered to be seed points. The maximum number of feature codes which may be specified is 32. The feature codes must lie in the range 0 to 32767. All features which lie within IFF layer 0 will be ignored.

  By default seed points are expected to have a feature code of 1.

/SEED=FILE:'file-spec'

- specifies a separate IFF file containing the polygon seed point features. Any parts of the file specification which are not supplied will be taken from the default specification LSL$IF:IFF.IFF;0.

  It is most important that the /SEED=FILE:'file-spec' argument is omitted if the seed points are included within the junction structured segments file. IPOLYGON checks that any /SEED=FILE:'file-spec' argument does not clash with the input segment file specification.

  All features in the seed file will be treated as seed points, and must accord with the rules for seed points, unless the seed point features are differentiated from other data in the file with /SEED=FC and/or /SEED=LAYER keyword arguments. The rules for IFF polygon seed point features are listed in the Description Section below. All features which lie within IFF layer 0 will be ignored.

If either the /SEED=LAYER or /SEED=FC combinations are present then seed points (whether they come from the input segment or a separate seed point file) are identified as those features that either belong to one of the specified layers or have one of the specified FCs (Feature Codes). It should be noted that these tests are complementary - that is a feature need only pass one of the tests to qualify as a seed point.

The default behaviour in the absence of both the /SEED=LAYER and /SEED=FC combinations depends on whether there is a separate seed point file or not. If there is a separate seed point file then all features within it are taken to be seed points. If there is not a separate seed point file then only features with FC 1 are deemed to be seed points.

/SEED:LAYER='integer'[,...]

- specifies the layers containing the polygon seed points. The seed points may either be in the junction structured input IFF file or in a separate file specified using the /SEED=FILE qualifier keyword. All features in the input file (or in the file specified with /SEED=FILE) which lie within the specified layer are considered to be seed points. The maximum number of layer numbers which may be specified is 32. The layer numbers must lie in the range 1 to 32767. All features which lie within IFF layer 0 will be ignored.

  By default no layers are assigned for seed points.

/SEED=PAIR:'file-spec'

- specifies a FC (Feature Code) pair file which contains one or more FC pairs. Any parts of the file specification which are not supplied will be taken from the default specification LSL$IF:IFF.FCP;0.

  The contents of the FC pair file is used to dynamically select the FC of the output polygon boundaries. The FC of the polygon boundary is determined from the FC of the corresponding polygon seed point. If seed point FC can be found in the first column of the FC pair file then the polygon boundary will have the FC given in the second column. If the seed point FC cannot be found, or no seed point has been assigned, then the FC specified by the /SEED=FC:'integer' combination will be used instead. It should be noted that this latter case always applies to the bounding polygon.

  The FC pair file is an ASCII file containing pairs of FCs - one pair per line. Comment lines may be included in the file providing they begin with a '!'. The FC pairs must be ordered so that the seed point FCs (first column) are arranged in ascending order. Seed point FCs must be unique.

  The /SEED=PAIR option can only be used when the /POLYGONS qualifier is present.

/SEED=SURROUND:'text-string'
/SEED=SURROUND:"Surrounding void" **(default)**

- specifies the label to be assigned to the bounding polygon. By default IPOLYGON uses the label "Surrounding void". The text-string may have a maximum length of 255 characters.

/SEED=USE:AC
/SEED=USE:FSN
/SEED=USE:TEXT **(default)**

- specifies how a text field is to be extracted from the seed point to provide the polygon label.

  The default behaviour is specified by the /SEED=USE:TEXT combination. Seed points are expected to be an IFF text feature and hence to contain a TX (TeXt entry). The contents of this TX entry are used as the label for the polygon which encloses the seed point.

  The /SEED=USE:AC combination indicates that the label is to be taken from the string part of an AC (Ancillary Code) entry on the seed point feature. The relevant AC type can be specified using the /SEED=AC:'integer' qualifier.

  When the /SEED=USE:FSN combination is used the seed points are expected to be unorientated point symbol features. The FSN (Feature Serial Number) of the seed point will be used as the label of the polygon which encloses the seed point.

  If the /SEED=USE:TEXT combination is superceded then the presence of TX entries will be issue a warning.

```
/SEGMENTS=([CONTAIN_IDENT_AC:'integer'],
           [CONTAIN_LABEL_AC:'integer'],
           [[NO]IDENT],
           [[NO]JUNCTIONS],
           [[NO]LABEL],
           [LEFT_IDENT_AC:'integer'],
           [LEFT_LABEL_AC:'integer'],
           [OUTPUT:'file-spec'],
           [RIGHT_IDENT_AC:'integer'],
           [RIGHT_LABEL_AC:'integer'],
           [SELECT_FC:'range;range;...'])
```

/[NO]SEGMENTS **(default)**

- indicates that coded segments are to be output to an IFF file. If /SEGMENTS is specified without any of the keyword arguments the following defaults are assumed:

  /SEGMENTS=(NOIDENT,
  NOJUNCTIONS,
  NOLABEL)

  Segments will be output to an IFF file without polygon labels and identifiers. The output of segments that are connected to one-arm junctions is governed by the state of the /ONEARM qualifier.

/SEGMENTS=OUTPUT:'file-spec'

- specifies the IFF file that the coded segments are output to. In the absence of the /SEGMENTS=OUTPUT qualifier the output file specification is taken from that of the segment input file but with the addition of the appropriate file extension. If /SEGMENTS=JUNCTIONS is specified, the extension used is '.IFJ'. If /SEGMENTS=NOJUNCTIONS is specified, or /SEGMENTS=JUNCTIONS is absent from the command line, the extension '.IFF' is used. This default mechanism may be overriden by use of the /SEGMENTS=OUTPUT:'file-spec' combination. Any file-spec argument given will be used as the output file specification. Any parts found to be missing from the file-spec will be taken from the default file-spec as described above.

/SEGMENTS=JUNCTIONS
/SEGMENTS=NOJUNCTIONS **(default)**

- provides IFF junction structured output of the left/right coded segments. Unless required at a later processing stage within a production flowline, it is recommended that the output file junction structure option is not selected as the resulting processing slows IPOLYGON output.

  By default IPOLYGON does not create junction structure.

/SEGMENTS=IDENT
/SEGMENTS=NOIDENT **(default)**

- indicates that the internally generated polygon identifiers are to be used to left/right code the output segments. The identifier is transferred as the string part of the relevant AC types.

  By default identifiers are not used to code the output segments.

/SEGMENTS=CONTAIN_IDENT_AC:'integer'
/SEGMENTS=LEFT_IDENT_AC:'integer'
/SEGMENTS=RIGHT_IDENT_AC:'integer'

- allows the user to specify which AC (Ancillary Code) types are to be used to code segments. The segments are coded by supplying the relevant internally generated polygon identifier as the string part of the AC. The /SEGMENTS=<...>_IDENT_AC:'integer' qualifier cannot be used unless the /SEGMENTS=IDENT qualifier is present.

  - o If the /ONEARM=CONTAIN qualifier is present a segment will have either a 'left' and 'right' coded pair of ACs or a single 'contains' AC.

  - o If the /ONEARM=DELETE qualifier is present then all output segments will have a 'left' and 'right' coded pair of ACs each with a different polygon identifier, unless the /SEGMENTS=JUNCTIONS qualifier is present in which case some segments will have the identifier "Ignored Segment".

  - o If the /ONEARM=USE qualifier is present then all output segments will have a 'left' and 'right' coded pair of ACs, although these may have the same identifier.

  By default the AC types used to output coded segments are:

  /SEGMENTS=CONTAIN_IDENT_AC:12
  /SEGMENTS=LEFT_IDENT_AC:4
  /SEGMENTS=RIGHT_IDENT_AC:5

/SEGMENTS=LABEL
/SEGMENTS=NOLABEL **(default)**

- indicates that labels obtained either from seed points or left/right codes are to be used to left/right code the output segments. The label is transferred as the string part of the relevant AC types. The /SEGMENTS=LABEL combination cannot be used unless the /SEED or /LRCODE qualifiers are present.

  By default identifiers are not used to code the output segments.

/SEGMENTS=CONTAIN_LABEL_AC:'integer'
/SEGMENTS=LEFT_LABEL_AC:'integer'
/SEGMENTS=RIGHT_LABEL_AC:'integer'

- allows the user to specify which AC (Ancillary Code) types are to be used to code segments. The segments are coded by supplying the relevant polygon label as the string part of the AC. The /SEGMENTS=<...>_LABEL_AC:'integer' qualifier cannot be used unless the /SEGMENTS=LABEL qualifier is present.

  o If the /ONEARM=CONTAIN qualifier is present a segment will have either a 'left' and 'right' coded pair of ACs or a single 'contains' AC.

  o If the /ONEARM=DELETE qualifier is present then all output segments will have a 'left' and 'right' coded pair of ACs each with a different polygon identifier, unless the /SEGMENTS=JUNCTIONS qualifier is present in which case some segments will have the identifier "Ignored Segment".

  o If the /ONEARM=USE qualifier is present then all output segments will have a 'left' and 'right' coded pair of ACs, although these may have the same identifier.

  If labelling has failed for a polygon then the polygon label "Undefined Polygon" is used.

  By default the AC types used to output coded segments are:

  /SEGMENTS=CONTAIN_LABEL_AC:12
  /SEGMENTS=LEFT_LABEL_AC:4
  /SEGMENTS=RIGHT_LABEL_AC:5

/SEGMENTS=SELECT_FC:'range'

- allows the user to restrict the output IFF file to contain only features with feature codes in one of the nominated ranges.

Summary of IFF Default Parameters

INPUT Segment File

Left Code	4
Right Code	5

INPUT Seed Point

Label extracted from AC	82	(/SEED=USE:AC only)

OUTPUT Polygon Boundary and Point-in-Polygon Files

IFF feature FC	1
IFF feature layer	1
Polygon Label AC	82
Polygon Identifier AC	82

OUTPUT Coded Segment File

Left Label AC	4
Right Label AC	5
Contain Label AC	12
Left Identifier AC	4
Right Identifier AC	5
Contain Identifier AC	12

NOTE: AC type 82 has a real numeric field. This can be used to record the polygon area.

RESTRICTIONS

Summary of Parameter RESTRICTIONS

- The maximum number of polygons (POLMAX) that may be processed in a single IPOLYGON run defaults to 10000, but can be increased by assigning a value to the logical name LSL$POLYGONS_POLMAX.

- The average number of sides per polygon is initially set to 5, but can be increased by assigning a value to the logical name LSL$POLYGONS_AVERAGE_SIDES.

- The maximum of input segments that may be processed in a single IPOLYGON run is calculated by multiplying the "maximum number of polygons" by "the average number of sides per polygon".

- A single polygon ring may have a maximum of 100000 coordinates.

- A single input segment may have a maximum of 20000 coordinates.

- Polygon labels are restricted to a maximum of 255 characters.

- At most 4 coverages may be propagated in a single IPOLYGON run.

Command Line Decoding RESTRICTIONS

- no two of /SEED /LRCODE and /PROPAGATE can be present at once - only one labelling mechanism can be used at a time.

- the /SEGMENTS=LABEL, /POLYGONS=LABEL, /PIP=LABEL and /ASCII=LABEL combinations cannot be present unless /SEED, /LRCODE or /PROPAGATE is present.

- the /OPTIONS=UNDEFINED combination cannot be present unless either the /SEED or /LRCODE qualifier is present.

- the /SEGMENTS=xxxx_LABEL_AC combinations cannot be used unless the /SEGMENTS=LABEL combination is present.

- the /POLYGONS=CONTAIN_LABEL_AC combination cannot be used unless the /POLYGONS=LABEL combination is present.

- the /PIP=CONTAIN_LABEL_AC combination cannot be used unless the /PIP=LABEL combination is present.

- the /SEGMENTS=xxxx_IDENT_AC combinations cannot be used unless the /SEGMENTS=IDENT combination is present.

- the /POLYGONS=CONTAIN_IDENT_AC combination cannot be used unless the /POLYGONS=IDENT combination is present.

- the /PIP=CONTAIN_IDENT_AC combination cannot be used unless the /PIP=IDENT combination is present.

- the /ONEARM=CONTAIN combination cannot be used if any of the /ASCII, /PIP or /POLYGONS qualifiers are present.

- the /ONEARM=USE, /ONEARM=DELETE and /ONEARM=CONTAIN combinations are mutually exclusive.

- the /SEED=USE:TEXT, /SEED=USE:AC and /SEED=USE:FSN combinations are mutually exclusive.

- the /OPTIONS=UNDEFINED:KEEP and /OPTIONS=UNDEFINED:DELETE options are mutually exclusive.

- the /SEGMENTS=CONTAIN_IDENT_AC and /SEGMENTS=CONTAIN_LABEL_AC combinations cannot be used unless the /ONEARM=CONTAIN combination is present.

- the /OPTIONS=NONEST combination cannot be used if either the /PIP or /SEGMENTS qualifier is present.

DESCRIPTION

Introduction

IPOLYGON is the Laser-Scan automatic **I**FF **POLYGON** creation and labelling utility. It is the primary IFF polygon creation utility and forms the basis of the Laser-Scan POLYGON Package.

IPOLYGON is designed to be run in batch mode and all options may be specified on the command line. No user interaction is required during processing.

IPOLYGON treatment of Polygons

This section deals with definition of polygons by their boundaries. It introduces a number of concepts used in the rest of this document and should be read by all users. The choice of defining a polygon by its boundary has relevance for all output options. However this section will illustrate the various options available with examples from the /POLYGONS output option. This option outputs polygon boundaries as IFF features.

For the purposes of discussion the segments in Fig. 1 (a&b) are used. The segment structure represents a stylised contour and drainage system. This dataset incorporates most of the elements likely to arise in polygon formation. For example there are nested polygons, some of which are connected to other polygons by a single segment. There is also a piece of linework that sits isolated in the middle of a polygon. It should be noted that this linework plays no part in defining areas.

There are many different requirements on polygon boundaries. Different aspects of polygon boundary formation are critical to different tasks. For this reason IPOLYGON offers a very flexible degree of control over polygon boundary formation through the use of three qualifiers:

```
$IPOLYGON/POLYGONS/ONEARM=[DELETE],[USE]

$IPOLYGON/POLYGONS/OPTIONS=[ANTI]CLOCKWISE

$IPOLYGON/POLYGONS/OPTIONS=[NO]NEST
```

Figure 1a - Polygon Test Figure

Figure 1b - Segment Directions and Numbering

The /ONEARM=[DELETE],[USE] qualifier indicates which segments can be used to form a polygon boundary. For example the user may specify whether isolated linework in the middle of a polygon, that does not itself bound an area, should be included in the polygon boundary. The /ONEARM=CONTAIN combination is a variant on the /ONEARM=DELETE combination and is explained in the section on the /SEGMENTS output option.

The /OPTIONS=[NO]NEST qualifier allows the user to specify whether first-order nesting should be taken into account when defining a polygon boundary. Nested polygons are a problem faced by all polygon construction software. It should be noted that internally IPOLYGON analyses the geometry to any required depth of nesting for the purposes of correct seed point assignment. However it produces boundaries which reflect only first order nesting or none at all.

The /OPTIONS=[ANTI]CLOCKWISE qualifier allows the user to specify in what sense the points in the polygon boundary should be ordered.

Together these three qualifiers allow the user to specify that polygon boundary formation should follow one of eight different sets of criteria. The /OPTIONS=[ANTI]CLOCKWISE combination plays no part in determining the linework involved in forming a boundary, merely its ordering. Thus, without loss of generality, the following discussion assumes that the /OPTIONS=ANTICLOCKWISE combination is present. This detail is only relevant to Fig. 2 (a&b) and Fig. 3 (a&b) where arrows indicate the direction of boundary formation.

Thus the discussion concentrates on the four remaining combinations obtained by use of the /OPTIONS=[NO]NEST and /ONEARM=[DELETE],[USE] combinations.

```
    /POLYGONS/ONEARM=DELETE/OPTIONS=NONEST      (Fig. 4)
    /POLYGONS/ONEARM=DELETE/OPTIONS=NEST        (Fig. 5)
    /POLYGONS/ONEARM=USE/OPTIONS=NONEST         (Fig. 6)
    /POLYGONS/ONEARM=USE/OPTIONS=NEST           (Fig. 7)
```

The first phase of polygon formation is the extraction of 'loops'. Any segment has two directions - the direction it was digitised in and the reverse. Specifying a segment and a direction specifies a unique directed edge. A loop is a list of directed segments such that when they are traversed the path ends where it starts and does not cross itself. Thus a segment that closed on itself could form two loops - one clockwise and one anticlockwise.

If the /ONEARM=DELETE qualifier is present then a segment *cannot* occur more than once in a loop. Fig 2a shows the set of anticlockwise loops that can be generated from Fig 1 under such a restriction. Fig 2b shows the corresponding set of clockwise loops. When Figs 2a and 2b are combined, all segments that form part of a boundary between two different polygons occur in exactly two loops - once in each direction. Those segments that do not form the boundary between two different polygons do not occur in either loop.

Figure 2a - Anticlockwise loops with /ONEARM=DELETE

Figure 2b - Clockwise loops with /ONEARM=DELETE

If the /ONEARM=USE qualifier is present then a segment *can* occur more than once in a loop. Figs 3a and 3b give the anticlockwise and clockwise loops derived from Fig 1 assuming the presence of the /ONEARM=USE qualifier. Note that this time *all* segments are used exactly twice (once in each direction). However in some cases both directions of a segment may belong to the same loop.

Fig 3b contains a clockwise loop in which each segment occurs twice. This is the loop around the isolated piece of linework. Unlike all the other loops in both Fig 3a and 3b, this loop has zero area. Such a loop is referred to as a 'tree'.

Figure 3a - Anticlockwise loops with /ONEARM=USE

Figure 3b - Clockwise loops with /ONEARM=USE

The /OPTIONS=[NO]NEST qualifier indicates whether output of polygon boundaries should take into account first-order nesting. It should be noted that clockwise and anticlockwise loops never intersect one another. Thus there is a unique ordering of these loops based on their nesting. This forms the basis of the nesting of polygons. Each clockwise loop is wholly contained within one or more anticlockwise loops. However those selected anticlockwise loops do not intersect among themselves. Thus there is a smallest anticlockwise loop that wholly contains any clockwise loop. This allows one to define parent-child relations between an anticlockwise loop and zero or more clockwise loops.

If the /OPTIONS=NEST qualifier is present then polygon boundaries reflect this nesting of loops. The fact that the nesting is only first-order indicates that any particular boundary may only bridge one generation. Each boundary is formed by taking an anticlockwise loop and logically connecting it to those clockwise loops nested within it.

If the /OPTIONS=NONEST qualifier is not present then the relations between clockwise and anticlockwise loops are ignored altogether. In particular the clockwise loops are lost, this means that linework belonging to trees will always be missing.

The arrows in Figs 5 and 7 denote the logical connections between anticlockwise and clockwise loops.

There is always one clockwise loop that does not lie within an anticlockwise loop. This represents the bounding polygon. Thus the bounding polygon boundary will only be present if the /OPTIONS=NEST qualifier is present (albeit by default). This is shown in Figs 5 and 7 by the loop pointed to by the lowermost arrow.

The /ONEARM=DELETE/OPTIONS=NEST combination (the default option, see Fig. 5) generates polygon boundaries suitable for area-fill. Using the /ONEARM=USE/OPTIONS=NEST combination (see Fig. 7) means that all segments are included in the output polygon boundaries. Although area-filling of these boundaries gives the expected results, the boundaries are unnecessarily complex if this is to be their sole use.

The /ONEARM=DELETE/OPTIONS=NONEST combination (see Fig. 4) provides a set of boundaries whose linework is sufficient to completely bound all polygon areas. However it should be noted that some elements occur twice while others occur only once. The /ONEARM=USE/OPTIONS=NONEST (see Fig. 6) combination selects out a subset of the total linework. It is difficult to provide a convincing example in which such an combination would be used.

Figure 4 - Polygon boundaries with /ONEARM=DELETE/OPTIONS=NONEST

Figure 5 - Polygon boundaries with /ONEARM=DELETE/OPTIONS=NEST

Figure 6 - Polygon boundaries with /ONEARM=USE/OPTIONS=NONEST

Figure 7 - Polygon boundaries with /ONEARM=USE/OPTIONS=NEST

IPOLYGON treatment of the Bounding Polygon.

IPOLYGON will only work reliably if it is able to trace consistently around the outer edge of the polygon area supplied in the input segments IFF file. This outer edge is referred to as the bounding polygon. The bounding polygon may represent a rectangle in a classic sheet edge situation or it may be defined by an irregular polygon, (an island for example).

IPOLYGON treats the bounding polygon is a special way. Using the terminology of the section on "IPOLYGON treatment of Polygons": the boundary of the bounding polygon is represented by a single loop. This loop can be recognised since there is no larger loop in which it is nested.

It is essential that IPOLYGON is given only one bounding polygon to deal with inside a single input segments IFF file. That is IPOLYGON only permits one loop which is not nested within another loop. IPOLYGON will not work properly, for example, if given the coastline of mainland Britain and the Isle of Wight in the same file.

This is clearly an important restriction. If several islands are to be processed at once, then a new bounding polygon should be generated so as to include all the input segments. If the range of the input segments is known then a rectangle can be simply generated that includes them all. It is advisable to provide a margin around the input data, otherwise the linework in the bounding polygon may corrupt the original linework during the creation of junction structure.

It should be noted that this is a significant change in operation from the last version of IPOLYGON.

If deriving polygon labels from seed points, the bounding polygon should never be given a seed point. The /SEED=SURROUND='text-string' qualifier should be used to define the label to be used for the bounding polygon. By default the bounding polygon will be given the label "Surrounding void".

Labels and Identifiers

IPOLYGON uses the concept of **Labels** and **Identifiers**. These are text-strings of up to 255 characters that can be used to reference the polygons that are formed using IPOLYGON.

A label is a user-defined text-string that is allowed to vary from polygon to polygon. These labels are inferred to belong to a particular polygon by one of two mechanisms. Either the label is initially held in a seed point feature whose location within a particular polygon is used, or the input segments have already been coded with left/right labels. Clearly labelling can fail for a number of reasons, for example insufficient or badly placed seed points or inconsistent left/right coding of input segments. Labels need not be unique. However if the /SEGMENTS=LABEL option is used then warnings will be issued if a segment has the same left and right label.

In contrast identifiers are internally generated text-strings that are guaranteed to be unique. The ability to generate identifiers is purely contingent on the ability to form the polygons themselves. Identifiers have the format of a piece of user-supplied text followed by a number. The numbers are internally generated and guaranteed to be unique, although not necessarily consecutive. The text part of the identifier is controlled by the /OPTIONS=IDENT_TEXT:'text-string' combination. By default the text part is "Polygon ".

It should be noted that polygon formation itself is a purely geometric operation and is not concerned with labels or identifiers. Thus the failure to obtain a full set of labels for the polygons does not prevent the operation of IPOLYGON.

Labels and identifiers can be output to IFF files as the string field of specified AC entries. The output of labels and identifiers are independent and each mechanism has its own set of default AC types, although these can be overridden by the user.

If a user only wants to output geometry, for example create a set of polygon boundaries, then both label and identifier output can be switched off. The output of labels and identifiers can be individually switched for each of the various output options - be it polygon boundary, polygon segments or points-in-polygons. The graphic content of the output segments or polygon boundaries will, however, be complete and the output data may be used for check plotting purposes. The geometric completeness of the output data may then be checked without having incurred the overhead of point in polygon tests used to allocate seed points, or disk file access time to read left/right AC text from the input segments.

Derivation of Polygon Labels

Polygon labels are text-strings of up to 255 characters. These are obtained from one of three sources:

- o A label may be extracted from information carried by 'seed points'. A seed point is a point feature that can be related to a polygon by its location within the polygon.

- o A label may be extracted from the string fields of pairs of ACs (Ancillary Codes) which are used to code the input segment file with left/right codes.

- o The third labelling method is a variant of left/right coding, in which labels are extracted from specified parts of the AC string fields of the input segments, and propagated onto segments forming part of the same polygon as the original segment, or of a polygon adjacent to it. In this mode of operation up to four independent codings may be propagated and used to produce combined labels.

Seed Point Identification

If the /SEED qualifier is present, polygon seed points are used to define the label for each polygon. The following rules must be observed when creating and manipulating IFF features which are to be used as polygon seed points within the Laser-Scan POLYGONS package:

- A seed point must lie **within** the polygon to which it applies. It must not lie outside or on the edge of a polygon.
- Every polygon must have a seed point.
- A polygon can have only one seed point.
- A seed point must have a single ST (STring) entry, containing a single locating point.

Seed points may be included in the input segment file or be supplied using a separate seed point file. This latter case is indicated by use of the /SEED=FILE qualifier. The user must supply sufficient information to make the identification of seed points possible. In addition to the ability to specify a separate seed point file the user can specify a number of layers and/or FCs (Feature Codes) by use of the /SEED=LAYER and /SEED=FC qualifiers.

If either the /SEED=LAYER or /SEED=FC qualifiers are present then seed points (whether they come from the input segment or a separate seed point file) are identified as those features that either belong to one of the specified layers or have one of the specified FCs (Feature Codes). It should be noted that these tests are complementary - that is a feature need only pass one of the tests to qualify as a seed point.

The default behaviour in the absence of both the /SEED=LAYER and /SEED=FC qualifiers depends on whether there is a separate seed point file or not. If there is a separate seed point file then all features within it are taken to be seed points. If there is not a separate seed point file then only features with FC one are deemed to be seed points.

Label Extraction From Seed Points

IPOLYGON provides three mechanisms by which a label can be derived from the relevant seed point:

1. **AC:** the label is the string field of the first Ancillary Code entry (with the required AC type) found in the seed point feature. This mechanism can be selected by using the /SEED=USE:AC qualifier. By default the required AC type is type one but the user can select an alternative by use of the /SEED=AC:'integer' qualifier.

2. **FSN:** the label is a text-string that represents the seed point FSN (Feature Serial Number). This mechanism can be selected by using the /SEED=USE:FSN qualifier.

3. **TEXT:** the label is extracted from the first TeXt entry in the seed point feature. This mechanism can be selected by using the /SEED=USE:TEXT qualifier.

Composite text in text features is not supported. Only a single TX (TeXt) entry will be read from the seed point feature. Composite texts can be used to define paragraphs of text as a single graphic feature. They are, however, only only available as a licensed option within a subset of the Laser-Scan LAMPS (Laser-Scan Automated Map Production System) system.

A seed point may contain any number of AC (Ancillary Code) entries. However these will only be used if /POLYGONS or /SEED=USE:AC is present.

Since no seed point is provided to label the bounding polygon a default label of "Surrounding void" is used. The user can override this choice by use of the /SEED=SURROUND:'text-string' combination. If the /SEED=USE:FSN qualifier is present then the bounding polygon has the label '65535', independent of the presence of the /SEED=SURROUND qualifier.

Depending on the mechanism to be used to extract labels from seed point data, the seed points must satisfy certain conditions.

1. If /SEED=USE:AC is present then each seed point feature must contain an AC entry of the correct AC type. The presence of of TeXt entries in such features will be signalled.

2. If /SEED=USE:FSN is present then each seed point feature must be an unorientated symbol feature. The presence of of TeXt entries in such features will be signalled. The FSN of seed-points must be unique.

3. If /SEED=USE:TEXT is present then each seed point feature must be a text feature and must contain a TX (TeXt) entry. Seed point text features must have a single locating point and a rotation defined by an RO (ROtation) entry.

Seed points may have multiple AC entries. If the /POLYGONS qualifier is present then the AC entries from the relevant seed point will be copied across to closed polygon boundary features if the /SEED=USE:AC or /SEED=USE:TEXT options are selected.

Left/Right Coding

If the /LRCODE qualifier is present then labels are extracted from the text field of the left and right AC codes on the input segments. By default the left code has AC type 4, and the right code has AC type 5. These default values can be overridden by the use of the /LRCODE=LEFT_AC:'integer' and /LRCODE=RIGHT_AC:'integer' qualifiers. In the event of an input segment having multiple left and right AC entries, IPOLYGON will use only the first AC found. All subsequent left and right ACs within that feature will be ignored. The label obtained from the AC text field in the first segment is then checked for consistency against those labels obtained from other segments in the polygon boundary.

Inconsistent labelling will result in warnings, and new entries in the LITES2 guidance file if it is being used. Failure to obtain consistent labelling will not result in IPOLYGON aborting, merely the use of the label "Undefined Polygon" in place of the desired label.

Rules For Left/Right Coded Input Segments

The following rules for left/right coding must be obeyed for successful IPOLYGON processing:

- A segment must have one left-code type AC (Ancillary Code) and one right-code type AC. By default these AC types are 4 and 5.
- The text field of the left-code type AC must carry the label of the polygon which lies to the left of the segment, relative to the direction of digitising.
- The text field of the right-code type AC must carry the label of the polygon which lies to the right of the segment, relative to the direction of digitising.
- If multiple type left-code and right-code ACs are supplied then only the first of each will be retained and used.
- The polygon labels must not exceed 255 characters.
- Any information placed in the code (longword) field of the ACs will be ignored.

Propagation Of Labels

Before describing the labelling operation involving propagation, it is helpful to define the concept of a coverage. This term will be used to mean, simply, a division of the area of interest by linear features into non-overlapping polygons. Labelling by seed points and left/right codes deal with a single coverage of the area contained within the bounding polygon, together, in some cases, with a set of features each contained in one polygon of the coverage.

A coverage is defined by a pair of ACs - a left/right pair as above - and, optionally, a set of text prefixes. A left coded segment in the coverage is then a segment in the input file having an AC whose type matches the left type and whose text string begins with one of the specified prefixes; a right coded segment is defined similarly. If text prefixes are not being used, the left and right AC types may be specified with a qualifier, and default to 4 and 5 respectively. Otherwise, and where multiple coverages are being used, a file is read to determine the AC type pairs and associated text prefixes. AC 4 PARISH NEWTON LE WILLOWS AC 5 DISTRICT ST HELENS AC 4 COUNTY LANCASHIRE Then to describe the coverage consisting of the smaller administrative

It often happens that two or more coverages of a given area are defined, and that there is a need to analyse them together. This involves forming the common area (intersection) of each pair of polygons, one from each coverage, and associating each such resultant polygon with its 'parent' polygons. This operation is referred to as polygon overlay.

Given a file containing two or more independent coverages, a geometric merging and splitting can be performed using the STRUCTURE package. The resulting file consists of segments each of which belongs to (at least) one coverage. The overlay operation is reduced to the identification of each segment in terms of the input coverages: this may be achieved by propagating each coverage in turn.

A prerequisite for labelling by left/right coding is that all input segments should possess both a left and a right AC. It is clear that this condition can be relaxed to allow some segments to exist without one or both of these ACs, without causing the intended labelling of any polygon to be ambiguous; if just one segment of a polygon P described in an anticlockwise sense has a left AC, then that AC implicitly applies to all the segments making up the polygon (as a left or right AC, depending on the segment direction). Suppose, further, that the polygon P is divided into two polygons, P1 and P2, by a feature which is not part of the coverage under consideration (for example, a woodland polygon split by a river), and the coded segment lies entirely in one of the resultant polygons, say P1. Then the coding of this segment applies to all segments of P1, including the dividing feature, and because this feature is not part of the coverage then it also applies to all segments of P2. Thus the coding may be extended across any linear feature which is not part of the coverage.

The propagation of ACs is simply the process of explicitly adding ACs to uncoded segments of a coverage in this manner. Two types of propagation are distinguished, according to the way in which segments initially having neither left nor right codes are handled. The default operation, partial propagation, treats such segments as if they were part of the coverage, and does not extend coding across them. The alternative operation, full propagation, assumes that such segments are not part of the coverage and propagates across them. It is not normally appropriate to apply partial propagation to multiple coverages.

If the /PROPAGATE qualifier is present then labels are extracted from the text field of the left and right AC codes on the input segments. If a single coverage is being processed, and text prefixes are not to be used then the coverage may be defined by specifying a pair of AC types. By default the left code has AC type 4, and the right code has AC type 5. These default values can be overridden by the use of the /PROPAGATE=LEFT_AC:'integer' and /PROPAGATE=RIGHT_AC:'integer' qualifiers. If more than one coverage is to be processed, or if text prefixes are to be used to specify the coverages, the definition of the coverages must be read from a file, using the /PROPAGATE=COVERAGE_FILE qualifier. The file layout is described in the qualifiers section. In the event of an input segment having multiple AC entries defining the left or right area for a single coverage, IPOLYGON will use only the first.

Inconsistent labelling will result in warnings, and new entries in the LITES2 guidance file if it is being used. Failure to obtain consistent labelling will not result in IPOLYGON aborting, merely the use of the label "Undefined Polygon" in place of the desired label.

IPOLYGON checks on input IFF data

IPOLYGON performs the following checks on the input IFF segment and (optionally) seed data:

(A) Junction structured segment data.

- o Each segment is checked to ensure that it has a junction at both ends. Segments which lack a junction are noted (and if selected, commands are written to a LITES2 command file). IPOLYGON attempts to continue.

- o Each junction is read and the number of arms determined. Any zero arm junctions are reported. If /ONEARM=WARN is specified any 1 arm junctions will also be reported and, if selected, messages written to a LITES2 command file.

- o Each segment is checked to ensure that it has a minimum of two coordinates.

- o IPOLYGON requires that all segment FSNs (Feature Serial Numbers) are unique. A check for duplicate FSNs is always performed.

(B) Seed points (if /SEED is present).

- o Each seed point is checked for the presence of a label:

/SEED=USE:AC	Seed points must contain an AC entry of the type specified by /SEED=AC:'integer'.
/SEED=USE:FSN	Seed points must have unique FSN.
/SEED=USE:TEXT	Seed points must contain a TX entry.

- o The ST (STring) entry of points are checked. There should be only one ST in a seed point feature and this should contain only one coordinate point.

- o Seed point coordinates are checked against the coordinate range of the segment data. Any which lie outside the segment data range must be outside a polygon and are therefore in error.

- o IPOLYGON requires that all seed point FSNs (Feature Serial Numbers) are unique.

IPOLYGON Production Flowline

IPOLYGON is designed to be used within the Laser-Scan LAMPS (Laser-Scan Automated Map Production System) environment. The user should have access to, and be familiar with the use of, the following Laser-Scan software products:

- IMP - IFF Map Processing Package
- ILINK - to generate IFF junction structure
- LITES2 - the Laser-Scan IFF graphic editor

IPOLYGON can be used in a variety of scenarios: for example segments may be coded from a set of seed points or a set of seed points can be generated from coded segments.

- **Generate Uniquely Coded Segments:**

  IPOLYGON is used to generate a set of coded segments. This illustrates the most basic function of IPOLYGON. Geometry is analysed into a set of polygons. The result of this analysis is expressed as codes on the input segments.

  1. Digitise Geometry - it should be noted that it is possible to digitise large linear features whole. This is because they will be broken up by ILINK as necessary. The geometry may include double digitising. This form of geometry is termed 'spaghetti'.

  2. Create Junction Structure - ILINK is used repeatedly to generate junction-structured IFF output. If the spaghetti includes double digitising the ILINK/LLJOIN and ILINK/MERGE options should be used. The geometry is now broken up into a number of uncoded segments.

  3. Uniquely Code Segments - run IPOLYGON/SEGMENTS=IDENT to code the segments. The internally generated polygon identifiers are used to code the segments. Segments can be coded with left/right codes or left/right/contain codes depending on the /ONEARM qualifier.

- **Generate Area-Filled Polygons**

  IPOLYGON is used to generate an IFF file which contains a set of area-filled polygons. Different polygons may have the same fill-pattern. Segments that do not contribute to the polygon boundaries are ignored.

  1. Digitise Geometry - see above.

  2. Create Junction Structure - see above.

3. Digitise Seed Points - a file of seed points is digitised. Seed points with different FCs are used to indicate the type of area-fill for each polygon.

4. Generate Filled Polygons - run IPOLYGON/POLYGONS/SEED=PAIR. The pattern for the area-fill of a polygon is determined by the FC of the polygon boundary. This in turn is derived from the FC of the seed point via the FC pair file.

   It should be noted that the /OPTIONS=NONEST should not be used because it may result in overlapping filled areas. Similarly the bounding polygon must not be area-filled since LITES and FPP ignore the sense of a boundary in determining which portions to fill. Since the FC for the boundary polygon feature cannot be derived from a seed point FC (the bounding polygon does not have a seed point), the /SEED=FC:'integer' combination can be used to give an appropriate FC.

- **Generate a Set of Seed Points:**

  IPOLYGON is used to automatically generate a set of seed points from an input IFF file of coded segments.

  1. Digitise Geometry - this may be done in one of two ways:

     EITHER

     (a) segments are individually digitised and each given both left and right codes - there should be no double digitising.

     OR

     (b) closed polygon boundaries are digitised and a left (or right) code given depending on whether the boundary is digitised in an anticlockwise (or clockwise) direction.

  2. Create Junction Structure -

     (a) since individual segments have been digitised and there is no double digitising ILINK can be used to snap line ends together and onto other line ends. ILINK/STRUCTURE can then be used to produce junction-structured output.

     (b) The presence of double digitising means that the ILINK/LLJOIN and ILINK/MERGE/ACP options are required. The ACP file is required to turn left codes into right codes (and vice versa) when segments digitised in opposite directions are merged together.

  3. Generate Seed Points - run IPOLYGON/LRCODE/PIP=LABEL to generate a set of points-in-polygons. These are labelled with the data from the coded segments.

Program Operation

Although IPOLYGON offers the user with many processing and output options the basic structure of program operation is easily defined. After user specification of the command line IPOLYGON processing may be broken down into 17 distinct stages. No user intervention is required at any stage.

1. The command line is interpreted and user defined options determined. Both input and output files are opened to check that IPOLYGON can open all the required files. Note this includes output IFF files. These are subsequently closed until the relevant output module opens them again for output. During this time the IFF files are degenerate but valid IFF files and no attempt should be made to use them by other applications.

2. **Building IFF Address Tables**

   The junction structured IFF input file is read in. Checks are performed on the junction structure of the input segments. If the /LRCODE qualifier is present then each segment will be checked to carry both left and right labels.

3. **Checking Seed Points**

   /SEED present

   If labelling from seed points which have been specified to lie in a separate file, the seed point IFF input file is read in. All seed points are checked to have a single locating coordinate. They are also checked for the presence of a label.

4. **Forming Polygons**

   Polygon formation is attempted. Errors due to the presence of double digitising may detected at this stage and prevent further execution. Once all polygons have been formed an attempt is made to identify the bounding polygon.

5. **Setting up Spatial Index for Polygons**

   A sectored spatial index is set up for the polygon segment lists. The polygon range is used to calculate which sector (or sectors) a polygon lies within and a list of the polygons within each sector is compiled.

   Internal lists of component segments for each polygon are constructed and the coordinate range of each polygon determined.

6. **Setting up Spatial Index for Seed Points**

   /SEED present

   This step is only performed if labelling from seed points. Using the IFF input file range, a sectored spatial index is set up within memory to speed the seed point selection process.

7. **Identifying Isolated Polygons**

   Isolated polygons are those loops that will form the inner boundaries of 'doughnut' type polygons. These are identified according to the sense of the loop. The bounding polygon is a special case that is ignored by this procedure.

8. **Identifying Nested Polygons**

   More than one isolated polygon found.

   Since loops do not intersect it is possible to determine the nesting of two loops by performing a point-in-polygon test on a point from one loop with respect to the other. A loop may be contained by more than one other loop. 1st order nesting relates a loop to those loops that only it contains. The parent loop is then removed from consideration and the operation repeated. By performing multiple passes it is possible to deal with any level of nesting.

   The use of sectored spatial indexes allows one to quickly determine which loops might (possibly) be nested within another. Explicit calculation need only be performed on this restricted subset of loops.

9. **Identifying Nested Linework**

   /ONEARM=USE present and more than one tree found.

   Linework that does not form part of a boundary between two different polygons and is not connected to any boundary are termed trees. It is necessary to identify which polygon each tree lies in. Since internally a tree is represented as a degenerate (zero-area) polygon, this process is very similar to that of determining the nesting of isolated polygons. There is no need for multiple passes since trees are unable to contain other polygons.

10. **Assigning Seed Points**

    /SEED present

    This step is only performed if labelling from seed points. Using the spatial indexes for both the polygons and seed points, seed points are assigned to polygons. Warnings will which polygons do not have exactly one seed point. Labelling will have failed for such polygons and the label "Undefined Polygon" will be substituted.

11. **Assigning Left/Right Codes**

    /LRCODE present

    A check is performed on the consistency of the left/right labelling of all the polygons that have been formed. This involves tracing along all the edges that form the polygon boundary and checking the relevant left (or right) labels for each edge. If any edge does not have the same label as the first edge then the labelling is declared inconsistent and the label 'Undefined Polygon' is used.

12. **Creating Segment Index for Propagation**

    /PROPAGATE present

    This step is only performed if partial or full AC propagation is to be performed. Indexes are set up to allow efficient association between segments and polygons.

13. **Assigning Propagated Codes**

    /PROPAGATE present

    For each coverage, each initially coded segment is associated with the uncoded segments in the polygon of which it forms a part. Where an uncoded segment is associated with more than one initially coded segment, a check is made that the segment label for that coverage is consistently defined.

14. **Writing Polygon Boundary File**

    /POLYGONS present

    The polygon boundary IFF file is reopened, a set of polygon boundary features written to it, and the file is closed.

15. **Writing Coded Segment File**

    /SEGMENTS present

    The coded segments IFF file is reopened, a set of coded segment features written to it (with or without junction structure), and the file is closed.

16. **Writing Point-in-Polygon File**

    /PIP present

    The point-in-polygon IFF file is reopened, a set of symbol features written to it, and the file is closed.

17. **Summary Listing of Polygon Components**

    /ASCII present

    Lists of directed segments are output to the output ASCII file.

IPOLYGON Output Options

1. **IPOLYGON/ASCII** option: an **ASCII** listing of the directed segments that go to make up polygon boundaries is produced. The ordering of the edges in the polygon boundary may be specified to lie in a clockwise or anticlockwise direction. Each polygon boundary may display the associated polygon label and identifier.

   If /OPTIONS=NEST is specified (default) the polygon boundary listings will include any first order nested polygons.

   In the event of polygon labelling failure, the retention or omission of the polygon from the output file is determined by the /OPTIONS=UNDEFINED keyword argument.

2. **IPOLYGON/PIP** option: an **IFF** file containing point symbol features is generated. These features are positioned so that there is one per polygon, with the exception of the bounding polygon. Each point feature may contain AC entries to carry the polygon label and identifier.

3. **IPOLYGON/POLYGONS** option: an **IFF** file containing complete closed polygons as single features is generated. The coordinates of the polygon may be specified to lie in a clockwise or anticlockwise order. Each polygon feature may contain AC entries to carry the polygon label and identifier.

   If /OPTIONS=NEST is specified (default) the closed polygon features will include any first order nested loops.

   In the event of polygon labelling failure, the retention or omission of the polygon from the output file is determined by the /OPTIONS=UNDEFINED keyword argument.

4. **IPOLYGON/SEGMENTS** option: an **IFF** file containing left/right coded segment features is generated. Left/right pairs of ACs can be used to carry polygon labels and identifiers.

   In addition the /ONEARM=CONTAIN option can be used to code segments that do not form part of a polygon boundary, with a 'contain' AC code.

   If /SEGMENTS=NOJUNCTIONS is specified (default) then the output file will not contain junctions. Segments that are excluded via the /ONEARM qualifier will not be output. If /SEGMENTS=JUNCTIONS is specified then the output file will have the same junction structure as the input segment file. Those segments that would otherwise have been excluded will have a label/identifier of "Ignored Segment".

IPOLYGON/ASCII Output

The /ASCII option provides a list of polygon boundaries expressed as lists of directed segments. A directed segment is described by specifying a segment and its 'direction' (frequently denoted 'positive' or 'negative'). The direction indicates in which order the segment should be traversed - either in the same direction as it was digitised (positive) or in the reverse direction (negative). The segments are specified by the segment FSN (Feature Serial Number).

The characteristics of this file are best indicated by the following examples:

- The segment listing resulting from the command line:

  $IPOLYGON/ASCII=IDENT/ONEARM=DELETE 'testfile'

  where the testfile is that shown in Fig 1b. Note that not all segments are output and that the polygon identifiers (although unique) are not consecutive.

```
BOUNDING POLYGON IDENT: "Polygon 1"
SEGMENTS:  1 2

POLYGON IDENT: "Polygon 2"
SEGMENTS:  -1 3 ( -10 ) ( -8 )

POLYGON IDENT: "Polygon 3"
SEGMENTS:  -2 -3

POLYGON IDENT: "Polygon 4"
SEGMENTS:  8 ( -9 )

POLYGON IDENT: "Polygon 7"
SEGMENTS:  9

POLYGON IDENT: "Polygon 10"
SEGMENTS:  10 ( -11 )

POLYGON IDENT: "Polygon 11"
SEGMENTS:  11
```

- The segment listing resulting from the slightly modified command line:

  $IPOLYGON/ASCII=IDENT/ONEARM=USE 'testfile'

  where the testfile is again that shown in Fig 1b.

```
BOUNDING POLYGON IDENT: "Polygon 1"
SEGMENTS:  1 2

POLYGON IDENT: "Polygon 2"
SEGMENTS:  -1 3 ( 7 16 -8 -16 -15 -10 15 -7 ) ( 4 -5 5 6 -6 -4 )

POLYGON IDENT: "Polygon 3"
SEGMENTS:  -2 -3

POLYGON IDENT: "Polygon 5"
SEGMENTS:  8 17 -17 ( -9 )

POLYGON IDENT: "Polygon 6"
SEGMENTS:  9

POLYGON IDENT: "Polygon 8"
SEGMENTS:  10 -14 18 -18 -13 -11 13 14

POLYGON IDENT: "Polygon 9"
SEGMENTS:  11 -12 12
```

- o The result of run on a hypothetical dataset containing soil-type polygons. The polygons have been labelled by a set of seed points and the /OPTIONS=AREA combination is present.

```
                BOUNDING POLYGON: "Surrounding void"
                SEGMENTS:  409 450 451 452 453 454 455 457 458 459 460 461 462 463
464 -465
                         466 449 448 447 445 446 -444 442 443 441 440 439 437 -438
436 434
                AREA: 2003.8

                POLYGON LABEL: "Haliomione portulacoides"
                SEGMENTS:  1 -415 13 3 -411
                SEED POINT FSN: 558
                SEED POINT POSITION: 11599.976 11014.499
                AREA: 78.9

                POLYGON LABEL: "Puccinellia maritima"
                SEGMENTS:  -1 -410
                SEED POINT FSN: 557
                SEED POINT POSITION: 10739.551 10261.782
                AREA: 1276.6

                POLYGON LABEL: "Suaeda maritime"
                SEGMENTS:  234 -10 -4 -17 -418 -16 20 74 -77 -76 -231 236 -75 -91
-409
                           416 ( 413 -414 ) ( 415 ) ( 410 -411 412 )
                SEED POINT FSN: 511
                SEED POINT POSITION: 15875.437 12733.957
                AREA: 23.1

                POLYGON LABEL: "Aster tripolium"
                SEGMENTS:  -2 -412 -3 -8 -6 -11
                SEED POINT FSN: 556
                SEED POINT POSITION: 11977.042 11659.745
```

AREA: 786.3

Records relating to the bounding polygon are always located at the start of the file and the first of these is flagged by the identifier "BOUNDING POLYGON:"

The FSNs of segments which form nested polygons are enclosed in round brackets. The brackets are opened before the first segment of the nested polygon and closed after the last.

By default IPOLYGON always calculates and output 1st order nested polygons. This may suppressed with the /OPTIONS=NONEST qualifier. If the /OPTIONS=AREA combination is present while 1st order nesting is suppressed then the area values for 'doughnut' type areas will be incorrect.

IPOLYGON/PIP Output

An IFF file of symbol features is generated. Each symbol feature is placed to lie within precisely one polygon. The output file is determined by the the /PIP=OUTPUT:'file-spec' combination. The layer and FC of the symbol features are determined by the /PIP=(FC:'integer',LAYER:'integer') combinations. The use of labels and identifiers is controlled with the /PIP=(IDENT,LABEL) combination. The numeric field of the output label and/or identifier ACs will carry the polygon area if the /OPTIONS=AREA combination is present.

The /PIP=ITERATE:'integer' combination allows the user to indicate that more than one iteration can be used in placing the point-in-polygon. All values of the iterate parameter produce points that lie inside the specified polygon. The polygon definition includes inner boundaries owing to first order nesting and, if the /ONEARM=USE combination is present, any linework within the polygon. Increasing the number of iterations attempts to place the point feature so that it is equally spaced between the polygon boundaries directly to its left and right, and also directly above and below it. This method is not guaranteed to produce perfect results even after a large number of iterations and should be used with care.

IPOLYGON/POLYGONS Output

An IFF file of polygon boundary features is generated. The output file is determined by the the /POLYGONS=OUTPUT:'file-spec' combination. The layer and FC of the boundary features are determined by the /POLYGONS=(FC:'integer',LAYER:'integer') combinations. If the /SEED=PAIR qualifier is present then the FCs of the boundary features can be derived from the FCs of the relevant seed points. The use of labels and identifiers is controlled with the /POLYGONS=(IDENT,LABEL) combination. The numeric field of the output label and/or identifier ACs will carry the polygon area if the /OPTIONS=AREA combination is present.

Nested polygons are a problem faced by all polygon construction software. IPOLYGON internally can deal with n'th order polygon nesting for the purposes of correct seed point assignment, but it produces output files which reflect only first order nesting.

IPOLYGON outputs first order nested polygons to either the /POLYGONS IFF file option or to the /ASCII output file. The coordinates of nested polygons are always ordered in the opposite direction to that of the enclosing polygon. Thus if the user specified that polygons are to be formed in a clockwise direction (/OPTIONS=CLOCKWISE) then any 1st order nested polygons will have coordinates ordered in an anticlockwise direction.

If the /POLYGONS IFF output option is chosen, nested polygons are joined to their enclosing polygon using invisible (pen-up) steps. Consider a forest containing a glade: then the IFF feature representing the forest boundary traces round part of the outside of the forest then an invisible (pen-up) step is provided between the outer boundary and a point on the glade boundary. The feature then traces around the glade boundary (in the opposite sense) until the point where the invisible step is reached. An invisible step is then made back along the original invisible step and tracing of the forest outer boundary is continued.

Because the two invisible steps exactly overlay each other, the accuracy of polygon area calculations is maintained. There is no limit to the number of glades which can be incorporated into the forest boundary feature. The glades are joined together by a sequence of invisible steps, each one beginning where the previous one finishes.

By default IPOLYGON always calculates and output 1st order nested polygons. This may suppressed with the /OPTIONS=NONEST qualifier. If the /OPTIONS=AREA combination is present while 1st order nesting is suppressed then the area values for 'doughnut' type areas will be incorrect.

IPOLYGON/SEGMENTS Output

An IFF file of coded segments is generated. The selection of the output file is controlled by the /SEGMENTS=OUTPUT:'file-spec' combination. The use of labels and identifiers is controlled with the /SEGMENTS=(IDENT,LABEL) combination.

This section assumes a knowledge of the /ONEARM=[DELETE],[USE] combinations as described in the section on "IPOLYGON treatment of Polygons". In particular the user should be clear as to which segments are implicated (or not) by the various /ONEARM options.

The presence of the /SEGMENTS qualifier automatically precludes the use of the /OPTIONS=NONEST combination. However a new option is introduced, namely the /ONEARM=CONTAIN combination. This is shown to be a simple variant on the /ONEARM=DELETE combination.

```
    /SEGMENTS/ONEARM=CONTAIN

    /SEGMENTS/ONEARM=DELETE

    /SEGMENTS/ONEARM=USE
```

With the /ONEARM=[USE],[DELETE] options the segments that are output are given a pair of left/right codes. If the /ONEARM=CONTAIN combination is present an output segment will have either a pair of left/right codes or a contain code.

If the /ONEARM=DELETE combination is present only a subset of the input segments are output. These are those segments that form part of a boundary between two different polygons. This means that the left and right identifiers (if selected) are necessarily different. If labels are output then there is no restriction that they need be different, but warnings are issued if they are not.

If the /ONEARM=USE combination is present then all segments are output with a pair of left/right codes. However a segment may have the same polygon on each side, for example a spur of linework that enters a polygon. Thus left and right identifiers are not guaranteed to be different.

The /ONEARM=CONTAIN combination is a variant on the /ONEARM=USE combination. Those segments that have the same polygon on both side have a contain code rather than a duplicate left/right pair.

If the /ONEARM=DELETE combination is present the output of those segments that do not form part of polygon boundaries is determined by the /SEGMENTS=[NO]JUNCTIONS combination. If junction structure output is required then it is always of the same structure as the input segment file. Thus the segments cannot be omitted. However if labels or identifiers are being output then they will have the text "Ignored Segment".

Using IPOLYGON with LITES2

If the /LITES2 qualifier is specified IPOLYGON creates a LITES2 command file which can be used to direct the user to the position of the suspected error within the Laser-Scan LITES2 graphical editor. The positions supplied in the LITES2 command file locate the suspected error.

Not all errors detected by IPOLYGON are output to LITES2 command file as a LITES2 command requires a coordinate position. Clearly errors related to the failure to read an IFF ST (STring) entry cannot result in the output of a LITES2 command. The LITES2 command file should therefore be used in conjunction with IPOLYGON terminal output, (which may optionally be directed to listing file using the /LIST qualifier).

If the user selects the /SEED=FILE=file-spec option and takes seed points from a separate IFF file to that read for segment data it is important that the user is aware that the LITES2 command file reflects errors which occur in both files. To aid identification of the type of data which is at fault each error message sent to the LITES2 command file is made up of two parts - the type of error and an error message relating to the particular fault at hand.

All errors are recorded in the LITES2 command file in essentially the same format:

```
! Tell the user what is wrong
%MESSAGE type of error:
%MESSAGE particular error message
! Locate cursor
%POSITION     'x coord'    'y coord'
! Is the point within the visible window?
%TEST $CURSINWIN
! If not, centre the window about the point and redraw the window
%ELSE %ZOOM 1
! Ring bell
%PING
! Pass control to the user for corrective action
%RESPOND
! Pass control back to the command file
%ABANDON
%ABANDON
```

There are several types of error that can be written to the LITES2 command file, these are dealt with briefly in the following sections.

Junction Check Errors

These errors occur while the input segment file is being read and its junction structure is being analysed.

- **'integer' arm junction detected at ('real', 'real')**

  A zero or one-arm junction has been detected. The latter will be reported if the /ONEARM=WARM combination is present.

- **Missing junction in IFF segment feature with FSN 'integer' ('integer')**

  The specified segment feature has one (or both) junctions missing.

- **Invalid IFF segment feature with FSN 'integer' ('integer')**

  The specified segment feature is invalid in some way - for example the ST entry is missing.

- **Duplicate segment FSN 'integer' found - FSN 'integer' ('integer')**

  There is more than one segment feature with the same FSN

Seed Point Errors

These errors occur while the seed points are being read in prior to polygon formation.

- **Multiple ST entries in feature with FSN 'integer' ('integer')**

  Seed point with more than one ST entry

- **Duplicate seed point FSN 'integer' found - FSN 'integer' ('integer')**

  More than one seed point with the same FSN

- **Multi-point ST in feature with FSN 'integer' ('integer')**

  Seed point with more than one coordinate

- **Text component found in FSN 'integer' ('integer') - ignored**

  Seed point with unexpected TS entry (For an explanation of text components see the "IFF User Guide", in particular the section on "Feature level entries").

- **TX entry missing from feature with FSN 'integer' ('integer')**

  The /SEED=USE:TEXT option is being used but the seed point has no TeXt entry.

- **AC ('integer') entry missing from feature with FSN 'integer' ('integer')**

  The /SEED=USE:AC option is being used but the seed point has no AC entry of the correct AC type.

- **/SEED=USE:FSN selected - TX or AC ('integer') entry in feature with FSN 'integer' ('integer')**

  Warning - the /SEED=USE:FSN option is being used but TeXt or AC entries (which might have been expected to supply the label) are also present.

Segment AC Check Error

This error occurs when the input segment file is read in and the /LRCODE qualifier is present. Input segments must have *both* left and right labels.

- **Missing left or right AC in IFF segment feature with FSN 'integer' ('integer')**

Double Digitising Errors

These errors locate areas of suspected double digitising. This constitutes an invalid junction structure for IPOLYGON. These errors must be removed before IPOLYGON can form polygons.

- **Suspected coincident segments near features with FSN 'integer' and 'integer'**

- **Suspected double digitising near feature with FSN 'integer'**

Polygon Formation Errors

These errors occur during polygon formation. If the error cannot be traced to bad junction-structure then an SPR should be submitted.

- **No junction arms to follow in feature with FSN 'integer' - polygon abandoned**

- **Unable to find current arm in feature with FSN 'integer' - polygon abandoned**

Seed Point Assignment Errors

These errors occur once polygon formation has been completed and an attempt is being made to label the polygons from the seed points.

- **Seed point assignment failed in polygon near ('real', 'real')**

  No seed points were assigned to a polygon. Output of the polygon will be determined by the /OPTIONS=UNDEFINED option. Labelling of the polygon will result in the "Undefined Polygon" label.

- **Disregarding multiple seed point in polygon at ('real', 'real')**

  An attempt has been made to assign more than seed point to a polygon. Only the first seed point will be considered for labelling purposes.

- **Segment has same polygon label on both sides at ('real', 'real')**

  Warning - the left and right labels for this segment are the same.

Polygon Labelling Errors

These errors occur once polygon formation has been completed and an attempt is being made to label the polygons from the left/right codes of the input segments.

- **Unable to find polygon label in AC texts in feature with FSN 'integer' - polygon abandoned**

  Although all segments were checked for the presence of left and right codes at input, the required left/right code can no longer be found.

- **Polygon labels in AC texts inconsistent in feature with FSN 'integer' - polygon abandoned**

  A polygon boundary that includes the specified segment does not have consistent labelling as derived from the segment left/right codes.

Identifier Failure

This error occurs during the output of coded segments.

- o **Unable to generate consistent identifiers for segment with FSN 'integer'**

  The inability to find a consistent choice of identifiers may reflect a corrupt junction structure. If the problem cannot be resolved then an SPR should be submitted.

EXAMPLES

This example shows IPOLYGON being run on the dataset shown in Fig 1. The /ONEARM=USE combination indicates that all segments are to be used. The /OPTIONS=NEST is present by default. There is no labelling mechanism specified. Two IFF output options have been selected; polygon boundary output and coded segment output. The output segments are coded with the relevant polygon identifiers. The output IFF file specifications have been derived from the input IFF segment file together with user supplied extensions.

All 18 of the input segments are output to the coded segments file. These formed 7 polygons for which polygon boundary features were constructed.

The /LOG qualifier is present and results in the output of run time statistics to SYS$OUTPUT.

$STATUS is set to SS$_NORMAL - normal successful completion.

```
$
$IPOLYGON/ONEARM=USE/POLYGONS=OUTPUT:.POL -<CR>
_$ /SEGMENTS=(IDENT,OUTPUT:.SEG)/LOG TEST1<CR>
%LSLLIB-I-IFFOPENED, LSL$DATAROOT:[POLYGONS.ACCEPT]TEST1.IFJ;1 opened for read
%LSLLIB-I-IFFOPENED, LSL$DATAROOT:[POLYGONS.ACCEPT]TEST1.POL;1 opened for write
%LSLLIB-I-IFFOPENED, LSL$DATAROOT:[POLYGONS.ACCEPT]TEST1.SEG;1 opened for write

%POLY-I-POLVAL, Maximum number of polygons set to a default of 10000
%POLY-I-DEFSID, Average number of sides per polygons set to a default of 5


+-----------------------------------------------------------------+
|                                                                 |
|                  Building IFF Address Tables                    |
|                                                                 |
+-----------------------------------------------------------------+

Number of IFF segment feature addresses tabulated ... 18
Segment coordinate range is:
X-min ..............................................     0.000
X-max ..............................................   170.000
Y-min ..............................................     0.000
Y-max ..............................................   120.000



+-----------------------------------------------------------------+
|                                                                 |
|                        Forming Polygons                         |
|                                                                 |
+-----------------------------------------------------------------+

Number of polygons formed (including isolations) ... 9
Number of trees formed ............................. 1
Number of segments used to form polygons ........... 18
Minimum number of segments used to form a polygon .. 1
```

```
Maximum number of segments used to form a polygon .. 8
Direction of polygon formation ..................... ANTICLOCKWISE



+-----------------------------------------------------------------+
|                                                                 |
|             Setting up Spatial Index for Polygons               |
|                                                                 |
+-----------------------------------------------------------------+

Number of boxes in X direction....................... 3
Number of boxes in Y direction....................... 3
Mean number of boxes per polygon ....................     1.000
Mean number of polygons per box .....................     1.000



+-----------------------------------------------------------------+
|                                                                 |
|                 Identifying Isolated Polygons                   |
|                                                                 |
+-----------------------------------------------------------------+

Number of polygons examined ......................... 9
Number of isolated polygons ......................... 2



+-----------------------------------------------------------------+
|                                                                 |
|                  Identifying Nested Polygons                    |
|                                                                 |
+-----------------------------------------------------------------+

Number of polygons examined ......................... 9
Number of polygons containing nested polygons ....... 3
Number of polygons nested inside others ............ 11



+-----------------------------------------------------------------+
|                                                                 |
|                  Identifying Nested Linework                    |
|                                                                 |
+-----------------------------------------------------------------+

Number of trees located inside polygons ............ 1



+-----------------------------------------------------------------+
|                                                                 |
|                  Writing Coded Segments File                    |
|                                                                 |
+-----------------------------------------------------------------+
Number of segments output to /SEGMENTS file ........ 18



+-----------------------------------------------------------------+
|                                                                 |
|                 Writing Polygon Boundary File                   |
|                                                                 |
+-----------------------------------------------------------------+

Number of features output to /POLYGONS file ........ 7
 ELAPSED:    0 00:00:25.95  CPU: 0:00:03.00  BUFIO: 19  DIRIO: 225  FAULTS: 873
```

In this example a separate seed point file is supplied. All the features in this file are treated as seed points. The output polygon boundaries are **labelled** with a label derived from the seed point FSNs.

The input IFF segment file contains 89 segments which define 35 polygons. The 34 seed points are assigned to these in 3 passes. This indicates there is up to 2nd order nesting of polygons. NOTE that one of the polygons (the bounding polygon) does not have a seed point.

The /LOG qualifier is present and results in the output of run time statistics to SYS$OUTPUT.

$STATUS is set to SS$_NORMAL - normal successful completion.

```
$
$IPOLYGON/SEED=(USE:FSN,FILE:TEST2.SEED) -<CR>
_$ /POLYGONS=(LABEL,OUTPUT:.POL)/LOG TEST2<CR>
%LSLLIB-I-IFFOPENED, LSL$DATAROOT:[POLYGONS.ACCEPT]TEST2.IFJ;1 opened for read
%LSLLIB-I-IFFOPENED, LSL$DATAROOT:[POLYGONS.ACCEPT]TEST2.SEED;1 opened for read
%LSLLIB-I-IFFOPENED, LSL$DATAROOT:[POLYGONS.ACCEPT]TEST2.POL;1 opened for write


%POLY-I-POLVAL, Maximum number of polygons set to  10000
%POLY-I-DEFSID, Average number of sides per polygons set to a default of     5


+-----------------------------------------------------------------+
|                                                                 |
|                  Building IFF Address Tables                    |
|                                                                 |
+-----------------------------------------------------------------+

Number of IFF segment feature addresses tabulated ... 89
Segment coordinate range is:
X-min ..............................................     0.000
X-max ..............................................  1000.000
Y-min ..............................................     0.000
Y-max ..............................................  1000.000



+-----------------------------------------------------------------+
|                                                                 |
|                     Checking Seed Points                        |
|                                                                 |
+-----------------------------------------------------------------+

Number of seed point feature addresses tabulated ... 34


+-----------------------------------------------------------------+
|                                                                 |
|                       Forming Polygons                          |
|                                                                 |
+-----------------------------------------------------------------+

Number of polygons formed (including isolations) ... 40
Number of segments used to form polygons ........... 89
Minimum number of segments used to form a polygon .. 1
Maximum number of segments used to form a polygon .. 24
Direction of polygon formation ..................... ANTICLOCKWISE


+-----------------------------------------------------------------+
|                                                                 |
|             Setting up Spatial Index for Polygons               |
|                                                                 |
+-----------------------------------------------------------------+

Number of boxes in X direction....................... 3
Number of boxes in Y direction....................... 3
Mean number of boxes per polygon ...................      0.225
Mean number of polygons per box ....................      4.444


+-----------------------------------------------------------------+
|                                                                 |
|            Setting up Spatial Index for Seed Points             |
|                                                                 |
+-----------------------------------------------------------------+

Number of boxes in X direction....................... 3
Number of boxes in Y direction....................... 3
Mean number of seed points per box .................      3.778


+-----------------------------------------------------------------+
|                                                                 |
|                 Identifying Isolated Polygons                   |
|                                                                 |
+-----------------------------------------------------------------+

Number of polygons examined ......................... 40
Number of isolated polygons ......................... 5
```

```
+----------------------------------------------------------------+
|                                                                |
|                  Identifying Nested Polygons                   |
|                                                                |
+----------------------------------------------------------------+

Number of polygons examined ......................... 40
Number of polygons containing nested polygons ....... 3
Number of polygons nested inside others ............. 15



+----------------------------------------------------------------+
|                                                                |
|                     Assigning Seed Points                      |
|                                                                |
+----------------------------------------------------------------+

Number of seed points assigned ...................... 34
Number of passes required to resolve nesting ........ 3



+----------------------------------------------------------------+
|                                                                |
|                 Writing Polygon Boundary File                  |
|                                                                |
+----------------------------------------------------------------+

Number of features output to /POLYGONS file ........ 35
 ELAPSED:    0 00:00:19.75  CPU: 0:00:04.00  BUFIO: 22  DIRIO: 187  FAULTS: 763
```

The next example uses the dataset shown in Fig 1. The /LIST qualifier is used to direct the runtime output to the file SYS$DISK:[]IPOLYGON.LIS;0. An IFF file LSL$IF:TEST1.PIP;0 is created and symbol features (with FC 24) are written into layer 2.

```
$ IPOLYGON/LIST/PIP=(FC:24,LAYER:2)/ONEARM=WARN TEST1<CR>
%POLY-W-BADJUN, 1 arm junction detected at (   15.000,   45.000)
%POLY-W-BADJUN, 1 arm junction detected at (   50.000,   55.000)
%POLY-W-BADJUN, 1 arm junction detected at (   60.000,   30.000)
%POLY-W-BADJUN, 1 arm junction detected at (   80.000,   10.000)
%POLY-W-BADJUN, 1 arm junction detected at (  135.000,   50.000)
%POLY-W-BADJUN, 1 arm junction detected at (   65.000,   85.000)
%POLY-W-BADJUN, 1 arm junction detected at (  110.000,   90.000)
 ELAPSED:    0 00:00:09.76  CPU: 0:00:02.69  BUFIO: 15  DIRIO: 149  FAULTS: 997
```

The presence of the /ONEARM=WARN combination means that the one-armed junctions have generated warnings. The contents of the /LIST file are given below:

```
============================= I P O L Y G O N =================================

IPOLYGON invoked by BUREAU using terminal LTA314: at 08-SEP-1988 15:58:30.99

Command line:

$ IPOLYGON/LIST/ONEARM=WARN/PIP=(FC:24,LAYER:2) TEST1

===============================================================================



%POLY-I-POLVAL, Maximum number of polygons set to a default of 40000
%POLY-I-DEFSID, Average number of sides per polygons set to a default of 5



+-------------------------------------------------------------+
|                                                             |
|                  Building IFF Address Tables                |
|                                                             |
+-------------------------------------------------------------+

1 arm junction detected at (   15.000,    45.000)
1 arm junction detected at (   50.000,    55.000)
1 arm junction detected at (   60.000,    30.000)
1 arm junction detected at (   80.000,    10.000)
1 arm junction detected at (  135.000,    50.000)
1 arm junction detected at (   65.000,    85.000)
1 arm junction detected at (  110.000,    90.000)
Number of IFF segment feature addresses tabulated ... 18
Segment coordinate range is:
X-min ...............................................     0.000
X-max ...............................................   170.000
Y-min ...............................................     0.000
Y-max ...............................................   120.000
```

```
+----------------------------------------------------------------+
|                                                                |
|                        Forming Polygons                        |
|                                                                |
+----------------------------------------------------------------+

Number of polygons formed (including isolations) ... 11
Number of segments used to form polygons ........... 18
Minimum number of segments used to form a polygon .. 1
Maximum number of segments used to form a polygon .. 2
Direction of polygon formation ..................... ANTICLOCKWISE



+----------------------------------------------------------------+
|                                                                |
|             Setting up Spatial Index for Polygons              |
|                                                                |
+----------------------------------------------------------------+

Number of boxes in X direction....................... 3
Number of boxes in Y direction....................... 3
Mean number of boxes per polygon ....................     0.818
Mean number of polygons per box .....................     1.222



+----------------------------------------------------------------+
|                                                                |
|                 Identifying Isolated Polygons                  |
|                                                                |
+----------------------------------------------------------------+

Number of polygons examined ......................... 11
Number of isolated polygons ......................... 4



+----------------------------------------------------------------+
|                                                                |
|                  Identifying Nested Polygons                   |
|                                                                |
+----------------------------------------------------------------+

Number of polygons examined ......................... 11
Number of polygons containing nested polygons ....... 5
Number of polygons nested inside others ............. 16



+----------------------------------------------------------------+
|                                                                |
|                 Writing Point-in-Polygon File                  |
|                                                                |
+----------------------------------------------------------------+
Number of points output to /PIP file ............... 6
```

The last example uses the dataset shown in Fig. 8(a to d) to illustrate the application of AC propagation to a simple polygon overlay operation. Figures 8a and 8b show two divisions of the same area, into areas of differing soil type and vegetation type. The soil types (A, B and C) are shown in Fig. 8a, and the vegetation types (P, Q and R) in Fig. 8b.

A merged file is created (Fig. 8c) containing the soil area boundaries, the vegetation area boundaries, and a bounding rectangle. The soil and vegetation type information is held in left-right AC types 4 and 5, the string components of these ACs having the form "SOIL: A" or "VEGETATION: P". The contents of the ACs are indicated in Fig. 8c; note that The bounding rectangle has been assigned ACs to indicate that the external area has unknown soil and vegetation types.

This merged file is processed using the ILINK module of the STRUCTURE package, first to break all features at intersection points, and secondly to set up a junction structure. The result of this operation is shown in Fig. 8d, where each feature is labelled with its FSN. Each new feature inherits the left-right codes of its parent feature.

The division of the area into soil polygons and vegetation polygons constitutes two coverages, and the text prefix mechanism is used to describe these in a coverage file, as follows:

```
! Cover left right in-prefix  out-prefix  left right contain polygon
seed
!  no    AC   AC                            AC    AC    AC      AC
AC
    1    4     5 "SOIL:"       "SOIL:"       4     5    12      82
82
    2    4     5 "VEGETATION:" "VEGETATION:" 4     5    12      82
82
```

Figure 8a - Soil areas

Figure 8b - Vegetation areas

Figure 8c - Merged file with ACs

Figure 8d - FSNs after structuring

The two left-right codings may be propagated using full propagation, as each boundary in each coverage is coded on at least one side. To create the intersection polygons and attach appropriate ACs, the /POLYGONS=LABEL qualifier would be used. The following command file would perform the initial structuring and run IPOLYGON to create labelled intersection polygons:

```
$ILINK/BREAK/BPF=10 Z.IFF Z.BRE
$!
$ILINK/STRUCTURE Z.BRE Z.IFJ
$!
$IPOLYGON/LOG/PROPAGATE=(FULL,COVERAGEFILE:OVERLAYTEST)-
             /POLYGONS=(OUTPUT:Z.POL,LABEL) Z.IFJ
```

The output below is created using the /ASCII=LABEL qualifier, and describes each polygon as a sequence of signed FSNs. Each polygon is assigned two ACs of type 82, one containing soil type and the other containing vegetation type.

```
============================== I P O L Y G O N ================================

IPOLYGON invoked by SIMON using terminal LTA5: at 23-MAR-1990 14:10:55.45

Command line:

$ IPOLYGON/LOG/PROPAGATE=(FULL,COVERAGEFILE:OVERLAYTEST)/ASCII=LABEL Z.IFJ

===============================================================================



%POLY-I-POLVAL, Maximum number of polygons set to a default of 40000
%POLY-I-DEFSID, Average number of sides per polygons set to a default of 5



+-----------------------------------------------------------------+
|                                                                 |
|                  Building IFF Address Tables                    |
|                                                                 |
+-----------------------------------------------------------------+

Number of IFF segment feature addresses tabulated ... 22
Segment coordinate range is:
X-min .............................................  1500.000
X-max .............................................  3000.000
Y-min .............................................  2000.000
Y-max .............................................  3500.000



+-----------------------------------------------------------------+
|                                                                 |
|                        Forming Polygons                         |
|                                                                 |
+----------------------------------------------------------------+

Number of polygons formed (including isolations) ... 12
Number of segments used to form polygons ........... 22
Minimum number of segments used to form a polygon .. 1
Maximum number of segments used to form a polygon .. 9
Direction of polygon formation ..................... ANTICLOCKWISE



+----------------------------------------------------------------+
|                                                                |
|             Setting up Spatial Index for Polygons              |
|                                                                |
+----------------------------------------------------------------+

Number of boxes in X direction....................... 3
Number of boxes in Y direction....................... 3
Mean number of boxes per polygon ....................     0.750
Mean number of polygons per box .....................     1.333



+----------------------------------------------------------------+
|                                                                |
|                 Identifying Isolated Polygons                  |
|                                                                |
+----------------------------------------------------------------+

Number of polygons examined ......................... 12
Number of isolated polygons ......................... 1



+----------------------------------------------------------------+
|                                                                |
|                  Identifying Nested Polygons                   |
|                                                                |
+----------------------------------------------------------------+

Number of polygons examined ......................... 12
Number of polygons containing nested polygons ....... 1
Number of polygons nested inside others ............. 2



+----------------------------------------------------------------+
|                                                                |
|             Creating Polygon Component Arrays                  |
|                                                                |
+----------------------------------------------------------------+



+----------------------------------------------------------------+
|                                                                |
|                 Associating segments for each                 |
|                                                               |
+---------------------------------------------------------------+



+---------------------------------------------------------------+
|                                                               |
|                   Assigning Propagated Codes                  |
|                                                               |
+---------------------------------------------------------------+

Number of polygons with consistent labelling ........ 11


+---------------------------------------------------------------+
|                                                               |
|             Summary Listing of Polygon Components             |
|                                                               |
+---------------------------------------------------------------+

BOUNDING POLYGON LABEL: "SOIL: UNKNOWN"
                        "VEGETATION: UNKNOWN"
SEGMENTS:  -14 -22 -21 -20 -19 -18 -17 -16 -15

POLYGON LABEL: "SOIL: A"
               "VEGETATION: P"
SEGMENTS:  1 -8 21

POLYGON LABEL: "SOIL: B"
               "VEGETATION: P"
SEGMENTS:  -1 22 14 4 -9

POLYGON LABEL: "SOIL: A"
               "VEGETATION: Q"
SEGMENTS:  2 -11 20 8

POLYGON LABEL: "SOIL: B"
               "VEGETATION: Q"
SEGMENTS:  -2 9 5 -12

POLYGON LABEL: "SOIL: A"
               "VEGETATION: R"
SEGMENTS:  3 19 11

POLYGON LABEL: "SOIL: B"
               "VEGETATION: R"
SEGMENTS:  -3 12 6 18

POLYGON LABEL: "SOIL: C"
               "VEGETATION: P"
SEGMENTS:  -4 15 -10

POLYGON LABEL: "SOIL: C"
               "VEGETATION: Q"
```

```
SEGMENTS:  -5 10 16 -13 ( 7 )

POLYGON LABEL: "SOIL: C"
               "VEGETATION: R"
SEGMENTS:  -6 13 17

POLYGON LABEL: "SOIL: B"
               "VEGETATION: Q"
SEGMENTS:  -7

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


No. of POLYGONS output to /OPTIONS=LIST file ....... 11
```

MESSAGES (INFORMATIONAL)

These messages give information only, and require no immediate action by the user. They are used to provide information on the current state of the program, or to supply explanatory information in support of a warning or error message.

LITESOPNOUT, /LITES2 command file 'file-spec' opened for output

Explanation: The specified LITES2 command file has been opened successfully.

User action: None.

LSTOPNOUT, /OUTPUT list file 'file-spec' opened for output

Explanation: The specified listing file has been opened successfully.

User action: None.

MESSAGES (WARNING)

These messages are output when an error has occurred that can be corrected immediately by the user or that the program will attempt to overcome.

ACINCON, polygon labels in AC texts inconsistent in'coverage'feature with FSN 'integer'%S

Explanation: Labelling has been enabled using the coded input segments. A polygon, which has the segment with the specified FSN as part of its boundary, does not have consistent labelling. Two different labels from segments in the current polygon are output after this message.

User action: Examine segments in the vicinity of the specified segment in the input IFF segment file. Use LITES2 to change the offending AC text entries.

ACMISS, Missing AC text polygon label'coverage' feature with FSN 'integer''status'

Explanation: The label on the coded input segment specified has been lost. This is despite the check for the existence of labels at an earlier stage. The polygon will be kept if the KEEP option was specified.

User action: When performing the propagation operation, this message is expected and often occurs where AC labels are incomplete with respect to one or more coverage. It ought not to happen during other operations as should be trapped before. In this case you should contact Laser-Scan for advice with precise details of the circumstances.

ACTRUNC, AC text truncated during concatenation

Explanation: The maximum length of an AC text would be exceeded by concatenating left-right codes or identifiers. Partial concatenation is performed.

User action: Avoid the use of /PROPAGATE=CONCATENATE when long AC texts are present in left-right codes or area identifiers

BADORDER, seed point FC out of order at line 'integer' in FC pair file 'file-spec'

Explanation: The first column in the FCP file contains a list of seed point FCs. These must be given in ascending order. The FC on the specified line is out of order.

User action: Edit the FCP file so that the seed point FCs are arranged in ascending order.

MULTISEG, segment with FSN 'integer' used to form multiple polygons

Explanation: One direction of the specified segment has been used to form more than one polygon.

User action: Check formed polygons in the vicinity of the sepcified segment for errors. This may indicate a corrupt junction structure.

MESSAGES (ERROR)

These messages indicate an error in processing which will cause the program to terminate. The most likely causes are a corrupt or otherwise invalid input file, or an error related to command line processing and file manipulation.

BADCOV, unable to read AC information on line 'integer' of coverage file 'file-spec'

Explanation: IPOLYGON was unable to read or decode the specified line of the file describing AC pairs for attribute propagation.

User action: Edit the ASCII file to correct the offending line.

BADEO, EO entry missing or misplaced in 'file-spec'

Explanation: IPOLYGON has failed to find an EO (End layer) entry where expected in the specified file. The EO entry is either missing or the address of the EO entry is incorrectly stored in the corresponding NO (New layer) entry in the file.

User action: See the IFF User Guide ("Layer Level Entries" section) for details of how to repair incorrect EO addresses in IFF NO entry.

BADPAIR, unable to read FC pair on line 'integer' of FC pair file 'file-spec'

Explanation: IPOLYGON was unable to decode one of the FCs on the specified line of the file specified by /SEED=FCP. The FC should be integers.

User action: Edit the ASCII FCP file to correct the offending line.

CLCOV, error closing coverage file 'file-spec'

Explanation: IPOLYGON has failed to close the input ASCII coverage file specified by the /PROPAGATE=COVERAGEFILE qualifier.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

CLLIST, error closing listing file 'file-spec'

Explanation: IPOLYGON has failed to close the output ASCII segment listing file specified by the /LIST qualifier.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

CLLITES, error closing LITES2 command file 'file-spec'

Explanation: IPOLYGON has failed to close the LITES2 command file selected with the /LITES2 qualifier.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

CLPAIR, error closing FC pair file 'file-spec'

Explanation: IPOLYGON has failed to close the input ASCII FCP file specified by the /SEED=FCP qualifier.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

CLPIPIFF, error closing output point-in-polygon IFF file 'file-spec'

Explanation: IPOLYGON has failed to close the output IFF point-in-polygon file specified by the /PIP=OUTPUT qualifier.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

CLPOLIFF, error closing output polygon boundary IFF file 'file-spec'

Explanation: IPOLYGON has failed to close the output IFF polygon boundary file specified by the /POLYGONS=OUTPUT qualifier.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

CLSEEDIFF, error closing input seed points IFF file 'file-spec'

Explanation: IPOLYGON has failed to close the input IFF seed point file specified by the /SEED=FILE qualifier.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

CLSEGIFF, error closing output coded segments IFF file 'file-spec'

Explanation: IPOLYGON has failed to close the output IFF coded segment file specified by the /SEGMENTS=OUTPUT qualifier.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

CLSRCIFF, error closing input segments IFF file 'file-spec'

Explanation: IPOLYGON has failed to close the input IFF segment file.

User action: The cause of failure to close this file will be indicated by the supplementary messages output following this error.

FAILTRLG, failed to translate logical name 'logical-name'

Explanation: IPOLYGON is unable to translate the specified logical name.

User action: Use the VMS ASSIGN or DEFINE commands to correctly define the logical name. Re-run IPOLYGON. Normally the logical names required by IPOLYGON are defined at user login time. See the "LAMPS Environment Guide" for details of these logical names.

INVALAC, 'integer' is an invalid ancillary code argument - value must lie in range 1 - 32767

Explanation: The specified ancillary code lies outside the indicated range for valid IFF ancillary codes.

User action: Respecify the IPOLYGON command line ensuring that any ancillary code specification lies in the correct range.

INVALFC, 'integer' is an invalid feature code argument - value must lie in range 0 - 32767

Explanation: The specified feature code lies outside the indicated range for valid IFF feature codes.

User action: Respecify the IPOLYGON command line ensuring that any feature code specification lies in the correct range.

INVALIT, 'integer' is an invalid iterate parameter - value must lie in the range 1 - 1000

Explanation: The specified value of the iterate parameter lies outside the indicated range.

User action: Respecify the IPOLYGON command line ensuring that the iterate parameter lies in the correct range.

INVALLAY, 'integer' is an invalid layer argument - value must lie in range 1 - 32767

Explanation: The specified layer lies outside the indicated range for valid IFF layer numbers.

User action: Respecify the IPOLYGON command line ensuring that any layer number specification lies in the correct range. (Layer 0 is reserved for registration marks and grid features).

INVCOVSEQ, coverage number out of sequence at line 'integer' in FC pair file 'file-spec'

Explanation: successive coverage numbers in the specified file must be equal or increase by 1

User action: edit or reorder the ASCII file

OPCOV, error opening coverage file 'file-spec' for input

Explanation: IPOLYGON is unable to open the file specified by the /PROPAGATE=COVERAGEFILE qualifier.

User action: The supplementary Laser-Scan, VMS system or RMS messages which are output in support of this message will facilitate diagnosis.

Possible causes for the error are:

- the file-spec was invalid
- the device, directory or file is read protected

OPLIST, error opening /OUTPUT list file 'file-spec' for output

Explanation: IPOLYGON is unable to open the listing file specified by the /LIST qualifier.

User action: The supplementary Laser-Scan, VMS system or RMS messages which are output in support of this message will facilitate diagnosis.

Possible causes for the error are:

- the file-spec was invalid
- the device, directory or file is write protected
- the device is full

OPLITES, error opening /LITES2 file 'file-spec' for output

Explanation: IPOLYGON is unable to open the LITES2 command file specified by the /LITES2 qualifier.

User action: The supplementary Laser-Scan, VMS system or RMS messages which are output in support of this message will facilitate diagnosis.

Possible causes for the error are:

- the file-spec was invalid
- the logical name LSL$LITES2CMD was incorrectly assigned to a non-existent device or directory

- o the device, directory or file is write protected

- o the device is full

OPPAIR, error opening /SEED=PAIR file 'file-spec' for input

Explanation: IPOLYGON is unable to open the FC pair specified by the /SEED=PAIR qualifier.

User action: The supplementary Laser-Scan, VMS system or RMS messages which are output in support of this message will facilitate diagnosis.

Possible causes for the error are:

- o the file-spec was invalid

- o the device, directory or file is read protected

SAMEFILE, segment and seed point files share the same specification - omit the /SEED=FILE:'filespec' keyword

Explanation: The input segment and seed point file specified with the /SEED=FILE keyword share the same specification. This will cause IPOLYGON to fail.

User action: If the seed points are in the same file as the segments omit the FILE:'file-spec' argument to the /SEED qualifier.

SEEDCHECK, previous warnings invalidate seed point data - aborting

Explanation: Seed point checks have been completed. IPOLYGON has detected errors in the seed point features that make further processing pointless.

User action: Use the warnings output by IPOLYGON and the LITES2 command file (if specified) to correct the seed point data using LITES2. When all the edits are complete re-run IPOLYGON using the corrected file.

TOOMNYCOV, too many coverages defined - maximum allowed is 'integer' - in file 'file-spec'

Explanation: AC propagation cannot operate on more than the specified number of coverages.

User action: Perform the propagation in two or more runs of IPOLYGON

TOOMNYFC, 'integer' feature code arguments specified - maximum allowed is 'integer'

Explanation: The user has specified more than the permitted maximum number of arguments to the /SEED=FC qualifier.

User action: Re-specify the IPOLYGON command line and ensure that the /SEED=FC qualifier is specified with less than the permitted maximum number of arguments. (Remember the value ranges of the form 'n:m' will be expanded by the IPOLYGON command decoder and may thus exceed the permitted maximum number of arguments.)

TOOMNYLAY, 'integer' layer arguments specified - maximum allowed is 'integer'

Explanation: The user has specified more than the permitted maximum number of arguments to the /SEED=LAYER qualifier.

User action: Re-specify the IPOLYGON command line and ensure that the /SEED=LAYER qualifier is specified with less than the permitted maximum number of arguments. (Remember the value ranges of the form 'n:m' will be expanded by the IPOLYGON command decoder and may thus exceed the permitted maximum number of arguments.)

TOOMNYPAIR, too many FC pairs read - maximum allowed is 'integer' - from FC pair file 'file-spec'

Explanation: There are too many FC pairs in the file specified by the /SEED=PAIR keyword.

User action: Edit the specified file so that it contains less FC pairs. Those polygon boundaries whose FC cannot be derived from a seed point will get the FC specified by /POLYGONS=FC:'integer' combination.

TOOMNYSEED, too many seed point features read - maximum allowed is 'integer'

Explanation: IPOLYGON can only process up to the specified number of polygons.

User action: Use LITES2 to split the seed point IFF file into two or more files. Ensure that there will be no more than the permitted number of seed points in any sub-file created from the original file. Re-run IPOLYGON.

TOOMNYTXT, too many AC prefixes read - maximum allowed is 'integer' - from file 'file-spec'

Explanation: Too many text prefixes are being used to describe a single coverage for AC propagation, in the file specified by the /PROPAGATE=COVERAGEFILE keyword.

User action: Edit the specified file so that it contains less prefixes. In order to be able to fully define the coverage, it may be necessary to recode some ACs.

UNEXPEOF, unexpected end of IFF file 'file-spec'

Explanation: The specified input IFF file terminated before an IFF EJ entry was encountered.

User action: Use IMEND to correctly terminate the file. If the segment file is in error, re-run ILINK on th repaired IFF file before using IPOLYGON. If the problem persists try reading the file into LITES2 and then exit from LITES2 thus creating a new version of the file. If necessary re-run ILINK followed by IPOLYGON.

MESSAGES (FATAL)

These messages indicate a severe error in processing, or some form of system failure, which has caused the program to terminate.

LOST, failed to locate IFF entry at recorded address - position lost

> **Explanation:** IPOLYGON has incorrectly stored the address of an entry within one of the input IFF files and has now attempted to locate that IFF entry. This is a very severe error. IPOLYGON is irrevocably lost.
>
> **User action:** Try reading the input IFF files into LITES2. If this is successfull then the problem lies within IPOLYGON itself; please make a copy of the input IFF files and report the problem to Laser-Scan.

UNEXPENTJP, unexpected entry 'entry' found while patching JP entries after NF 'integer'

> **Explanation:** IPOLYGON issues this message when unable to patch a JP entry for the /SEGMENTS=JUNCTIONS output option. This should never normally occur.
>
> **User action:** Please make a copy of the input IFF files and report the problem to Laser-Scan.

MESSAGES (OTHER)

In addition to the above messages which are generated by the program itself, other messages may be produced by the command line interpreter (CLI) and by Laser-Scan libraries. In particular, messages may be generated by the IFF library and by the Laser-Scan I/O library, LSLLIB. IFF library messages are introduced by '%IFF' and are documented in the IFF library users' guide. In most cases IFF errors will be due to a corrupt input file, and this should be the first area of investigation. If the cause of the error cannot be traced by the user, and Laser-Scan are consulted, then the output file should be preserved to facilitate diagnosis. LSLLIB messages are introduced by '%LSLLIB' and are generally self-explanatory. They are used to explain the details of program generated errors.

CHAPTER 3

MODULE ISTSEL

MODULE ISTSEL

REPLACES NONE

FUNCTION

ISTSEL is the Laser-Scan **I**FF **ST**ring **SEL**ection utility and forms part of the Laser-Scan POLYGONS Package. ISTSEL reads an IFF file and compares text strings held as AC (Ancillary Code) left/right codes and TX (TeXt) entries to keys given in a user specified lookup file.

If a match is found between the IFF text string and any of the keys in the lookup file then the left/right code is replaced with the matched key defined in the lookup file. After left/right code replacement, segments with identical left/right codes are identified and are not written to the output IFF file. This provides ISTSEL with the mechanism for segment selection and hence selective polygon aggregation.

FORMAT

```
$ ISTSEL input-file-spec output-file-spec
Command qualifiers                          Default
/DEF_FILE=file-spec                         See text.
[NO]/LOG                                    /NOLOG
```

PROMPTS

```
_Input data file:  input-file-spec

_Output data file:  output-file-spec
```

PARAMETERS

input-file-spec

- specifies the IFF file which will be processed. Any part of the file specification that is not supplied will be taken from the default specification 'LSL$IF:IFF.IFF;0'.

output-file-spec

- specifies the IFF file to be created. Any part of the output file specification that is not provided will be taken from the parsed input specification. Note that a version number must **not** be specified for the output file. If a file with the specified name already exists, a new file will be created with the version number incremented by one.

COMMAND QUALIFIERS

/DEF_FILE=file-spec

- defines the lookup file which contains the code definitions supplied by the user for the output IFF file. If /DEF_FILE is not specified or if /DEF_FILE is specified with no argument the default used is SYS$DISK:[]CODE_DEF.DAT;0.

/LOG
/NOLOG (default)

- this will result in supplementary messages being sent to SYS$OUTPUT. Supplementary messages are generated when a file is successfully opened, and a messages indicating the progress of ISTSEL processing are also output.

DESCRIPTION

General

ISTSEL allows polygon segment selection to be performed on an IFF file which contains data with left/right AC codes (AC types 4 and 5). This enables the user to group together polygons of a given code under a new code in the output IFF file.

The selection is based on information supplied by the user in a lookup file which defines the new code groupings. Each new left/right code (keyword) can be up to 255 characters in length. Up to 100 new codes may be specified. Each new code may be defined to replace up to 100 old codes.

ISTSEL does not require IFF junction entries to be present in the input IFF file, only that all links should have left/right AC codes. If junction entries are present they currently are not transferred to the output file. The STRUCTURE utility ILINK/STRUCTURE should be run on the ISTSEL output file if IFF junction structure is required for later processing.

Program operation

ISTSEL operation can be defined as follows.

- The user specified code definitions lookup file is opened and read. ISTSEL expects a new keyword followed by a list of old codes to be grouped under the new heading.

- The input IFF file is scanned for the text strings associated with AC types 4 or 5, and TX entries. When a text string is located it is compared with the list of old codes defined under each heading in the text file. If a match is found then the old code is replaced by the keyword of the group in which it was found in the lookup file. Leading and trailing spaces and nulls are ignored in this comparison.

- If a substitution has been made for an AC entry then the left and right codes (types 4 and 5) are compared. If they are identical then they will not be written to the output IFF file.

- All other IFF entries including all altered TX entries are copied to the output IFF file.

- All opened files are closed down and the program terminates setting $STATUS to SS$_NORMAL if the run was successful.

The Code Definition File

ISTSEL reads a user defined lookup file to get the information required for the segment selection process. A default code definition file CODE_DEF.DAT is supplied with the polygon package and should be present in the directory defined by the logical name LSL$LOOKUP. A copy of the example file is given below.

```
!       Example code definition file for POLYGONS utility ISTSEL
!
!       Copyright 1987 Laser-Scan Ltd, Cambridge, England
!
!       Author: G S Tobiss                                      17-Feb-1987
!
!       File format is:
!
!       %New_code
!       Old_code_1 [, Old_code_n ... ]
!
!       Where:
!
!       !         -  is a comment. Text after a "!" is ignored.
!       %         -  indicates that the following text is to be read as a new
!                    segment code
!       New_code -  is the new segment code
!       Old_code -  is [one of a list of] old segment codes to be replaced
!                    by the new code. Old segment codes must be separated by
!                    commas .
!
! For example:
!
%LOW_STRAW_INCORPORATION_POTENTIAL
P123,P67a,P23/u
!
%MEDIUM_STRAW_INCORPORATION_POTENTIAL
```

```
P34, 901, River_terrace_material
!
%MEDIUM_STRAW_INCORPORATION_POTENTIAL
P67c,P98,P6Ab
!
```

An exclamation mark '!' indicates a comment or comment line and is ignored by ISTSEL when reading from the code definition file. The new soil codes must be preceded by a percent '%' character. The old codes must be separated by commas. The limit on the number of old codes in each group is set at 100. Although IFF TX and AC entries can hold up to 255 characters, the number of characters allowed in a new or old code is set at 80. Duplication of old codes between the different new code groups is detected, a warning identifying the suspect code is given. Processing is abandoned after the whole file has been checked for duplication. Duplication of codes within a single new code group is permitted.

Using the example lookup file given above, all segments having the left or right code 'P34', '901', and 'River_terrace_material' will be recoded with with the new code 'MEDIUM_STRAW_INCORPORATION_POTENTIAL'. Segments which have this new code on both left and right sides will be omitted from the output IFF file and thus polygons formed by the segments having the 3 original codes 'P34', '901', and 'River_terrace_material' will be merged. The new polygon now formed by the merged original polygons will have the code 'MEDIUM_STRAW_INCORPORATION_POTENTIAL'.

EXAMPLES

```
$ ISTSEL/DEF_FILE=POLDEF.DAT/LOG<CR>
_Input IFF file: DUA3:[BUREAU.SOILS]OLDCODES.IFF;7<CR>
_Output IFF file: GLEYS<CR>
%ISTSEL-I-DEFOPENED, /DEFCODE file SYS$DISK:[]POLDEF.DAT;0 opened for read
%LSLLIB-I-IFFOPENED, DUA3:[BUREAU.SOILS]OLDCODES.IFF;7 opened for read
%LSLLIB-I-IFFOPENED, DUA3:[BUREAU.SOILS]OLDCODES.IFF;8 opened for write
Number of left/right coded links processed = 129
Number of left/right coded links deleted = 23
 ELAPSED: 00:02:01.10  CPU: 0:00:57.15  BUFIO:8250  DIRIO: 3230  FAULTS: 1037
$
```

In this run of ISTSEL the user has supplied a specific code definition lookup file-spec using the /DEF_CODES qualifier. As only the filename and extension for the lookup file have been specified, the missing device and directory are taken from the default 'SYS$DISK:[]' (i.e. user's current default set using the VMS SET DEFAULT or the SD command) and the version number defaults to ';0', i.e. the latest version of the file.

The /LOG qualifier has also been selected. The user has then pressed carriage return after which he has received a prompt for the input file. In response to this prompt he has supplied a complete specification for the input file. The user has again pressed carriage return and has been prompted for the output file-spec. The missing parts of the output file specification have been taken from the input file-spec.

Notice that as a result of specifying the /LOG qualifier, messages have been output indicating the successful opening of the code definition lookup file and the input and output IFF files. At the end of the run ISTSEL has output a summary of the number of left/right coded links processed and how many have been omitted from the output file. The run completed successfully. $STATUS is set to SS$_NORMAL - normal successful completion.

```
$ ISTSEL/LOG<CR>
_Input IFF file: DUA3:[BUREAU.SOILS]OLDCODES.IFF;7<CR>
_Output IFF file: GLEYS<CR>
%ISTSEL-E-OPDEF, error opening code definition file SYS$DISK:[]CODE_DEF.DAT;0
-LSLLIB-E-NOSUCHFILE, file cannot be found
 ELAPSED: 00:02:01.10  CPU: 0:00:57.15  BUFIO:8250  DIRIO: 3230  FAULTS: 1037
$
```

In this run of ISTSEL the user has pressed carriage return after the ISTSEL command and has received a prompt for input. In response to the prompt he has supplied a complete specification for the input file. The missing parts of the output file specification have been taken from the input file-spec. The default definition file SYS$DISK:[]CODE_DEF.DAT is assumed (i.e. that it is in the current default directory), since no alternative has been specified with the

/DEF_FILE qualifier. No lookup file of that name exists, however, and ISTSEL aborts execution. STATUS is set to SS$_ABORT.

The user must ensure that a lookup file having the default specification is available, (or else explicitly supply a lookup file-spec with the /DEF_FILE qualifier), before attempting to re-run ISTSEL.

```
$ ISTSEL POLFILE NEWFILE<CR>
 ELAPSED: 00:01:10.20  CPU: 0:00:37.15  BUFIO:6250  DIRIO: 2230  FAULTS: 797
$
```

In this run of ISTSEL the user has only specified a filename for the input and output file-specs. The missing parts of the file specifications have been taken from the default file-spec 'LSL$IF:IFF.IFF;0'. The default definition file SYS$DISK:[]CODE_DEF.DAT;0 is assumed (i.e. that it is in the current default directory), since no alternative has been specified with the /DEF_FILE qualifier. The run completed successfully. $STATUS is set to SS$_NORMAL - normal successful completion.

MESSAGES (INFORMATIONAL)

These messages give information only, and require no immediate action by the user. They are often used to supply explanatory information in support of a warning or error message.

DEFOPENED, /DEFCODE file 'file-spec' opened for read

> **Explanation:** The specified (or default) code definition lookup file has been opened successfully.
>
> **User action:** Check that the file specified is the one that was intended for use.

MESSAGES (WARNING)

These messages are output when an error has occurred which the user can correct immediately.

ERRAC, Only one left/right AC detected in feature with FSN 'integer' ('integer')

Explanation: The feature identified has has either a type 4 (left hand code) or type 5 (right hand code) AC (Ancillary Code) missing. Processing will continue and the feature will be copied to the output file.

User action: Use LITES2 to investigate the feature in question and add the relevant AC.

MESSAGES (ERROR)

These messages indicate an error in processing which has caused the program to terminate. The most likely causes are a corrupt or otherwise invalid input IFF file, or an error related to command line processing and file manipulation. As it is most unlikely that any output file produced will be correctly processed, the output file will normally be deleted.

DUPCODES, duplicate code "code" detected at line 'integer' of code definition file 'file-spec'

> **Explanation:** Duplicate codes have been detected in the definition file, supplementary messages will identify the suspect codes. Processing will continue until the whole definition file has been checked.
>
> **User action:** Edit the suspect codes out of the definition file

FOR, error in code definition file format at line 'integer of 'file-spec'

> **Explanation:** A list of old codes was detected before a new code definition was found.
>
> **User action:** Check the format of the code definition file.

MAXCODEEX, maximum number of new codes exceeded at line 'integer' of code definition file - maximum allowed is 'integer'

MAXCODEEX, maximum number of old codes exceeded at line 'integer' of code definition file - maximum allowed is 'integer'

> **Explanation:** The maximum number of old or new left/right codes currently allowed has been exceeded.
>
> **User action:** Ensure that the code definition file does not contain more than the specified number of old or new codes. If it does then use VMS EDIT (or similar) to split the file into two (or more) new files. Re-run ISTSEL once for each code definition file, thus performing the segment selection process in two stages.

NODATA, no data found in code definition file 'file-spec'

Explanation: No data was read before the end of the code definitions file was found.

User action: Check the content of the code definition file.

OPDEF, error opening code definition file 'file-spec'

Explanation: ISTSEL is unable to open the code definition file specified by the /DEF_CODE qualifier argument (or default).

User action: The supplementary Laser-Scan, VMS system or RMS messages which are output in support of this message will facilitate diagnosis.

Possible causes for the error are:

- the file-spec was invalid
- the device, directory or file is protected against read access.

READEF, error reading line 'integer' of code definition file 'file-spec'

Explanation: An error has occurred while reading the specified code definition file. The supplementary error messages should give more information on the error.

User action: Check that none of the codes in the code definition file are longer than 255 characters. Use the information supplied in the supplementary messages to diagnose the fault, then use the VMS EDIT (or similar) editor to correct the code definitions file. Re-run ISTSEL.

UNEXPEOF, unexpected end of IFF input file

Explanation: The input IFF file terminated before an IFF EJ entry was encountered.

User action: Use IMEND to correctly terminate the file.

MESSAGES (OTHER)

In addition to the above messages which are generated by ISTSEL itself, other messages may be produced by the command line interpreter (CLI) and by Laser-Scan libraries. In particular, messages may be generated by the IFF library. These are introduced by '%IFF' and are documented in the IFF library users' guide. If the cause of the error cannot be traced by the user and Laser-Scan are consulted, then the output file should be preserved to facilitate diagnosis. Generally, however, ISTSEL will attempt to delete any corrupt output file before program termination.

CHAPTER 4

MODULE POLMERGE

MODULE POLMERGE

FUNCTION

POLMERGE is the Laser-Scan automatic IFF **POL**YGON **MERG**ing and elimination utility. It forms part of the Laser-Scan POLYGONS Package.

POLMERGE is designed to be run in batch mode and all options may be specified on the command line. No user interaction is required during processing.

POLMERGE operates on an IFF junction structured (IFJ) file containing left/right coded links. Because output is also to an IFJ file the program may be used as a pre-processor before polygon creation using IPOLYGON.

Polygon merging takes place by means of user-defined rules. Both a look up table and command line qualifiers may be incorporated in their definition.

FORMAT

```
$ POLMERGE in-file-spec out-file-spec
```

Command qualifiers	**Defaults**
/[NO]ACCHECK	/ACCHECK
/[NO]AREA=(tolerance-spec[...])	See text
/[NO]BOUNDING	/NOBOUNDING
/[NO]ELIMINATE(elimination-option)	See text
/[NO]LIST[='file-spec']	/NOLIST
/[NO]LOG	/NOLOG
/LOOKUP = file-spec	No lookup file used.
/[NO]MERGE = (merging-option)	See text
/[NO]RATIO=(tolerance-spec[...])	/NORATIO

PROMPTS

```
_Input-IFJ-file:       input-file-spec
_Merged-IFJ-file:      output-file-spec
```

PARAMETERS

input-file-spec

- specifies the junction structured IFF file to be processed. Any part of the file specification which is not supplied will be taken from the default specification 'LSL$IF:IFF.IFJ'.

output-file-spec

- specifies the junction structured IFF file to be created. Any part of the file specification which is not explicitly given will be taken from the parsed input specification. Note that a version number must not be specified for the output IFJ file. If a file with the specified name already exists a new file will be created with the version number incremented by one.

COMMAND QUALIFIERS

/ACCHECK (default)
/NOACCHECK

- determines whether POLMERGE checks each polygon for consistent left right AC coding. In some cases this check is not necessary - eg. if the input file is derived from the other Laser-scan programs IPOLYGON or VECTORISE or if the merging mechanism used does not depend on the presence of such codes (eg. MERGE =(BY_SMALL).

  An error message is output for each inconsistency and program execution will be terminated.

/AREA =([[NO]MERGE_TOL:'real'],[[NO]ELIM_TOL:'real']) (default)
/NOAREA

- determines whether polygon area is taken into account in polygon merge and elimination operations. Only polygons which have area less than or equal to the supplied tolerances are eliminated or merged.

  Both the tolerances below which isolated polygons are eliminated and below which a polygon is merged into its neighbour may be specified. If only one is specified then the other defaults to the same value.

  The default area tolerances are calculated arbitrarily from the range entry in the IFJ file:-

  ```
  TOLERANCE = (XMAX-XMIN)*(YMAX-YMIN)/100
  ```

  If /NOAREA is specified then polygon merging and elimination takes place regardless of polygon area. The two tolerances may also be negated individually. This allows area of polygons to be taken into account for elimination, but not merging - or vice versa.

/BOUNDING
/NOBOUNDING (default)

- determines whether those polygons which have a common link with the bounding polygon are merged. If /BOUNDING is specified then such polygons are preserved in the output file.

  This option is useful if files processed by POLMERGE are to be merged with adjacent files using the IMP utility IMERGE. Polygons which straddle the boundary between map sheets will be preserved.

/ELIMINATE = ([BY_LOOKUP],[BY_UPPER],[BY_LOWER]) (default)
/NOELIMINATE

- specifies that isolated polygons are to be removed from the data. The basis of the elimination is specified by one of three mutually exclusive keywords.

  - BY_LOOKUP - A polygon is eliminated on the basis of coding rules specified in a lookup file.
  - BY_UPPER - A polygon is eliminated only if the surrounding polygon has a higher numeric code.
  - BY_LOWER - A polygon is eliminated only if the surrounding polygon has a lower numeric code.

  If /ELIMINATE is specified with no keyword arguments (or is not supplied at all) then the default operation is to eliminate all isolated polygons according to the current /AREA and /RATIO tolerances. If /LOOKUP is specified then isolated polygons are eliminated on the basis of primary codes in the lookup table (see Lookup File section).

  If /NOELIMINATE is specified, then all isolated polygons are retained in the output file.

/LIST[=file-spec]
/NOLIST (default)

- by default, message and diagnostic output will be to SYS$OUTPUT. This option allows the user to redirect output to the specified text file. If the optional file-spec argument is omitted, POLMERGE directs output to a file named SYS$DISK:[]POLMERGE.LIS. If the user supplies only a partial file-spec, the missing file specification components are taken from the default specification SYS$DISK:[]POLMERGE.LIS;0.

/LOG
/NOLOG (default)

- this will result in supplementary messages being sent to SYS$OUTPUT. Supplementary messages are generated when a file is successfully opened, and messages indicating the progress of POLMERGE processing are also output.

/LOOKUP = file-spec
/NOLOOKUP

- specifies a look-up file which is used to define rules for polygon merging. Any part of the file specification not supplied is taken from the default LSL$LOOKUP:LOOKUP.DAT.

  The lookup file is used to specify the action to be taken for specific left/right AC codes (defining polygon attributes) which are encountered in the file.

The description section contains details of the way the look-up file should should be constructed.

/MERGE= ([BY_LOOKUP],[BY_UPPER],[BY_LOWER],[BY_LARGE],[BY_SMALL] (default)
/NOMERGE

- specifies that polygons are to be merged and defines the basis of the merging process. The /MERGE qualifier takes 5 keyword arguments.

  - o BY_LOOKUP - Polygons to be merged on the basis of coding rules specified in a lookup file.

  - o BY_LARGE - A polygon is merged into the largest of the neighbouring polygons

  - o BY_SMALL - A polygon is merged into the smallest of the neighbouring polygons.

  - o BY_UPPER - A polygon is merged into the neighbouring polygon with a higher numeric code.

  - o BY_LOWER - A polygon is merged into the neighbouring polygon with a lower numeric code.

If /MERGE is specified with no keyword arguments (or is not supplied at all) then the default operation is to use BY_LARGER. If, however the /LOOKUP qualifier is also present then the default operation is BY_LOOKUP. Some combinations of keywords are not allowed (see restrictions).

Whether a polygon is considered for merging depends on the current /AREA and /RATIO tolerances. IF both /NORATIO and /NOAREA are specified, then merging /BY_LOOKUP is still possible. In this case polygons are merged purely on the basis of the coding rules set up in the look-up file.

If /NOMERGE is specified then non-isolated polygons are not merged into neighbouring polygons.

/RATIO = ([LARGER],[SMALLER],[[NO]MERGE_TOL:'real'],[[NO]ELIM_TOL:'real']
,[NO]AREA_TOL:'real'])
/NORATIO (default)

- causes polygons to be merged or eliminated on the basis of their perimeter - area ratio (see description section for ratio definition). If /RATIO = (LARGER) is specified, then any polygon which is larger than the supplied ratio tolerances is merged or eliminated. This is useful for eliminating long thin areas. This is the default mode if /RATIO is specified with neither the LARGER or SMALLER keyword arguments. If /RATIO = (SMALLER) is specified then any polygon which has a ratio tolerance smaller than the supplied tolerance is eliminated or merged. This is useful for preserving long thin areas.

The default tolerances are defined arbitrarily to be 8. See Figure 3 for a guide to ratio definition. If only one tolerance is supplied then the other is assumed to be the same.

The AREA_TOL keyword allows a separate area tolerance to be specified for the /RATIO test. In order to be merged or eliminated a polygon with ratio that falls above (or below if /RATIO = (SMALLER) is set) the ratio tolerance must also be smaller than the AREA_TOL tolerance. This tolerance is by default set to the /AREA = (MERGE_TOL) tolerance. For merging on the basis of ratio alone, /RATIO = (NOAREA_TOL) should be set.

Ratio consideration may also be negated individually for elimination or merging by using the NOMERGETOL or NOELIMTOL keyword options.

RESTRICTIONS

- Any two of /MERGE = (BY_UPPER), /MERGE = (BY_LOWER) or /MERGE = (BY_LOOKUP) are invalid.

- /MERGE = (BY_LARGE) is invalid with /MERGE = (BY_SMALL)

- Only one keyword argument may be specified with the /ELIMINATE qualifier.

- /MERGE = (BY_UPPER) or /MERGE = (BY_LOWER) require that link left/right codes held in the AC text field are numeric.

- ELIMINATE = (BY_UPPER) or ELIMINATE = (BY_LOWER) require that link left/right codes held in the AC text field are numeric.

- /MERGE = (BY_LOOKUP) requires that /LOOKUP is specified.

- /ELIMINATE = (BY_LOOKUP) requires that /LOOKUP is specified.

- /NOELIMINATE is invalid with use with /NOMERGE

- /RATIO = (LARGER) and /RATIO = (SMALLER) are mutually exclusive.

- If AC left/right coding is not present in the input file, then only /MERGE = (BY_LARGE), /MERGE = (BY_SMALL), /ELIMINATE = (BY_LARGE), or /ELIMINATE = (BY_SMALL) processing is valid.

- All IFF segment features must have unique feature serial numbers.

- The maximum number of polygons (POLMAX) that may be processed in a single POLMERGE run defaults to 10000, but can be increased by assigning a value to the logical name LSL$POLYGONS_POLMAX.

- The average number of sides per polygon is initially set to 5, but can be increased by assigning a value to the logical name LSL$POLYGONS_AVERAGE_SIDES.

- o The maximum of input segments that may be processed in a single POLMERGE run is calculated by multiplying the "maximum number of polygons" by "the average number of sides per polygon".
- o A single polygon ring may have a maximum of 100000 coordinates.
- o A single input segment may have a maximum of 20000 coordinates.
- o Polygon labels are restricted to a maximum of 255 characters.
- o At most 4 coverages may be propagated in a single POLMERGE run.
- o A polygon may have only one label.
- o In the lookup file a maximum of 127 primary codes may be specified.
- o In the lookup file a maximum of 127 secondary codes may be specified for each definition set.

DESCRIPTION

POLMERGE is the Laser-Scan automatic IFF **POL**YGON **MERG**ing and elimination utility. It forms part of the Laser-Scan POLYGONS Package.

POLMERGE is designed to be run in batch mode and all options may be specified on the command line. No user interaction is required during processing.

Polygons may be eliminated or merged by means of user-supplied rules, and it is to be stressed that good results depend on careful selection of qualifiers and lookup file criteria.

Input and output file characteristics.

The input file for POLMERGE is an IFF junction structured (IFJ) file. Such files will normally have been produced by other Laser-scan programs such as VECTORISE, ILINK or IPOLYGON. Within the program polygons are first formed . Merging is then carried out by selectively deleting the links which form the boundaries of the polygons to be formed. Output is also to an IFF junction structured file. If required IPOLYGON may then be run on the merged file to produce a variety of polygon output options.

The presence of left/right codes held in the text field of AC 4 and 5 entries in the IFF file allow merging to take place by means of a lookup file. These codes are preserved correctly in the output file regardless of the merging or elimination mechanism used.

File Checking.

POLMERGE carries out checks on the fidelity of the input file geometry and attribute coding. The following checks are carried out :-

Checks that segments in the input IFF file have junctions at each end.

Checks that all segments in the input IFF file have a minimum of two coordinates.

Checks that all junctions in the input IFF file have at least two arms.

Checks that all segments in the input IFF file have a unique FSN.

If the /ACCHECK qualifier is specified, the first type 4 and type 5 AC codes in each segment are checked for consistency in polygon formation.

POLMERGE and IPOLYGON.

Before polygon merging POLMERGE forms polygons internally from the junction structured IFJ file in a similar way to IPOLYGON . The user is therefore guided to the IPOLYGON chapter for further information on polygon formation and on data preparation procedures prior to processing.

IPOLYGON also provides checking facilities and in addition provides a /LITES2 qualifier which produces a LITES2 guidance file to take the user to those places where errors in the input data exist. It may in some cases, therefore, be useful to run IPOLYGON prior to POLMERGE processing.

Polygon Merging and Elimination.

POLMERGE distinguishes isolated polygons, which it handles separately. Such polygons are dealt with only by the /ELIMINATE qualifier. An isolated polygon is nested inside another and contains no common boundaries with any other polygons. In figure 1 polygon A is isolated and therefore may be eliminated whereas polygon B has a common boundary with C and is not eliminated but is a candidate for merging.

Both merging and elimination take place by default, but may be switched off with the /NOMERGE and /NOELIMINATE qualifiers. If both options are enabled then elimination takes place after merging. In this way it may be possible for isolations which are created as a result of the merging process to then be eliminated. A different result is therefore likely if the program is first run with /ELIMINATE and then /MERGE than if both options are enabled in the same run.

POLMERGE Tolerances.

Both eliminate and merge tolerances may be specified independently, although if only one is supplied then the other is assumed to be the same. The choice of tolerance is particularly important for the production of good results. An overlarge area tolerance may result in unpredictable results whereas a tolerance which is too small will produce an output file which is little different from the input file.

Any polygon which has an area greater than the area tolerance will not be merged into its neighbour regardless of its coding.

The ratio tolerance gives the user control over the shape of the polygons which are to be merged or eliminated. It is defined by

```
            Ratio = P / A
    where
            P  is the perimeter of the polygon
            A  is the square root of the area
```

In all area and perimeter measurements, nested polygon are taken into account. Thus in Figure 2 the polygon area of polygon A is represented by the shaded area and the perimeter includes both the inside and outside boundaries. Figure 3 provides a guide for ratio definition for various shapes.

In some circumstances it is preferable to preserve long thin areas at the expense of those with with a smaller ratio - and vice versa. To enable this there are /RATIO = LARGER and /RATIO = SMALLER options available.

Polygon Selection.

POLMERGE operates by first forming polygons internally. Each polygon is then passed through a series of tests which determine if the polygon is to be considered for merging as follows: -

1. Is the polygon below the merge tolerance (if /AREA = (MERGE_TOL is specified)?

2. Is the polygon above or below the ratio tolerance (if /RATIO = (MERGE_TOL, LARGER) or /RATIO = (MERGE_TOL,SMALLER) are specified)?

3. Does the polygon have a code which corresponds to any of the primary polygon labels in the lookup file (if /LOOKUP is specified)?

Only if the above tests are passed is the polygon considered for merging - although it should be noted that not all tests need be applied. Thus if /NOAREA and /NORATIO are set then only the /LOOKUP test would apply. Exactly analagous selection is applicable for polygon elimination.

It may be desirable to be able to specify a different area tolerance for those polygons which would be merged on the basis of the ratio test alone. Thus if it was required that all polygons be removed which are smaller than 3 units in area, but that long thin areas of up to 20 units in area were also removed then the following qualifier combination would achieve this.

```
  /AREA = (MERGE_TOL:3,ELIM_TOL:3)
  /RATIO =(MERGE_TOL:7,ELIM_TOL:7,AREA_TOL:20)
```

Note that the presence of the /LOOKUP qualifier restricts considerably the number of polygons considered for merging or elimination. Thus an empty lookup file would result in no polygons being merged or eliminated.

The Merging Process.

Once a polygon has been selected for merging, POLMERGE builds a list of all neighbouring polygons. Only those polygons with a common boundary to the candidate polygon are added to the neighbour list. This includes nested polygons, but not those polygons touching at only one point. Thus in Figure 4 the neighbours of polygon A are C,D,E and F. The file's bounding polygon is never considered as a neighbour.

The options specified with the /MERGE qualifier determine into which of the polygons neighbours it is merged. /MERGE = (BY_SMALL) and /MERGE = (BY_LARGE) merge the polygon into its smallest or largest neighbours by area. It is to be noted that neither of these mechanisms require the existence of AC left right codes in the input file.

/MERGE = (BY_UPPER) and /MERGE = (BY_LOWER) make use of numeric codes in the AC 4 and 5 left right codes.

With /MERGE = (BY_UPPER) specified a polygon is merged into a neighbour if it has a higher numeric code. If more than one of the neighbours has a higher value than the candidate polygon then the polygon with the value closest to it will be selected. If none of the neighbouring codes is greater then no merging takes place. /MERGE = (BY_LOWER) is analagous but with the lower numeric codes being considered.

/MERGE = (BY_LOOKUP) require that the input file contains valid AC 4 or 5 left right codes which are consistent for polygon generation. (see Lookup File section)

Normally only one merging option will be specified on the command line, and most combinations are disallowed (and undesirable). However it is possible for example to specify /MERGE =(BY_LOOKUP,BY_LARGE). In this case the largest neighbouring polygon is first determined. Merging only then takes place if this polygon is matched by one of the secondary codes in the appropriate definition set of the lookup file. In general, therefore, geometric constraints take priority over those based on coding.

The Elimination Process.

Once a polygon has been selected as a candidate for elimination the options specified with the /ELIMINATE qualifier determine whether the polygon is eliminated or not. If no keyword arguments are specified then elimination will always take place.

If /ELIMINATE = (BY_LOOKUP) is set then the polygon will only be eliminated if the code of the polygon's "mother" polygon corresponds to one of the secondary codes in the appropriate definition set of the lookup file. If /ELIMINATE =(BY_UPPER) or /ELIMINATE = (BY_LOWER) are set then the polygon is only eliminated if the "mother" has a higher or lower numeric code.

The Lookup File.

A POLMERGE Lookup file consists of a series of definition sets which defines polygon merging and elimination on the basis of polygon codes held in the text field of the AC 4 or 5 entries. Definition sets are separated by at least one blank line in the file.

A definition set is made up of one primary code and a number of subsequent secondary codes each placed on a separate line in the file. In the case of polygon merging if a polygon's code matches any of the primary codes in the file then it is considered for merging. If any of the neighbour polygons match the first secondary code in the same definition set then merging takes place. The new merged polygon will retain the secondary code. If none of the neighbours matches the first secondary code then the search is repeated for the next secondary code. Thus the order of the secondary codes in the definition set represent their priority. If no matches are found for any of the neighbours with any of the secondary codes then merging does not take place.

In the following small lookup file the primary codes are "AAA", "BBB", and "CCC"

```
! Definition set 1
AAA
ccc
FFF
A B C

! Definition set 2
BBB
ccc

! Definition set 3
CCC
```

The last definition set only contains only a primary code. In this case merging will take place, but the neighbouring polygon with which it is merged will be selected arbitrarily. Using a definition set with only one code is useful for constraining the coding of those polygons which are considered for merging, while allowing a different merging mechanism to be specified. For elimination it is even more useful because those polygons with the specified codes may be eliminated regardless of the coding of the "mother" polygon in which they lie. This would be achieved by specifying the /LOOKUP qualifier and the /ELIMINATE qualifier with no keyword arguments.

The following rules are used to determine whether polygon codes match with those in the definition sets.:-

```
1.  One space in a code is significant, but two are not :-
           "ABC" does not match "A BC"
    but    "A BC" matches  "A  BC"

2.  Matching is case sensitive :-
           "aBC" does not match "ABC"
```

The "!" character may be used to include comment lines in the lookup file. It should only be placed at the beginning of a line.

EXAMPLES

```
$ POLMERGE TEST50/LOG EXAMPLE1<CR>
%LSLLIB-I-IFFOPENED, LSL$SOURCEROOT:[POLYGONS.POLMERGE]TEST50.IFJ;1 opened for
read
%LSLLIB-I-IFFOPENED, LSL$SOURCEROOT:[POLYGONS.POLMERGE]EXAMPLE1.IFJ;3 opened for
write


%POLY-I-POLVAL, Maximum number of polygons set to a default of 40000
%POLY-I-DEFSID, Average number of sides per polygons set to a default of 5
+----------------------------------------------------------------+
|                                                                |
|                     Building IFF address tables                |
|                                                                |
+----------------------------------------------------------------+

Number of IFF segment feature addresses tabulated ... 465
Segment coordinate range is:
X-min ...............................................     0.000
X-max ...............................................    50.000
Y-min ...............................................     0.000
Y-max ...............................................    50.000


+----------------------------------------------------------------+
|                                                                |
|                         Forming polygons                       |
|                                                                |
+----------------------------------------------------------------+

Number of polygons formed (including isolations) ... 229
Number of segments used to form polygons ........... 465
Minimum number of segments used to form a polygon .. 1
Maximum number of segments used to form a polygon .. 88
Direction of polygon formation ..................... ANTICLOCKWISE


+----------------------------------------------------------------+
|                                                                |
|          Setting up sectored spatial index for polygons        |
|                                                                |
+----------------------------------------------------------------+

Number of boxes in X direction....................... 3
Number of boxes in Y direction....................... 3
Mean number of boxes per polygon ....................     0.039
Mean number of polygons per box .....................    25.444
```

```
+----------------------------------------------------------------+
|                                                                |
|                 Identifying isolated polygons                  |
|                                                                |
+----------------------------------------------------------------+

Number of polygons examined ......................... 229
Number of isolated polygons ......................... 18


+----------------------------------------------------------------+
|                                                                |
|                    Identifying nested polygons                 |
|                                                                |
+----------------------------------------------------------------+

Number of polygons examined ......................... 229
Number of polygons containing nested polygons ....... 4
Number of polygons nested inside others ............. 75


+----------------------------------------------------------------+
|                                                                |
|                   Checking Polygon Consistency                 |
|                                                                |
+----------------------------------------------------------------+



+----------------------------------------------------------------+
|                                                                |
|                      Calculating Polygon Areas                 |
|                                                                |
+----------------------------------------------------------------+

Maximum Polygon area ...   751.000
Minimum Polygon area ...     1.000


+----------------------------------------------------------------+
|                                                                |
|                      Merging Polygons                          |
|                                                                |
+----------------------------------------------------------------+

Area Tolerance .................................... 25.000
Number of polygons considered for merging ......... 123
Number of polygons merged ......................... 123
```

```
+-----------------------------------------------------------------+
|                                                                 |
|                      Eliminating Polygons                       |
|                                                                 |
+-----------------------------------------------------------------+

Area Tolerance ....................................... 25.000
Number of polygons considered for elimination ......... 52
Number of polygons eliminated ......................... 52


+-----------------------------------------------------------------+
|                                                                 |
|                      Writing output IFJ file                    |
|                                                                 |
+-----------------------------------------------------------------+

Number of segments written to output file ......... 24


 ELAPSED:    0 00:04:12.96  CPU: 0:03:40.90  BUFIO: 17  DIRIO: 204  FAULTS: 843
```

This example shows a successful run of POLMERGE in default mode. Both polygon merging and elimination has taken place and a default area tolerance of 25 square units has been calculated from the range entry of the input file. The polygons are first checked for consistency of left/right coding. Non-isolated polygons with an area less then the area tolerance are merged into their largest neighbours and all isolated polygons with an area less than the area tolerance are eliminated.

Note that the number of polygons eliminated is greater than the number of isolations first identified. This happens because isolated polygons are created as a result of the merging process.

The user has specified the /LOG qualifier which results in the output of run time statistics to SYS$OUTPUT.

$STATUS is set to SS$NORMAL - normal successful completion.

```
 $ POLMERGE TEST50/MER=(BYLOOKUP)/ELIM/AREA=(MER:5,ELIM:5) -
 EXAMPLE2/LOG/RATIO=(MER:7,ELIM:7,AREA:25)/LOOKUP=POLMERGE.DAT<CR>
%LSLLIB-I-IFFOPENED, LSL$SOURCEROOT:[POLYGONS.POLMERGE]TEST50.IFJ;1 opened for
read
%LSLLIB-I-IFFOPENED, LSL$SOURCEROOT:[POLYGONS.POLMERGE]EXAMPLE2.IFJ;5 opened for
write
%PMERGE-I-LKPOPNREAD, Lookup file LSL$LOOKUP:POLMERGE.DAT opened for reading
```

(For the sake of clarity all the log information has not been included here or in the following examples because the polygon formation statistics are the same as in the first example.)

```
               [ POLYGON FORMATION STATISTICS HERE ]

+-----------------------------------------------------------------+
|                                                                 |
|                      Merging Polygons                           |
|                                                                 |
+-----------------------------------------------------------------+

Area Tolerance ....................................  5.000
Ratio Tolerance ...................................  7.000
Ratio Area Tolerance ..............................  25.000
Lookup File ....LSL$LOOKUP:POLMERGE.DAT
Number of polygons considered for merging .........  112
Number of polygons merged .........................  112


+-----------------------------------------------------------------+
|                                                                 |
|                      Eliminating Polygons                       |
|                                                                 |
+-----------------------------------------------------------------+

Area Tolerance ........................................   5.000
Ratio Tolerance .......................................   7.000
Ratio Area Tolerance ..................................  25.000
Lookup File.....LSL$LOOKUP:POLMERGE.DAT
Number of polygons considered for elimination .........   38
Number of polygons eliminated .........................   38


+-----------------------------------------------------------------+
|                                                                 |
|                      Writing output IFJ file                    |
|                                                                 |
+-----------------------------------------------------------------+

Number of segments written to output file ......... 72


 ELAPSED:    0 00:51:12.80  CPU: 0:03:27.71  BUFIO: 27  DIRIO: 424  FAULTS: 830
```

This example shows POLMERGE used with both /LOOKUP and /RATIO qualifiers set. The user has also set all the tolerances explicitly. /RATIO =(LARGE) operation has been assumed so all polygons with a ratio larger then 7 and area less than 25 units are merged or eliminated, as well as all polygons less then the area tolerances of 5 units.

$STATUS is set to SS$NORMAL - normal successful completion.

```
$ POLMERGE TEST50/LOG/NOMERGE/ELIM=(BY_LOOKUP)/LOOKUP=POLMERGE.DAT
EXAMPLE3<CR>
%LSLLIB-I-IFFOPENED, LSL$SOURCEROOT:[POLYGONS.POLMERGE]TEST50.IFJ;1 opened for
read
%LSLLIB-I-IFFOPENED, LSL$SOURCEROOT:[POLYGONS.POLMERGE]EXAMPLE3.IFJ;5 opened for
write
%PMERGE-I-LKPOPNREAD, Lookup file LSL$LOOKUP:POLMERGE.DAT opened for reading

               [ POLYGON FORMATION STATISTICS HERE ]

+-----------------------------------------------------------------+
|                                                                 |
|                      Eliminating Polygons                       |
|                                                                 |
+-----------------------------------------------------------------+

Area Tolerance ........................................   25.000
Lookup File .....LSL$LOOKUP:POLMERGE.DAT
Number of polygons considered for elimination ......... 18
Number of polygons eliminated ......................... 10


+-----------------------------------------------------------------+
|                                                                 |
|                      Writing output IFJ file                    |
|                                                                 |
+-----------------------------------------------------------------+

Number of segments written to output file ......... 447

 ELAPSED:    0 00:07:27.07  CPU: 0:04:10.41  BUFIO: 23  DIRIO: 2344  FAULTS: 794
```

This example shows POLMERGE used to eliminate isolated polygons only (because NOMERGE is specified) using the lookup table. Note that all the isolated polygons formed fall below the default area tolerance. Their codes must also all appear as primary codes in the lookup table because all of the 18 isolated polygons are considered for elimination. However not every one of the polygons is eliminated because all their "mother" polygons do not appear as secondary codes in the appropriate definition sets of the lookup file.

$STATUS is set to SS$NORMAL - normal successful completion.

```
 $ POLMERGE TEST20.DUFF/LOG EXAMPLE4<CR>
%POLMERGE-E-ACINCON, polygon labels in AC texts inconsistent in feature with FSN
13 (13)
current segment: 3
previous segment: 2
Polygon labels in AC texts inconsistent in feature with FSN 13 - polygon
abandoned
```

```
%POLMERGE-E-ACINCON, polygon labels in AC texts inconsistent in feature with FSN
15 (15)
current segment: 1
previous segment: 4
Polygon labels in AC texts inconsistent in feature with FSN 15 - polygon
abandoned
 ELAPSED:    0 00:00:21.61  CPU: 0:00:19.43  BUFIO: 8  DIRIO: 21  FAULTS: 920
```

This example shows a run of POLMERGE which fails during the polygon consistency checking phase. The specified AC labels are inconsistent for the polygons which have been formed. POLMERGE processing stops after this run with $STATUS set to SS$ABORT. Polygon consistency checking can be negated with the /NOACCHECK qualifier, but is dangerous if /LOOKUP is used and the input data has not been produced directly from an other utility such as VECTORISE or IPOLYGON.

```
$  POLMERGE TEST50/AREA=(MER:10,NOELIM)/ELIM=(BYLOOKUP)  -
 /MER=(BYLARGE)/LOOKUP=POLMERGE.DAT EXAMPLE5/LOG<CR>
%LSLLIB-I-IFFOPENED, LSL$SOURCEROOT:[POLYGONS.POLMERGE]EXAMPLE5.IFJ;1 opened for
read
%LSLLIB-I-IFFOPENED, LSL$SOURCEROOT:[POLYGONS.POLMERGE]EXAMPLE5.IFJ;11 opened
for write

                [ POLYGON FORMATION STATISTICS HERE]

+-----------------------------------------------------------------+
|                                                                 |
|                      Merging Polygons                           |
|                                                                 |
+-----------------------------------------------------------------+

Area Tolerance ..................................   10.000
Lookup File ....LSL$LOOKUP:POLMERGE.DAT
Number of polygons considered for merging ......... 121
Number of polygons merged ......................... 121


+-----------------------------------------------------------------+
|                                                                 |
|                      Eliminating Polygons                       |
|                                                                 |
+-----------------------------------------------------------------+

Lookup File ....LSL$LOOKUP:POLMERGE.DAT
Number of polygons considered for elimination ......... 48
Number of polygons eliminated ......................... 40
```

```
+-----------------------------------------------------------------+
|                                                                 |
|                     Writing output IFJ file                     |
|                                                                 |
+-----------------------------------------------------------------+

Number of segments written to output file ......... 50
 ELAPSED:    0 00:04:15.21  CPU: 0:03:34.31  BUFIO: 22  DIRIO: 347  FAULTS: 1055
```

This example demonstrates the way in which the rules for merging and elimination can be different. An area tolerance of 10 is applied for merging, but the NOELIM keyword with the /AREA qualifier specifies that no area tolerance is to be applied to isolated polygons for elimination. Polygons for merging are merged into their largest neighbours whereas polygons for elimination are merged according to the definition sets in the lookup file. Note, however, that even although merging is not by means of the lookup table, those polygons which are considered for merging are limited to those having codes which match one of the primary polygon labels in the lookup file.

$STATUS is set to SS$NORMAL - normal successful completion.

MESSAGES (INFORMATIONAL)

These messages give information only, and require no immediate action by the user. They are used to provide information on the current state of the program, or to supply explanatory information in support of a warning or error message.

LKPOPNREAD, Look up file 'file-spec' opened for reading.

Explanation: The specified POLMERGE look-up file has been opened successfully.

User action: None.

LSTOPNOUT, /LIST list file 'file-spec' opened for output.

Explanation: The specified listing file has been opened successfully.

User action: None.

MESSAGES (WARNING)

These messages are output when an error has occurred that can be corrected immediately by the user or that the program will attempt to overcome.

ACIGN, AC of type other than 4 or 5 ignored in feature 'number'.

Explanation: The input file contains an AC of type other than the left right codes 4 or 5. It is not copied to the output file.

User action: Check that useful information held in these codes is not being lost. In general polygon data should not contain ACs which do not represent left right codes.

MISSAC, Missing Left or Right AC entry in NF 'number'.

Explanation: A feature was present in the input file with no AC 4 or 5 left right codes. This message is only output when the /ACCHECK qualifier is present.

User action: Check the input file and correct using LITES2. If polygon merging is to occur regardless of left right coding then /NOACCHECK should be specified.

MULTAC4, multiple AC 4 codes - only first is output in feature with FSN 'number'.

Explanation: A feature in the input file has more than one AC 4 code. POLMERGE only reads one AC 4 per feature. Any subsequent ones are ignored and not output to the merged file.

User action: Check that no useful information is held in the secondary ACs which would be lost after POLMERGE processing.

MULTAC5, multiple AC 5 codes - only first is output in feature with FSN 'number'.

Explanation: A feature in the input file has more than one AC 5 code. POLMERGE only reads one AC 5 per feature. Any subsequent ones are ignored and not output to the merged file.

User action: Check that no useful information is held in the secondary ACs which would be lost after POLMERGE processing.

PHANNOMOTH, Phantom neighbour for polygon number 'number' has no mother polygon, and will be eliminated.

Explanation: The phantom neighbour for an isolated polygon which is being eliminated has no mother polygon, and will be eliminated regardless of specified elimination basis.

User action: Check the input .IFJ file if the resulting .IFF has an error.

TOOMNYBOUNDS, too many common boundaries between polygons starting with FSN 'number' and 'number'

Explanation: The number of common boundaries between two polygons to be merged has exceeded the current limit. Processing continues but the polygon is not merged.

User action: The specified FSNs will allow the polygon boundary to be identified using LITES2. It is not envisaged that this boundary limit will be exceeded and so Laser-Scan should be notified if this message occurs.

TOOMNYNEIGH, too many neighbours in polygon with first FSN 'number' - not merged.

Explanation: The limits for the number of neighbours a polygon may have for merging to occur has been exceeded. Processing continues but the polygon is not merged.

User action: The specified FSNs will allow the polygon to be identified using LITES2. Generally the polygon will be fairly large. It is not envisaged that this boundary limit will be exceeded and so Laser-Scan should be notified if this message occurs.

MESSAGES (ERROR)

These messages indicate an error in processing which will cause the program to terminate. The most likely causes are a corrupt or otherwise invalid input file, or an error related to command line processing and file manipulation.

ACINCON, polygon labels in AC texts inconsistent in feature with FSN %N (%N).

Explanation: If the /ACCHECK qualifier is present, this message is output for those polygons formed with inconsistent labels in the text fields of their AC 4 or 5 codes. POLMERGE processing stops after the consistency checking stage.

User action: Correct the input file using LITES2. If polygon merging is to occur regardless of the left right coding then specify /NOACCEHCK.

CLINIFF, error closing input IFF file 'file-spec'.

Explanation: POLMERGE has failed to close the input IFF file.

User action: Supplementary messages which follow this message should help to identify the cause of the failure.

CLLIST, error closing list file 'file-spec'.

Explanation: POLMERGE has failed to close the listing file specified with the /LIST qualifier.

User action: Supplementary messages which follow this message should help to identify the cause of the failure.

CLLOOKUP, error closing lookup file 'file-spec'.

Explanation: POLMERGE has failed to close the lookup file selected with the /LOOKUP qualifier.

User action: Supplementary messages which follow this message should help to identify the cause of the failure.

CLOUTIFF, error closing output IFF file 'file-spec'.

Explanation: POLMERGE has failed to close the output IFF file.

User action: Supplementary messages which follow this message should help to identify the cause of the failure.

INVALELIMTOL, 'tolerance' is an invalid eliminate tolerance - must be positive.

Explanation: One of the eliminate tolerances specified with the /AREA or /RATIO qualifiers is negative.

User action: Correct the tolerance values on the command line.

INVALMERTOL, 'tolerance' is an invalid merge tolerance - must be positive.

Explanation: One of the merge tolerances specified with the /AREA or /RATIO qualifiers is negative.

User action: Correct the tolerance values on the command line.

OPLIST, error opening /OUTPUT list file 'file-spec' for output.

Explanation: POLMERGE is unable to open the file specified by the /LIST qualifier.

User action: Supplementary messages which follow this message should help to identify the cause of the failure.

OPLOOKUP, error opening /LOOKUP file 'file-spec' for reading.

Explanation: POLMERGE is unable to open the file specified by the /LOOKUP qualifier.

User action: Supplementary messages which follow this message should help to identify the cause of the failure.

TOOMNYCODES, too many primary or secondary codes in lookup file.

Explanation: The limit of the number of primary or secondary codes which may be specified in the lookup file has been exceeded.

User action: If possible, remove the number of non-essential codes in the lookup file. If the number of codes required are still greater then the maximum permitted, then please consult Laser-Scan.

UNEXPEOF, unexpected end of IFF file 'file-spec'.

Explanation: The specified input IFF file was corrupt and did not have an IFF EJ entry.

User action: Use IMEND to correctly terminate the file. If the segment file is bad, re-run ILINK/STRUCTURE on the repaired IFF file before using POLMERGE. If the problem persists try reading the suspect file into LITES2 (and then exiting to create a new version of the file) before retrying ILINK and POLMERGE. If the problem still persists then consult Laser-Scan.

MESSAGES (FATAL)

These messages indicate a severe error in processing, or some form of system failure, which has caused the program to terminate.

INTRNLERR, internal consistency error 'number' detected in routine 'routine-name' - please submit an SPR.

Explanation: POLMERGE has detected a sever error in the named routine.

User action: Please report the problem to Laser-Scan. Include the input file and run information in a Software Performance Report (SPR).

LOST, failed to locate IFF entry at recorded address - position lost.

Explanation: POLMERGE has attempted to locate an entry in one of the input IFF files which no longer appear to exist. This is an unrecoverable error and POLMERGE is consequently lost.

User action: Try reading the input files into LITES2. If this succeeds than the problem lies within POLMERGE. Please make a copy of the input IFF files and report the problem to Laser-Scan.

UNEXPENTJP, unexpected entry 'entry' found while patching JP entries after NF 'feature number'.

Explanation: POLMERGE was unable to patch a JP entry while updating a file.

User action: Please make a copy of the input IFF files and report the problem to Laser-Scan.

MESSAGES (OTHER)

In addition to the above messages which are generated by the program itself, other messages may be produced by the command line interpreter (CLI) and by Laser-Scan libraries. In particular, messages may be generated by the IFF library and by the Laser-Scan I/O library, LSLLIB. IFF library messages are introduced by '%IFF' and are documented in the IFF library users' guide. In most cases IFF errors will be due to a corrupt input file, and this should be the first area of investigation. If the cause of the error cannot be traced by the user, and Laser-Scan are consulted, then the output file should be preserved to facilitate diagnosis. LSLLIB messages are introduced by '%LSLLIB' and are generally self-explanatory. They are used to explain the details of program generated errors.

CHAPTER 5

MODULE POLY_MESSAGES

POLYGONS library messages

The programs which make up the POLYGONS package share a common library of polygon formation and manipulation routines. These POLYGONS library routines issue messages having the prefix 'POLY__'. The messages are as follows:

MESSAGES (SUCCESS)

These messages are used to indicate that the program has succeeded in performing some action, and do not require any user action.

NORMAL, normal successful completion.

Explanation: The current POLYGONS library routine has completed successfully.

User action: None.

MESSAGES (INFORMATIONAL)

These messages give information only, and require no immediate action by the user. They are used to provide information on the current state of the program, or to supply explanatory information in support of a warning or error message.

ALLOCD, Allocated 'integer' Bytes before failure.

Explanation: Additional message to POLY_F_MEMORY.

User action: None.

DEFPOL, Maximum number of polygons set to a default of 'integer'

Explanation: The logical name LSL$POLYGONS_POLMAX was either undefined or bad. POLYGONS is setting the maximum number of polygons to the specified default value.

User action: None.

DEFSID, Average number of sides per polygons set to a default of 'integer'

Explanation: The logical name LSL$POLYGONS_AVERAGE_SIDES was either undefined or bad. POLYGONS is setting the maximum number of polygons to the specified default value.

User action: None.

POLVAL, Maximum number of polygons set to 'integer'

Explanation: The logical name LSL$POLYGONS_POLMAX was successfully parsed to the specified value

User action: None.

SIDVAL, Average number of sides per polygon set to 'integer'

Explanation: The logical name LSL$POLYGONS_AVERAGE_SIDES was successfully parsed to the specified value

User action: None.

SUGGEST, Suggested maximum value for LSL$__POLYGONS__POLMAX = 'integer'.

Explanation: Additional message to POLY_F_MEMORY.

User action: set the logical name LSL$_POLYGONS_POLMAX to a number less-than or equal-to the suggested value.

MESSAGES (WARNING)

These messages are output when an error has occurred that can be corrected immediately by the user or that the program will attempt to overcome.

BADJUN, 'integer' arm junction detected at ('x coord','y coord').

Explanation: A zero or one arm junction has been detected at the specified location. Warnings relating to one arm junctions are only issued if the user has specified the /ONEARM=WARN command qualifier.

User action: A zero arm junction would cause serious problems to the POLYGONS library routines. Program execution is normally abandoned after one (or more) zero arm junctions have been detected. Such junctions should not occur. Examine all stages of the flowline used to create the junction structured input file. Check the HI (HIstory) record of the input file for signs of abnormal processing termination earlier in the flowline. If the problem cannot be traced, please contact Laser-Scan. One arm junctions are quite acceptable to the POLYGONS library polygon formation routines. The presence of an unwanted one arm junction may indicate poor quality control in the production flowline.

DBLDIG, double digitising suspected.

Explanation: Evidence of double-digitising has been discovered. This has been more specifically located in previous warnings. Execution will probably be aborted soon after this warning is detected.

User action: Investigate the segments specified in earlier warnings for possible overlap or coincidence. If double-digitising is found then the offending segment(s) should be edited out using LITES2 and junction-structure rebuilt using ILINK.

DUPSEGFSN, duplicate segment FSN found.

Explanation: Two or more input file features (segments) share the same FSN (Feature Serial Number). As the POLYGONS library routines use the segment FSN to uniquely identify segments, this would cause later processing problems. Processing will usually be terminated after these duplicate FSN warnings.

User action: Use the IMP utility IRENUMBER to reallocate the input feature FSNs to unique values.

INVFEAT, invalid IFF segment feature.

Explanation: Problems have been found in one or more of the input IFF segments. These will have been detailed in previous warnings. Execution will probably be aborted soon after this warning is given.

User action: Correct the problems detailed by previous warnings and re-run IPOLYGON.

LOSTARM, unable to find current arm in feature with FSN 'integer'.

Explanation: The program has come along the specified segment to a junction in which it cannot locate the current arm. This problem may be caused by double digitising, whereby two arms lie at exactly the same angle within the junction.

User action: Use LITES2 to examine the input segment file, either use LITES2 or ILINK/LLJOIN followed by ILINK/MERGE to remove all traces of double digitising. Use ILINK/STRUCTURE to re-create junction structure and the re-run the POLYGONS utility.

MDDEFAULT, MD error: origin defaulted to (0,0).

Explanation: The /ABSOLUTE qualifier has been given, but there was some error in the MD (map descriptor) entry, either not type 2 or not a valid MD type 2 entry. The origin offset has been set to a default value of (0,0) and the /ABSOLUTE qualifier ignored.

User action: Check the MD entry, and check that it is type 2 and of the correct length.

MULTIDUF, polygon 'integer' multiply flagged as duff.

Explanation: POLYGONS utilities usually attempt to continue processing even if polygon formation fails due to bad input IFF junction structure. The incomplete polygon is flagged as being duff and is carefully avoided throughout the remaining processing stages. The input IFF junction structure is corrupt in such a way that recursive polygon formation has occurred.

User action: While the POLYGONS utility will make every effort to ensure that the output files will not contain corrupt polygons, the output files should be checked very carefully. Ideally, the cause of the junction structure corruption should be traced and remedied, and then the POLYGONS utility should be re-run.

MULTISEG, segment with FSN 'integer' used to form multiple polygons.

Explanation: POLYGONS utilities attempt to continue processing even if polygon formation fails due to bad input IFF junction structure. The incomplete polygon is flagged as being duff and is carefully avoided throughout the remaining processing stages. One result of such action is that segments cannot properly be flagged as 'used' in a given direction - the polygon which used the segment is flagged as duff and invalid. Segments may thus be used several times near the site of a corrupt polygon and this message is output to warn the user that the output files may contain spurious polygon features.

User action: While the POLYGONS utility will make every effort to ensure that the output files will not contain corrupt polygons the output files should be checked very carefully. Ideally, the cause of the junction structure corruption should be traced and remedied, and then the POLYGONS utility should be re-run.

NOARM, no junction arms to follow in feature with FSN 'integer'.

Explanation: The program has encountered a junction in which there are no arms, or no arms left unused to follow. This should not normally happen, as zero arm junctions are trapped prior to polygon formation.

User action: Use LITES2 to examine and correct the input segment file. Use ILINK/STRUCTURE to re-create junction structure and the re-run the POLYGONS utility.

PIPFAIL, unable to generate a point inside polygon with ID 'integer' and coordinate ('x coord','y coord').

Explanation: an attempt was made to automatically generate a point that was guaranteed to lie inside a specified polygon. The routine has unexpectedly failed for this particular polygon.

User action: please make a copy of the dataset and command line being used for possible later analysis and submit an SPR.

TOOMNYNODES, too many nodes.

Explanation: An attempt has been made to add extra arms to a node.

User action: Check for previous errors.

UNEXPSEC, unexpected sector 'integer' found.

Explanation: The program has calculated that an output file junction lies in a junction sector that does not lie within the sector header.

User action: Use LITES2 to check the output file very carefully.

MESSAGES (ERROR)

These messages indicate an error in processing which will cause the program to terminate. The most likely causes are a corrupt or otherwise invalid input file, or an error related to command line processing and file manipulation.

BOUNDING, unable to determine bounding polygon.

Explanation: The program cannot identify a unique bounding polygon.

User action: Use LITES2 to ensure that all junctions are properly formed around the periphery of the polygon area. There should be a single boundary. Attempts to run IPOLYGON on groups of islands will fail in this way. (See the IPOLYGON chapter "IPOLYGON treatment of the Bounding Polygon" section.) If necessary add an artificial bounding polygon away from the original linework using LITES2, then use ILINK/STRUCTURE to re-create the junction structure and the re-run the POLYGONS utility.

FORMPOLY, previous warnings invalidate polygon formation - aborting

Explanation: warnings have been issued during polygon formation. Execution has continued in an attempt to find all the causes. Now the polygon formation phase has completed and is known to be corrupt, execution will cease.

User action: Take what action is suggested by the previous warnings.

MDABSENT, MD (map descriptor) entry missing.

Explanation: The /ABSOLUTE qualifier has been given, but there was no MD (map descriptor) entry.

User action: Check the IFF file for a valid MD entry.

POLPTS, too many points in polygon coordinate buffer - max. allowed is 'integer'.

Explanation: To facilitate seed point assignment and nested polygon determination, the program must place all the coordinates which define each polygon in turn into a temporary buffer. At least one polygon is defined by more than the permitted number of coordinates.

User action: Check the input segment IFF file. Do the polygons need to have so many defining points? If so, then use LITES2 to split the segment data into two or more files to ensure that no polygon will be defined by more than the permitted number of coordinates. Re-run ILINK on the divided segment files and then re-run IPOLYGON.

POSNEST, too many possible nested polygons - maximum allowed is 'integer'

Explanation: IPOLYGON can only process up to the specified maximum number of nested polygons. If the polygon structure is too complicated IPOLYGON runs out of internal storage for candidate nested polygons during the nested polygon identification phase of execution.

User action: Check the input segment IFF file. Do the polygons need to be so complex? If so, then use LITES2 to split the segment data into two or more files to ensure that no polygon will contain no more than the permitted number of nested polygons. Re-run ILINK on the divided segment files and then re-run IPOLYGON.

SEGPTS, too many points in segment coordinate buffer - max. allowed is 'integer'

Explanation: To facilitate seed point assignment and nested polygon determination IPOLYGON must be able to read all the coordinates which define each segment forming a polygon into a temporary buffer. At least one segment is defined by more than the permitted number of coordinates.

User action: Check the input segment IFF file. Do the segments need to have so many defining points? If so, then use LITES2 to split segments which have more than the permitted number of coordinates into two or more segments to ensure that no single segment feature is defined by more than the permitted number of coordinates. Re-run ILINK on the divided segment files and then re-run IPOLYGON. IPOLYGON is designed to cope with the "2 arm" IFF junctions which result from such division of features.

TABBUILD, previous warnings invalidate lookup tables - aborting

Explanation: IPOLYGON has detected errors in the input IFF segment and/or seed point features that make further processing pointless.

User action: Use the warnings output by IPOLYGON and the LITES2 command file (if specified) to correct the input IFF files using LITES2. When all the edits are complete (and if necessary ILINK run on the segment IFF file), re-run IPOLYGON using the corrected IFF files.

TOOMNYNEST, too many nested polygons - maximum allowed is 'integer'

Explanation: IPOLYGON can only process up to the specified maximum number of nested polygons.

User action: Check the input segment IFF file. Do the polygons need to be so complex? If so, then use LITES2 to split the segment data into two or more files to ensure that no polygon will contain no more than the permitted number of nested polygons. Re-run ILINK on the divided segment files and then re-run IPOLYGON.

TOOMNYPOL, too many polygons formed - maximum allowed is 'integer'

Explanation: IPOLYGON can only process up to the specified maximum number of polygons. This value must also include 'fake' polygons caused by isolated polygons leading to polygon duplication within IPOLYGON workspace. Such 'fake' polygons are stored twice: once as the inner boundary of a "doughnut" shaped outer polygon and once as the isolated polygon in its own right. Thus polygon maps which contain many isolated polygons will significantly increase demands on IPOLYGON internal workspace.

User action: Use LITES2 to split seed point and the segment IFF files into two or more files. Ensure that there will be no more than the permitted number of polygons in any sub-file created from the original file. Re-run ILINK on the divided segment files and then re-run IPOLYGON.

TOOMNYPS, too many polygons segments - maximum allowed is 'integer'

Explanation: IPOLYGON can only process up to the specified maximum number of segments involved in polygon boundaries.

User action: Use LITES2 to split seed point and the segment IFF files into two or more files. Re-run ILINK on the divided segment files and then re-run IPOLYGON.

TOOMNYRING, too many rings extracted - maximum allowed is 'integer'

Explanation: Internally IPOLYGON traces round a polygon boundary using all connected segments regardless of the setting of /ONEARM. If /ONEARM=USE is not specified then these boundaries are analysed into loops. This multiple pass operation can only deal with a maximum number of extracted loops.

User action: Check to see that IPOLYGON runs with the /ONEARM=USE combination present.

TOOMNYSEED, too many seed point features read - maximum allowed is 'integer'

Explanation: IPOLYGON can only process up to the specified maximum number of seed points.

User action: Use LITES2 to split seed point and the segment IFF files into two or more files. Ensure that there will be no more than the permitted number of segments in any sub-file created from the original file. Re-run ILINK on the divided segment files and then re-run IPOLYGON.

TOOMNYSEG, too many segment features read - maximum allowed is 'integer'

Explanation: IPOLYGON can only process up to the specified maximum number of segments.

User action: Use LITES2 to split seed point and the segment IFF files into two or more files. Ensure that there will be no more than the permitted number of segments in any sub-file created from the original file. Re-run ILINK on the divided segment files and then re-run IPOLYGON.

TOOMNYTREE, too many trees formed - maximum allowed is 'integer'

> **Explanation:** IPOLYGON can only process up to the specified maximum number of trees - zero-area nested polygons.
>
> **User action:** Check the input segment IFF file. Are trees required? They can be eliminated by not specifying /ONEARM=USE. If they are required, then use LITES2 to split the segment data into two or more files to ensure that no polygon will contain no more than the permitted number of trees. Re-run ILINK on the divided segment files and then re-run IPOLYGON.

UNEXPEOF, unexpected end of IFF file 'file-spec'

> **Explanation:** specified input IFF file terminated before an IFF EJ entry was encountered.
>
> **User action:** Use IMEND to correctly terminate the file. Re-run the POLYGONS program on the repaired IFF file. If the problem persists try reading the file into LITES2 and then exit. Re-run the POLYGONS program.

MESSAGES (FATAL)

These messages indicate a severe error in processing, or some form of system failure, which has caused the program to terminate.

BADFLAG, bad internal segment flags

Explanation: Internal flags have become incorrectly set, possibly as a result of the use of an input segment file containing invalid IFF junction structure.

User action: It is possible that the input segment file contains corrupt junction structure and this should be investigated before contacting Laser-Scan. Use the IMP utility IPATCH to examine the file or ITOTEXT/ADDRESS, (if the file is not too big)! If the junction structure is believed correct, or is incorrect but produced by a Laser-Scan utility, please save the input files and submit an SPR to Laser-Scan.

FATAL, Unspecified fatal error

Explanation: The current POLYGONS library routine has failed in a bad way. This message should not normally be seen by the user.

User action: Contact Laser Scan and describe in detail the steps which led to this message.

LOST, failed to locate IFF entry at recorded address - position lost

Explanation: IPOLYGON has incorrectly stored the address of an entry within one of the input IFF files and has now attempted to locate that IFF entry. This is a very severe error. IPOLYGON is irrevocably lost.

User action: Try reading the input IFF files into LITES2. If this is successful then the problem lies within IPOLYGON itself; please make a copy of the input IFF files and report the problem to Laser-Scan.

MEMORY, **E**RROR allocating virtual memory.

Explanation: The polygons package dynamically memory handling routines were not able to allocate the memory that it required.

User action: Reduce the values of LSL$POLYGONSPOLMAX and LSL$POLYGONSSIDES.

STACKOVER, polygon formation stack overflow

Explanation: The polygon library routines form each polygon on a stack which is only flushed into the programs polygon storage when the polygon is complete. The current polygon has more segments than there is room on the formation stack. This should never happen.

User action: Please save the input files and submit an SPR to Laser-Scan.

MESSAGES (OTHER)

In addition to the above messages which are generated by the program itself, other messages may be produced by the command line interpreter (CLI) and by Laser-Scan libraries. In particular, messages may be generated by the IFF library and by the Laser-Scan I/O library, LSLLIB. IFF library messages are introduced by '%IFF' and are documented in the IFF library users' guide. In most cases IFF errors will be due to a corrupt input file, and this should be the first area of investigation. If the cause of the error cannot be traced by the user, and Laser-Scan are consulted, then the output file should be preserved to facilitate diagnosis. LSLLIB messages are introduced by '%LSLLIB' and are generally self-explanatory. They are used to explain the details of program generated errors.

INDEX

CLI, 1-2
CODSEG, 2-1

Getting started with POLYGONS, 1-2

IFF
  and POLYGONS, 1-1
ILINK, 3-2
  and ISTSEL, 3-2
  Command Qualifiers
    /ACP, 2-53
    /LLJOIN, 2-52 to 2-53
    /MERGE, 2-52 to 2-53
    /STRUCTURE, 2-53
IPOLYGON, 2-1
  ACs, 2-45
    contain, 2-7, 2-25 to 2-27
    left, 2-6, 2-25 to 2-27
    polygon identifier, 2-12 to 2-13, 2-16 to 2-17, 2-21, 2-27
    right, 2-6, 2-25 to 2-27
    type 12, 2-25 to 2-27
    type 4, 2-6, 2-25 to 2-27, 2-48, 2-50
    type 5, 2-6, 2-25 to 2-27, 2-48, 2-50
    type 82, 2-12 to 2-13, 2-16 to 2-17, 2-21, 2-27
  and Area-Fill, 2-39, 2-52
  and Batch Mode, 2-1
  and CODSEG, 2-1
  and Composite Text, 2-47
  and Double Digitising, 2-52 to 2-53
  and FC pairs, 2-22, 2-53
  and Identifiers, 2-1, 2-9, 2-44
  and ILINK, 2-52
  and IMP, 2-52
  and Labelling, 2-1, 2-21, 2-44 to 2-45
    by Left/Right Code, 2-1, 2-6, 2-45, 2-48
    by propagation, 2-49
    by Seed Point, 2-1, 2-21, 2-45 to 2-46
  and LITES2, 2-52, 2-63
  and Nesting, 2-1, 2-9, 2-15, 2-33, 2-39, 2-61
    first-order, 2-15, 2-39
  and One-Arm Junctions, 2-7, 2-33, 2-62
  and POLCHK, 2-1
  Area Calculation, 2-9, 2-12, 2-16, 2-60
  Bounding Polygon, 2-39, 2-44, 2-47, 2-60
    and /OPTIONS=NEST, 2-39
    and /SEED, 2-22
    Label, 2-23, 2-44
    restrictions, 2-44
  Checks on IFF input, 2-51
    junction structure, 2-51
    seed points, 2-51
  Command Qualifiers, 2-2 to 2-3
    /ABSOLUTE, 2-2 to 2-3
    /ASCII, 2-2, 2-4, 2-57 to 2-58
      and /LIST, 2-4
      default file-spec, 2-4
      default setting, 2-4
    /LIST, 2-2, 2-5
      and /ASCII, 2-5
      default file-spec, 2-5
    /LITES2, 2-2, 2-5
      default file-spec, 2-5
    /LOG, 2-2, 2-5
    /LRCODE, 2-2, 2-6, 2-48
      and /SEED, 2-6
      left AC, 2-6, 2-48
      right AC, 2-6, 2-48
    /ONEARM, 2-2, 2-7, 2-25 to 2-26, 2-33, 2-62
      and /LITES2, 2-8
      and /SEGMENTS, 2-7
      and Junctions, 2-7
    /ONEARM=CONTAIN, 2-7, 2-62
    /ONEARM=DELETE, 2-7, 2-62
    /ONEARM=USE, 2-8, 2-62
    /ONEARM=WARN, 2-8, 2-51
    /OPTIONS, 2-2, 2-9
      default setting, 2-9
    /OPTIONS=AREA, 2-9
    /OPTIONS=CLOCKWISE, 2-9, 2-33
    /OPTIONS=IDENT_TEXT, 2-9, 2-44
    /OPTIONS=NONEST, 2-9, 2-33
    /OPTIONS=UNDEFINED, 2-10
    /PIP, 2-2, 2-11, 2-57, 2-60
      default feature code, 2-11
      default file-spec, 2-11

```
      default layer, 2-11
      default setting, 2-11
    /PIP=ITERATE, 2-60
    /PME, 2-2, 2-14
    /POLYGONS, 2-2, 2-15, 2-57,
        2-61
      and /SEED, 2-16, 2-47
      default feature code, 2-15
      default file-spec, 2-15
      default layer, 2-16
      default setting, 2-15
    /PRINTER, 2-2, 2-18
      and /LIST, 2-18
    /PROPAGATE, 2-3, 2-49
      left AC, 2-50
      right AC, 2-50
    /PROPAGATE=CONCATENATE, 2-19
    /PROPAGATE=COVERAGE_FILE,
        2-19
    /PROPAGATE=FULL, 2-19
    /PROPAGATE=LEFT, 2-19
    /PROPAGATE=PARTIAL, 2-19
    /SEED, 2-3, 2-21, 2-46, 2-51
      and /LRCODE, 2-21
    /SEED=FC, 2-21, 2-46
    /SEED=FILE, 2-21
      and /SEED=FC, 2-21
      and /SEED=LAYER, 2-21
      default file-spec, 2-21
    /SEED=LAYER, 2-22, 2-46
    /SEED=PAIR, 2-53
      and /POLYGONS, 2-22
      default file-spec, 2-22
    /SEED=SURROUND, 2-23, 2-44
    /SEED=USE, 2-23, 2-46
      and /POLYGONS, 2-47
    /SEED=USE:AC, 2-21, 2-23,
        2-46 to 2-47
    /SEED=USE:FSN, 2-23, 2-46 to
        2-47
    /SEED=USE:TEXT, 2-23, 2-47
    /SEGMENTS, 2-3, 2-24, 2-57,
        2-62
      and ACs, 2-25 to 2-26
      and output selection, 2-26
      default ACs, 2-25 to 2-26
      default file-spec, 2-24
      default setting, 2-24
    /SEGMENTS=JUNCTIONS, 2-62
  Coverage, 2-49
  Default IFF parameters, 2-27
  Description, 2-30
  Directed Segments, 2-33, 2-58
  Examples, 2-68
  Format, 2-2
```

```
  Function, 2-1
  "Ignored Segment", 2-25 to 2-26
  Invisible Steps, 2-15, 2-61
  Labelling Failure, 2-4, 2-10,
      2-12, 2-16, 2-26, 2-44,
      2-48, 2-50
  Left/Right coding, 2-1, 2-6,
      2-45, 2-48
    requirements, 2-48
  Loops, 2-33, 2-44
    definition, 2-33
  Output Options, 2-1, 2-57
    Coded Segments, 2-1, 2-24,
        2-57, 2-62
      junction structure, 2-24 to
          2-26
    Point-in-Polygon, 2-1, 2-11,
        2-57, 2-60
    Polygon Boundaries, 2-1, 2-15,
        2-57, 2-61
    Segment Listings, 2-1, 2-4,
        2-57 to 2-58
  Parameter, 2-3
  "Polygon ", 2-9, 2-44
  Polygon Boundaries, 2-30
  Polygon Formation, 2-30, 2-45
  Polygon Overlay
    example, 2-75
  Production Flowline, 2-52
  Program Operation, 2-54
  Prompt, 2-3
  Propagation
    and AC types, 2-19
    and concatenation, 2-19
    and coverage file, 2-19
    full, 2-19
    partial, 2-19
  Propagation of left-right
      codings, 2-19
  Propagation of Left/Right
      coding, 2-45
  Restrictions, 2-28
    Bounding Polygon, 2-44
    Command Line Decoding, 2-29
    Parameters, 2-28
  Seed Point, 2-1, 2-21, 2-45 to
      2-46, 2-51
    identification, 2-21 to 2-22,
        2-46
    requirements, 2-46
  "Surrounding void", 2-23, 2-44,
      2-47
  Trees, 2-8, 2-36
    definition, 2-36
  TX entries, 2-23, 2-47
```

  "Undefined Polygon", 2-10, 2-26
ISTSEL, 3-1
  and AC entries, 3-1
  and IFF junction structure, 3-2
  and ILINK, 3-2
  and TX entries, 3-1
  code definition file, 3-3
    example, 3-3
  command qualifiers, 3-1 to 3-2
    /DEF_FILE, 3-2
    /LOG, 3-2
  description, 3-2
    general, 3-2
    selection critera, 3-2
  examples of use, 3-5
  format, 3-1
  parameters, 3-1
  program operation, 3-2
  prompts, 3-1

Layer 0, 2-21
logical name LSL$IF:, 1-1
LSL$IF:, 1-1 to 1-2
  and POLYGONS, 1-1

POLCHK, 2-1
POLMERGE, 4-1
  and BATCH mode, 4-1, 4-6
  and IMERGE, 4-2
  and IPOLYGON, 4-6 to 4-7
  and LSL$IF:, 4-1
  and the Bounding polygon, 4-2, 4-9
  average number of sides per polygon, 4-5
  Checks, 4-2, 4-7
  Command Format, 4-1
  command qualifiers, 4-1 to 4-2
    /ACCHECK, 4-1
    /AREA, 4-1 to 4-2
    /BOUNDING, 4-1
    /ELIMINATE, 4-1, 4-3
    /LIST, 4-1, 4-3
      default file-spec, 4-3
    /LOG, 4-1, 4-3
    /LOOKUP, 4-1, 4-3
      default file-spec, 4-3
    /MERGE, 4-1, 4-4
    /RATIO, 4-1, 4-4
  default file-specs
    input file, 4-1
    output file, 4-2
  Description, 4-6
  Elimination, 4-3, 4-7 to 4-8, 4-10
  EXAMPLES, 4-12
  File Characteristics, 4-6
  Function, 4-1
  Isolated Polygons, 4-2 to 4-3, 4-7
  Lookup File, 4-3 to 4-4, 4-6, 4-9 to 4-10
    definition set, 4-10
    primary code, 4-8, 4-10
    secondary code, 4-10
  LSL$POLYGONS_AVERAGE_SIDES, 4-5
  LSL$POLYGONS_POLMAX, 4-5
  maximum label size, 4-6
  maximum number of input segments, 4-5
  maximum number of polygons, 4-5
  Merging, 4-4, 4-6 to 4-9
  Neighbour, 4-2, 4-8 to 4-10
  Nested Polygons, 4-7 to 4-9
  Parameters, 4-1
  Polygon selection, 4-8
  Prompt, 4-1
  Restrictions, 4-5
  Tolerances, 4-2, 4-5, 4-8 to 4-9
POLMERGE command qualifiers
  /ACCHECK, 4-2
  /BOUNDING, 4-2
Polygon Overlay, 2-49
POLYGONS
  and $STATUS, 1-1
  and DCL symbol $STATUS, 1-2
  and default IFF file extension, 1-1
  and IFF, 1-1
  and the SI utility, 1-1
  and three dimensional strings, 1-2
  definition of acronym, 1-1
  features, 1-1
  getting started, 1-2
  message definitions, 1-2
  role within automated mapping, 1-1
  specification of POLYGONS command lines, 1-3
  values set in $STATUS, 1-2
POLYGONS and logical name LSL$IF:, 1-1
POLYGONS package initialisation, 1-2
POLYGONSINI, 1-2
  DCL symbols, 1-2
  in login file, 1-2

SI
  example, 1-2
SI command, 1-2
SI utility, 1-1
$STATUS, 1-1 to 1-2

SPR form for POLYGONS REF

READERS COMMENTS

Note: This form is for comments about this document only.

Did you find this manual understandable, usable and well organised? If you did not, please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and page number.

Name: **Date:**

Organisation:

SPR form for POLYGONS REF

READERS COMMENTS

Note: This form is for comments about this document only.

Did you find this manual understandable, usable and well organised? If you did not, please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and page number.

Name: **Date:**

Organisation:

SPR form for POLYGONS REF

READERS COMMENTS

Note: This form is for comments about this document only.

Did you find this manual understandable, usable and well organised? If you did not, please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and page number.

Name: **Date:**

Organisation: