NBSIR 87-3672

AMRF DATABASE REPORT FORMAT: PART MODEL

September 30, 1987

By:
Theodore H. Hopp

U.S. DEPARTMENT OF COMMERCE National Bureau of Standards Gaithersburg, Maryland

AMRF DATABASE REPORT FORMAT: PART MODEL

Theodore H. Hopp

September 30, 1987

Certain commercial equipment, instruments, or materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

TABLE OF CONTENTS

Page

I. INTRODUCTION 1
1. WHAT THIS MANUAL IS ABOUT 1
2. AUDIENCE 1
3. OVERVIEW 1

II. CONTENTS OF THE AMRF PART MODEL REPORT 3
1. TOPOLOGY AND GEOMETRY 3
2. FEATURES 4

III. FORMAL LANGUAGE SYNTAX AND SEMANTICS 4
1. FORMAL LANGUAGE SYNTAX 4
2. SEMANTICS 5
3. LEXICAL SCANNING 7

GLOSSARY 9

APPENDICES 11
APPENDIX A. TOPOLOGY AND GEOMETRY GRAMMAR 11
APPENDIX B. FEATURES GRAMMAR 29
APPENDIX C. FUNCTIONALITY GRAMMAR 34
APPENDIX D. DICTIONARY STRUCTURES 46
APPENDIX E. TYPE DEFINITIONS 49
APPENDIX F. EXAMPLES 52

REFERENCES 71

READER COMMENT FORM 72

LIST OF FIGURES

Page
Figure 1. Topology and Geometry Hierarchy of the Boundary Model 3
Figure F-1. Sketch of "CAMCLEVIS_FV" 52
Figure F-2. Sketch of "PIPECLAMP_FV" 52

LIST OF TABLES

Page
Table 1. Dictionaries Used by the Semantic Rules 6

AMRF DATABASE REPORT FORMAT: PART MODEL

I. INTRODUCTION

1. WHAT THIS MANUAL IS ABOUT

Many systems in the NBS Automated Manufacturing Research Facility (AMRF) need to manipulate part model data. Some systems, such as Design and Process Planning, must generate new models, access existing models, and update existing models. Other systems, such as the Cleaning and Deburring Workstation, only require access to existing models. Sharing of part model data is supported in the AMRF by the AMRF distributed data management system (IMDAS). Part models are stored in databases integrated with the IMDAS architecture, and these models are accessed by the application processes via queries through IMDAS. In the current architecture, a query concerning a part model, whether the query is an insert, delete, select, or update, always has an associated file containing the part model data. In the AMRF terminology, a data file associated with a query is termed a *Database Report*. This document specifies the format of part model database reports as used in the AMRF.

Part model database reports are similar in spirit to so-called "neutral" formats for geometry data such as IGES [ANSI81] or PDES [SMIT86]. AMRF part model database reports differ from IGES in that the part model reports contain mathematical model, feature, and tolerance data, while IGES files primarily contain drawing data. AMRF part model database reports are similar in content to the proposed PDES files. The AMRF part model database reports evolved before the PDES standard was available to fill a need in the AMRF. The actual data content of AMRF part model database reports and the proposed PDES files for mechanical parts appear to be quite similar. Current plans for the AMRF include studying this relationship, migrating AMRF part model support to conform to PDES, and using this experience to evaluate the conceptual models being used in PDES.

2. AUDIENCE

This document is intended for use by programmers implementing systems that make use of AMRF part model data. It is also intended as an introduction to the capabilities of the part model for systems analysts who must decide what applications can be supported by the AMRF part model. A set of software tools exist that allow application programs to easily make use of the AMRF part model database report format; these are documented elsewhere [RESS87]. The use of part model data in the AMRF is described more fully in [TU87].

3. OVERVIEW

AMRF part model data consists of basic shape data (topology and geometry), features, and functionality (tolerances). The specifics of each of these three types of part model data are discussed in Section II of this document. Section III discusses the formal language techniques used to specify the format of the database reports. Appendices A through C provide a formal definition of the various sections of the part model report. Appendices D and E provide a formal definition of the data types and data structures referenced in the formal language semantics used in Appendices A through C. Finally, Appendix F contains sample part model database reports along with sketches of the parts being modeled.

II. CONTENTS OF THE AMRF PART MODEL REPORT

1. TOPOLOGY AND GEOMETRY

The topology and geometry of parts in the AMRF is represented in a solid model boundary hierarchy. The structure of this hierarchy is shown in the following figure.

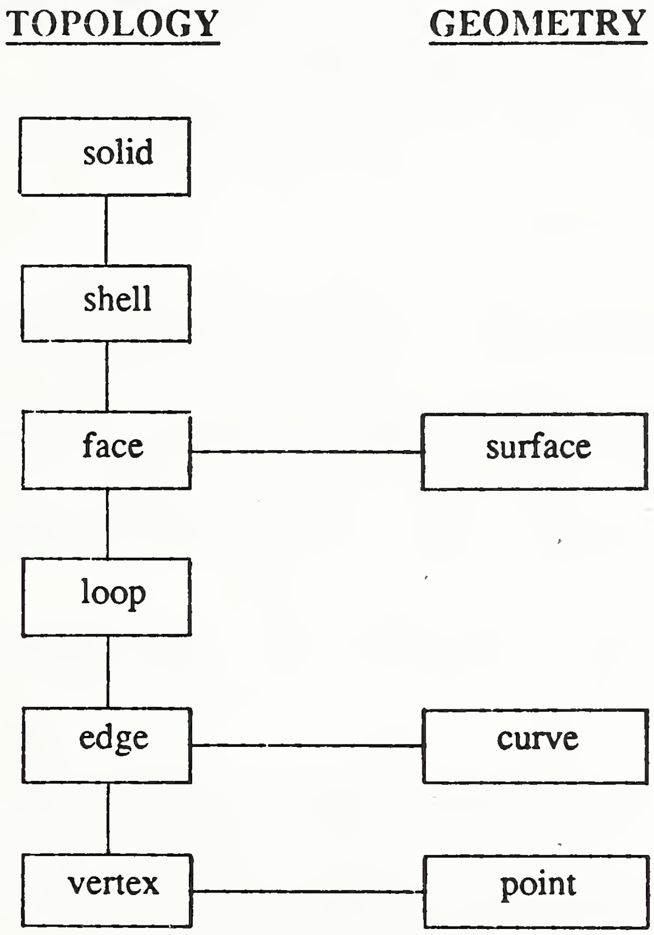

Figure 1. Topology and Geometry Hierarchy of the Boundary Model

The format implements a considerable amount of data sharing. For instance, points that represent the location of vertices can also be used in the definition of curves and surfaces.

All numeric data specified in the geometry section indicates inches unless the header section of the report contains a header entry UNITS = 'MM' . or UNITS = 'CM' ., in which case the units are millimeters or centimeters, respectively.

The topology and geometry sections of part model database reports define entities in the following dictionaries: shellDictionary, faceDictionary, loopDictionary, edgeDictionary, vertexDictionary, surfaceDictionary, curveDictionary, pointDictionary, and unitVectorDicionary.

2. FEATURES

The features section of AMRF part model reports defines part features as named collections of attribute values. Essentially arbitrary features can be defined. Attribute values are homogeneous lists of entities such as faces, edges, other features, and so forth. Heterogeneous lists of entities, such as face/edge pairs, are not directly supported, although such structures can be represented indirectly. The features section is intended to be a general feature definition facility. Several types of features being used in the AMRF have been specified in this way, and new feature types can be defined by additional formal language specifications.

Within the AMRF, functional features are collections of faces to be used in the definition of tolerances and datum reference frames. Functional features are either *simple features*, naming individual faces of a part, or *pattern features*, which are named collections of other functional features. Functional features are defined at the time the part is designed. Functional features and their definition in the part model report are described more fully elsewhere [CLAR87]. The other large class of features for which the grammar has been defined are machinable features. The definitions of specific machinable feature types and required attribute values for each type are specified in the process planning work elements for each workstation in the AMRF.

The features section of part model database reports defines entities in featureDictionary.

3. FUNCTIONALITY

The functionality section of AMRF part model reports defines the tolerance information needed for process planning and quality control. The format is capable of representing all of the tolerance control capabilities defined by the ANSI Standard for dimensioning and tolerancing [ANSI82] except datum targets, multiple datum features, and free state variation. Support of dimensioning is limited to that implied by the geometry and topology. This consists primarily of linear dimensions, with limited support of angles and form dimensioning. A system for defining the functionality supported by the AMRF part model report format has been in use for some time in the AMRF [CLAR87].

The functionality section of part model database reports defines entities in the following dictionaries: toleranceDictionary, drfDictionary, and datumDictionary.

III. FORMAL LANGUAGE SYNTAX AND SEMANTICS

1. FORMAL LANGUAGE SYNTAX

The syntactic structure of part model reports is specified by a context-free grammar. The grammar has been annotated with semantic rules that define how such reports should be interpreted when parsed. The semantic rules specify the standard AMRF interpretation for these reports. As such, these rules should be viewed as a guide rather than a requirement for how a report parser should be implemented.

The syntax rules are of the form:

left-hand-side ::= right-hand-side

where left-hand-side is a single non-terminal symbol in the grammar and right-hand-side is a sequence of one or more grammar symbols. Grammar symbols in right-hand-side that are to appear verbatim in the report appear in boldface type. All other symbols are enclosed in angle brackets <like-this>. Terminal symbols other than literals are underlined. All terminal symbols should be recognized by the lexical scanner, as described below.

There is a special grammar symbol, <null>, that represents the empty string. <null> only occurs as the right-hand-side of a grammar rule. It indicates that the left hand side can be (but need not be) eliminated from a sentential form whenever it occurs in a parse. (The context in which left-hand-side occurs will determine whether such a "null rule" should be used.) Although <null> appears to denote a terminal symbol, the lexical scanner should never report recognition of <null>. The null rule mechanism is being used as a convenience, as such rules can always be eliminated from a context-free grammar.

2. SEMANTICS

Semantic rules are associated with each reduction in the grammar. The semantic routines define how attributes for the left-hand side of a reduction should be computed from attributes of the right-hand side syntactic elements. This information can be directly translated into action rules for table-driven parsers such as those generated by the Unix[1] YACC utility. They can also be incorporated into hand-built parsers, such as recursive descent parsers, generated without the aid of a parser-generator tool.

The semantic rules have been divided into two parts. The first part, labeled **Semantics:**, defines the actions that should be taken when a particular reduction is applied during the parsing. The second part, labeled **Constraints:**, defines consistency criteria that can be used at the time a reduction is applied to check for errors in the report. Note that these criteria have not been designed to check for errors in the parsing routines themselves. It should be noted that wherever consistency criteria require a test of equality or inequality between two real numbers, the test should be sensitive to the numerical inaccuracies. Generally, a criterion of the form

$$expr1 = expr2$$

should be implemented as a test

$$|expr1 - expr2| < epsilon$$

[1] UNIX is a trademark of AT&T Bell Laboratories.

where epsilon represents a tolerance for numerical accuracy. Similar comments apply regarding inequality constraints.

The semantic rules have been designed to make use of several global dictionary data structures. Semantic rules may specify modifications to one or more dictionaries as well as specifying values for left-hand-side attributes. Such dictionary modifications should be considered side effects of the semantic rules. They considerably simplify normal semantic processing, but can make error recovery considerably more complicated than it might be. Since we anticipate parsers will not require any sort of error recovery (other than simple error detection), this was not felt to be a problem.

Dictionaries can be any sort of convenient data structure, from a linked list to a hash table to a relational database. They function as symbol tables for objects named in the report. A dictionary should be considered an array of identical elements indexed by symbolic names. The dictionaries used in the grammar are listed in Table 1.

Dictionary Name	Contents
headerDictionary	keyword/value pairs
toleranceDictionary	tolerances
drfDictionary	datum reference frames
datumDictionary	datum features
featureDictionary	feature definitions
shellDictionary	shells
faceDictionary	faces
loopDictionary	edge loops
edgeDictionary	edges
vertexDictionary	vertices
surfaceDictionary	surface descriptors
curveDictionary	curve descriptors
pointDictionary	vectors
unitVectorDictionary	unit vectors

Table 1. Dictionaries Used by the Semantic Rules

Entries in all dictionaries except headerDictionary are named entities. The name of each entity does not change from one report to another. Entities in different dictionaries for a part may have the same name. (E.g., there may be a shell s1 and a surface s1.) These are distinct entities, and the grammar unambiguously determines the type associated with each identifier in the report. The headerDictionary is used to store part attributes, such as the part name or group technology code, that are specified in the header section of part model reports.

3. LEXICAL SCANNING

Aside from literals, the only terminal symbols in this grammar are <number>, <string>, and <identifier>. The following rules describe the lexical structure of these tokens, as well as the rules concerning token separators within a report.

<number> tokens have a *val* attribute representing the value of the number. Within part model database reports, this value is always taken to be a real number. Numbers are represented according to the following conventions:

1. All numbers are in base 10.

2. Only ASCII digits **0** through **9**, a plus (+) or minus (-) sign, and a decimal point (**.**) are allowed in the representation. In particular, so-called scientific notation is not allowed.

3. If a decimal point is present, there must be digits on both sides of it. Thus, **10.0** is the number ten, while **10.** is the number ten followed by the lexical token period (**.**). The purpose of this is to avoid ambiguity in scanning numbers that might be followed by a period as a separate syntactic element (as in the definition of <point>).

4. A plus or minus sign is optional. If present, it must be the first character of the number. A plus sign is assumed if no sign is present.

5. Spaces are not allowed between any of the characters of a number; spaces represent the end of a lexical token.

<string> tokens have a *string* attribute representing the character string value of the token. Character strings are represented by ASCII printable characters enclosed in single quotes ('). The string value includes all characters between the quotes, exclusive of the quotes. A single quote can be represented in a string by two consecutive single quotes in the string. (E.g., 'I can''t do it'). All white space characters (see below) are significant inside a string. Also, a string can extend over multiple text lines.

<identifier> tokens have a *symbol* value that represents the symbolic name (usually a character string value) for that identifier. <identifier> symbol values are used to index into dictionaries, as described above. Identifiers are sequences of from 1 to 16 alphanumeric ASCII characters, the first of which must be alphabetic. Identifiers are case-sensitive. That is, the identifier **firstFace** is different than the identifier **firstface**. As a rule, all identifiers in AMRF reports will contain only upper case letters.

White space characters (ASCII space, tab, cr, and lf) can appear anywhere between lexical structures. White space is mandatory between lexical structures that might otherwise be taken as a single token (such as in **1 2 3** -- three <number> tokens that would be scanned as a single <number> were the spaces not there.) White space is ignored except within strings and as it serves to delimit lexical tokens.

Comments can be included in a database report by enclosing them in comment delimiters. The delimiter that starts a comment is the two-character sequence /* and the ending delimiter is the two-character sequence */. Comments can appear anywhere in a database report except within strings, and are treated as white space. A comment can contain arbitrary text except for the two-character ending delimiter.

GLOSSARY

AMRF - The NBS Automated Manufacturing Research Facility

context-free grammar - A grammar (q.v.) in which the set of rules specify how individual grammar symbols can be replaced by strings of grammar symbols.

database report - A file associated with a database query. Database reports are the principal means of transferring large amounts of data between IMDAS and an application program.

features - Of a part, relationships among part definition data that are meaningful to an application. Thus, for instance, form features are portions of the part shape such as pockets, grooves, or holes that are meaningful for machining.

functionality - Of a part, the data that specifies the constraints on the part imposed by the intended part function. In the AMRF part model, functionality is specified by tolerance data based on the U. S. standard for dimensioning and tolerancing (ANSI Y14.5M-1982).

geometry - For a solid, the data that specifies the geometric shape of faces, and edges, and the position of vertices.

grammar - A way of specifying a language by a set of symbols and rules for replacing symbols. The symbols may be terminals, which make up the vocabulary of the language (e.g., the words in an English dictionary), or non-terminals, which are the abstract components of the language (e.g., <noun phrase> or <verb phrase> in English). The rules specify the allowed replacement of grammar symbols by other grammar symbols. (E.g., "<noun phrase> ::= <article> <noun>" specifies that a noun phrase might be an article followed by a noun.)

IGES - The Initial Graphics Exchange Specification, a U. S. standard for exchange of drawing information between computer-aided-design systems. IGES is specified in ANSI Standard Y14.26M-1981.

IMDAS - The Integrated Manufacturing Data Administration System, a distributed, heterogeneous database environment developed as part of the AMRF.

literals - Terminal symbols (q.v.) of a grammar that may appear literally in the language.

part model data - All the data needed to define the part being produced, including shape information (topology and geometry), features, and functionality (tolerance) data.

PDES - The Product Definition Exchange Specification, an ongoing standardization effort to specify the data needed to complete define a product in terms of components, assemblies, materials, and processes. The standardization effort addresses mechanical, electrical, and architectural applications.

solid model - The specification of the shape of a (3-dimensional) solid object. A solid model is distinguished from wire-frame or surface models in that it unambiguously separates the inside of an object from the outside.

syntax - The specification of the formal structure of a language, apart from any meaning or semantics. (E.g., "Time flies the moon" is correct English syntax.)

terminal symbols - The symbols of a grammar that correspond to vocabulary elements. A terminal symbol may be a literal (q.v.) or it may be an abstraction such as <proper name> that are specified by some means other than the grammar.

tolerance - The specification of the acceptable deviation from the as-designed nominal shape. Tolerances used in the AMRF relate to dimensions or surface finish characteristics.

topology - For a solid object, the data that specifies the connectivity of the faces, edges, and vertices.

APPENDICES

APPENDIX A. TOPOLOGY AND GEOMETRY GRAMMAR

1. <part-description> ::= **/PART_MODEL**
   <header-section>
   <topology-section>
   <geometry-section>
   /END_PART_MODEL

   Semantics: This reduction is the top level goal of the parsing; when it is applied, the report has been completely parsed. The meaning of the report is contained in the various dictionaries. The name of the part is stored in headerDictionary, while the topology and geometry can be accessed by following pointers, starting with shellDictionary.

   Constraints: NONE

2. <header-section> ::= **/HEADER** <header-entry-list> **/END_HEADER**

   Semantics: NONE

   Constraints: headerDictionary[name(PART_NAME)] should exist and have a "value" attribute value that is a string.

3. <header-entry-list> ::= <header-entry>

   Semantics: NONE

   Constraints: NONE

4. <header-entry-list> ::= <header-entry> <header-entry-list>

   Semantics: NONE

   Constraints: NONE

5. <header-entry> ::= <keyword> = <keyword-value> **.**

   Semantics:
   establish headerDictionary[name(<keyword>)]

   value(headerDictionary[name(<keyword>)]) ← value(<keyword-value>)

   Constraints: headerDictionary[name(<keyword>)] should not already exist.

6. <keyword> ::= <identifier>

Semantics: name(<keyword>) ← symbol(<identifier>)

Constraints: NONE

7. <keyword-value> ::= <string>

Semantics: value(<keyword-value>) ← string(<string>)

Constraints: NONE

8. <topology-section> ::= **/TOPOLOGY**
<shell-section>
<face-section>
<loop-section>
<edge-section>
<vertex-section>
/END_TOPOLOGY

Semantics: NONE

Constraints: NONE

9. <shell-section> ::= **/SHELLS** <shells> **/END_SHELLS**

Semantics: NONE

Constraints: Each entry in shellDictionary should have a faces attribute value. There must be at least one entry in shellDictionary.

10. <shells> ::= <shell>

Semantics: NONE

Constraints: NONE

11. <shells>$_1$ ::= <shell> <shells>$_2$

Semantics: NONE

Constraints: NONE

12. <shell> ::= <shell-id> ; <face-id-list> .

Semantics:

faces(shellDictionary[name(<shell-id>)]) ← faces(<face-id-list>)

```
for each face f in faces(<face-id-list>) {
    shell(f) ← LIST(head: shellDictionary[name(<shell-id>)], tail: shell(f))
    }
```

Constraints: shellDictionary[name(<shell-id>)] should not already have a faces attribute value.

13. <face-id-list> ::= <face-id>

Semantics:

faces(<face-id-list>) ← LIST(head: faceDictionary[name(<face-id>)], tail: nil)

Constraints: NONE

14. <face-id-list>$_1$ ::= <face-id> , <face-id-list>$_2$

Semantics:

faces(<face-id-list>$_1$) ←

LIST(head: faceDictionary[name(<face-id>)], tail: faces(<face-id-list>$_2$))

Constraints: NONE

15. <face-section> ::= **/FACES** <faces> **/END_FACES**

Semantics: NONE

Constraints: Each entry in faceDictionary should have surface, sense, and loops attribute values. Each entry in faceDictionary should have a shell attribute value that is a list exactly one long.

16. <faces> ::= <face>

Semantics: NONE

Constraints: NONE

17. <faces>$_1$ ::= <face> <faces>$_2$

    Semantics: NONE

    Constraints: NONE

18. <face> ::= <face-id> ; <loop-id-list> ; <surface-tag> .

    Semantics:

    ```
    surface(faceDictionary[name(<face-id>)]) ← surface(<surface-tag>)

    sense(faceDictionary[name(<face-id>)]) ← sense(<surface-tag>)

    loops(faceDictionary[name(<face-id>)]) ← loops(<loop-id-list>)

    for each loop l in loops(<loop-id-list>) {
        face(l) ← LIST(head: faceDictionary[name(<face-id>)], tail: face(l))
        }
    ```

    Constraints: faceDictionary[name(<face-id>)] should not already have values for surface, sense, or loop attributes.

19. <face> ::= <face-id> ; ; <surface-tag> .

    Semantics:

    ```
    surface(faceDictionary[name(<face-id>)]) ← surface(<surface-tag>)

    sense(faceDictionary[name(<face-id>)]) ← sense(<surface-tag>)

    loops(faceDictionary[name(<face-id>)]) ← nil
    ```

    Constraints: faceDictionary[name(<face-id>)] should not already have values for surface, sense, or loop attributes. surface(<surface-tag>) should be a closed surface type (currently, this must be a SPHERE or TORUS).

20. <loop-id-list> ::= <loop-id>

    Semantics:

    ```
    loops(<loop-id-list>) ← LIST(head: loopDictionary[name(<loop-id>)], tail: nil)
    ```

    Constraints: NONE

21. <loop-id-list>$_1$ ::= <loop-id> , <loop-id-list>$_2$

Semantics:
loops(<loop-id-list>$_1$) ←
LIST(head: loopDictionary[name(<loop-id>)], tail: loops(<loop-id-list>$_2$))

Constraints: NONE

22. <surface-tag> ::= <surface-id> <sense>

Semantics:
surface(<surface-tag>) ← surfaceDictionary[name(<surface-id>)]

sense(<surface-tag>) ← sense(<sense>)

Constraints: NONE

23. <sense> ::= **+**

Semantics: sense(<sense>) ← POSITIVE

Constraints: NONE

24. <sense> ::= **-**

Semantics: sense(<sense>) ← NEGATIVE

Constraints: NONE

25. <loop-section> ::= **/LOOPS** <loops> **/END_LOOPS**

Semantics: NONE

Constraints: Each entry in loopDictionary should have edge and sense attribute values. Each entry in loopDictionary should have a face attribute value which is a list exactly one long.

26. <loops> ::= <loop>

Semantics: NONE

Constraints: NONE

27. <loops>$_1$ ::= <loop> <loops>$_2$

Semantics: NONE

Constraints: NONE

28. <loop> ::= <loop-id> ; <edge-tag-list> .

Semantics:

```
edges(loopDictionary[name(<loop-id>)]) ← edges(<edge-tag-list>)

senses(loopDictionary[name(<loop-id>)]) ← senses(<edge-tag-list>)

for each edge e in edges(<edge-tag-list>) {
    loops(e) ← LIST(head: loopDictionary[name(<loop-id>)], tail: loops(e))
    }
```

Constraints: NONE

29. <edge-tag-list> ::= <edge-tag>

Semantics:

```
edges(<edge-tag-list>) ← LIST(head: edge(<edge-tag>), tail: nil)

senses(<edge-tag-list>) ← LIST(head: sense(<edge-tag>), tail: nil)
```

Constraints: NONE

30. <edge-tag-list>$_1$ ::= <edge-tag> , <edge-tag-list>$_2$

Semantics:

```
edges(<edge-tag-list>$_1$) ←
    LIST(head: edge(<edge-tag>), tail: edges(<edge-tag-list>$_2$))

senses(<edge-tag-list>$_1$) ←
    LIST(head: sense(<edge-tag>), tail: senses(<edge-tag-list>$_2$))
```

Constraints: NONE

31. <edge-tag> ::= <edge-id> <sense>

Semantics:
edge(<edge-tag>) ← edgeDictionary[name(<edge-id>)]

sense(<edge-tag>) ← sense(<sense>)

Constraints: NONE

32. <edge-section> ::= **/EDGES** <edges> **/END_EDGES**

Semantics: NONE

Constraints: Each entry in edgeDictionary should have values for curve and sense attributes. Each entry in edgeDictionary should have a loops attribute value that is a list exactly two long.

33. <edges> ::= <edge>

Semantics: NONE

Constraints: NONE

34. <edges>$_1$ ::= <edge> <edges>$_2$

Semantics: NONE

Constraints: NONE

35. <edge> ::= <edge-id> ; <vertex-id-list> ; <curve-tag> .

Semantics:
curve(edgeDictionary[name(<edge-id>)]) ← curve(<curve-tag>)

sense(edgeDictionary[name(<edge-id>)]) ← sense(<curve-tag>)

vertices(edgeDictionary[name(<edge-id>)]) ← vertices(<vertex-id-list>)

```
for each vertex v in vertices(<vertex-id-list>) {
    edges(v) ← LIST(head:  edgeDictionary[name(<edge-id>)], tail: edges(v))
    }
```

Constraints: edgeDictionary[name(<edge-id>)] should not already have values for curve, sense, or vertices attributes. vertices(<vertex-id-list>) should be a list exactly two long.

36. <edge> ::= <edge-id> ; ; <curve-tag> .

Semantics:
curve(edgeDictionary[name(<edge-id>)]) ← curve(<curve-tag>)

sense(edgeDictionary[name(<edge-id>)]) ← sense(<curve-tag>)

vertices(edgeDictionary[name(<edge-id>)]) ← nil

Constraints: edgeDictionary[name(<edge-id>)] should not already have values for curve, sense, or vertices attributes. curve(<curve-tag>) must be a closed curve (currently, it must be a CIRCLE).

37. <vertex-id-list> ::= <vertex-id>

Semantics:
vertices(<vertex-id-list>) ←
LIST(head: vertexDictionary[name(<vertex-id>)], tail: nil)

Constraints: NONE

38. <vertex-id-list>$_1$ ::= <vertex-id> , <vertex-id-list>$_2$

Semantics:
vertices(<vertex-id-list>$_1$) ←
LIST(head: vertexDictionary[name(<vertex-id>)],
tail: vertices(<vertex-id-list>$_2$))

Constraints: NONE

39. <curve-tag> ::= <curve-id> <sense>

Semantics:
curve(<curve-tag>) ← curveDictionary[name(<curve-id>)]

sense(<curve-tag>) ← sense(<sense>)

Constraints: NONE

40. <vertex-section> ::= **/VERTICES** <vertices> **/END_VERTICES**

Semantics: NONE

Constraints: Each entry in vertexDictionary should have a point attribute value. Each entry in vertexDictionary should have an edges attribute value that is a list at least 3 long.

41. <vertices> ::= <vertex>

Semantics: NONE

Constraints: NONE

42. <vertices>$_1$ ::= <vertex> <vertices>$_2$

Semantics: NONE

Constraints: NONE

43. <vertex> ::= <vertex-id> ; <point-id> .

Semantics:
point(vertexDictionary[name(<vertex-id>)]) ← pointDictionary[name(<point-id>)]

Constraints: vertexDictionary[name(<vertex-id>)] should not already have a point attribute value.

44. <geometry-section> ::= **/GEOMETRY**
<surface-section>
<curve-section>
<point-section>
<unit-vector-section>
/END_GEOMETRY

Semantics: NONE

Constraints: NONE

45. <surface-section> ::= **/SURFACES** <surfaces> **/END_SURFACES**

Semantics: NONE

Constraints: Each entry in surfaceDictionary should have a surface attribute value.

46. <surfaces> ::= <surface>

Semantics: NONE

Constraints: NONE

47. <surfaces>$_1$ ::= <surface> <surfaces>$_2$

Semantics: NONE

Constraints: NONE

48. <surface> ::= <surface-id> ; <surface-description> .

Semantics:
surface(surfaceDictionary[name(<surface-id>)]) ← surface(<surface-description>)

Constraints: surfaceDictionary[name(<surface-id>)] should not already have a surface attribute value.

49. <surface-description> ::= **PLANE ;** <unit-vector-id> **;** <number>

Semantics:
surface(<surface-description>) ←
PLANAR_SURFACE(normal: unitVectorDictionary[name(<unit-vector-id>)],
distance: val(<number>))

Note: The positive direction of the plane is along <unit-vector-id>.

Constraints: NONE

50. <surface-description> ::= **CYLINDER** ; <point-id> ; <unit-vector-id> ; <number>

Semantics:

```
surface(<surface-description>) ← CYLINDRICAL_SURFACE(
        axis-point: pointDictionary[name(<point-id>)],
              axis: unitVectorDictionary[name(<unit-vector-id>)],
            radius: val(<number>))
```

Note: The positive direction of the cylinder is from the surface toward the axis.

Constraints: <number> > 0

51. <surface-description> ::= **CONE** ; <point-id> ; <unit-vector-id> ; <number>

Semantics:

```
surface(<surface-description>) ←
    CONICAL_SURFACE(vertex: pointDictionary[name(<point-id>)],
                      axis: unitVectorDictionary[name(<unit-vector-id>)],
         cone-angle-cosine: val(<number>))
```

Note: The positive direction of the cone is from the surface toward the axis.

Constraints: 0 < val(<number>) < 1

52. <surface-description> ::= **SPHERE** ; <point-id> ; <number>

Semantics:

```
  surface(<surface-description>) ←
SPHERICAL_SURFACE(center: pointDictionary[name(<point-id>)],
                  radius: val(<number>))
```

Note: The positive direction of the sphere is from the surface toward the center.

Constraints: val(<number>) > 0

53. <surface-description> ::=
TORUS ; <point-id> ; <unit-vector-id> ; <number>$_1$; <number>$_2$

Semantics:
surface(<surface-description>) ← TOROIDAL_SURFACE(
center: pointDictionary[name(<point-id>)],
axis: unitVectorDictionary[name(<unit-vector-id>)],
major-radius: val(<number>$_1$)
minor-radius: val(<number>$_2$))

Note: The positive direction of the torus is from the surface toward the major circle.

Constraints: val(<number>$_1$) > val(<number>$_2$) > 0

54. <curve-section> ::= **/CURVES** <curves> **/END_CURVES**

Semantics: NONE

Constraints: All entries in curveDictionary should have a curve attribute value.

55. <curves> ::= <curve>

Semantics: NONE

Constraints: NONE

56. <curves>$_1$ ::= <curve> <curves>$_2$

Semantics: NONE

Constraints: NONE

57. <curve> ::= <curve-id> ; <curve-description> .

Semantics:
curve(curveDictionary[name(<curve-id>)]) ← curve(<curve-description>)

Constraints: curveDictionary[name(<curve-id>)] should not already have a curve attribute value.

58. <curve-description> ::= **LINE** ; <point-id> ; <unit-vector-id>

Semantics:

curve(<curve-description>) ←
    LINEAR_CURVE(start: pointDictionary[name(<vector>)],
                  direction: unitVectorDictionary[name(<unit-vector-id>)])

Note: The positive direction is along the unit vector identified by <unit-vector-id>.

Constraints: NONE

59. <curve-description> ::= **CIRCLE** ; <point-id>$_1$; <unit-vector-id> ; <point-id>$_2$

Semantics:

curve(<curve-description>) ←
    CIRCULAR_CURVE(center: pointDictionary[name(<point-id>$_1$)],
                   axis: unitVectorDictionary[name(<unit-vector-id>)],
                   start: pointDictionary[name(<point-id>$_2$)])

Note: The positive direction is defined to be a right-hand rule. That is, the positive direction is clockwise when viewing the circle along the direction identified by <unit-vector-id>.

Constraints: The vector from <point-id>$_1$ to <point-id>$_2$ should be orthogonal to <unit-vector-id>. That is, if

v_1 = vec(pointDictionary[name(<point-id>$_1$)]),

v_2 = vec(pointDictionary[name(<point-id>$_2$)]),

and

d = dir(unitVectorDictionary[name(<unit-vector-id>)])

then

$$(v_2 - v_1) \circ d = 0$$

60. <point-section> ::= **/POINTS** <points> **/END_POINTS**

Semantics: NONE

Constraints: All entries in pointDictionary should have a vec attribute value.

61. <points> ::= <point>

Semantics: NONE

Constraints: NONE

62. <points>$_1$ ::= <point> <points>$_2$

Semantics: NONE

Constraints: NONE

63. <point> ::= <point-id> ; <vector> .

Semantics:
vec(pointDictionary[name(<point-id>)]) ← vec(<vector>)

Constraints: pointDictionary[name(<point-id>)] should not already have a vec attribute value.

64. <vector> ::= <number>$_X$, <number>$_Y$, <number>$_Z$

Semantics:
vec(<vector>) ← VECTOR(x: val(<number>$_X$),
y: val(<number>$_Y$),
z: val(<number>$_Z$))

Constraints: NONE

65. <unit-vector-section> ::= **/UNIT_VECTORS** <unit-vectors> **/END_UNIT_VECTORS**

Semantics: NONE

Constraints: All entries in unitVectorDictionary should have a dir attribute value.

66. <unit-vectors> ::= <unit-vector>

Semantics: NONE

Constraints: NONE

67. <unit-vectors>$_1$ ::= <unit-vector-entry> <unit-vectors>$_2$

Semantics: NONE

Constraints: NONE

68. <unit-vector-entry> ::= <unit-vector-id> ; <unit-vector> .

Semantics:
dir(unitVectorDictionary[name(<unit-vector-id>)]) ← dir(<unit-vector>)

Constraints: unitVectorDictionary[name(<unit-vector-id>)] should not already have a dir attribute value.

69. <unit-vector> ::= <number>$_X$, <number>$_Y$, <number>$_Z$

Semantics:
dir(<unit-vector>) ← UNIT_VECTOR(x: val(<number>$_X$),
y: val(<number>$_Y$),
z: val(<number>$_Z$))

Constraints:
val(<number>$_X$)^2 + val(<number>$_Y$)^2 + val(<number>$_Z$)^2 = 1

70. <shell-id> ::= <identifier>

Semantics:
name(<shell-id>) ← symbol(<identifier>)

If shellDictionary[name(<shell-id>)] does not exist:

establish shellDictionary[name(<shell-id>)]

name(shellDictionary[name(<shell-id>)]) ← name(<shell-id>)

Constraints: NONE

71. <face-id> ::= <identifier>

Semantics:
name(<face-id>) ← symbol(<identifier>)

If faceDictionary[name(<face-id>)] does not exist:

establish faceDictionary[name(<face-id>)]

name(faceDictionary[name(<face-id>)]) ← name(<face-id>)

shell(faceDictionary[name(<face-id>)]) ← nil

Constraints: NONE

72. <loop-id> ::= <identifier>

Semantics:
name(<loop-id>) ← symbol(<identifier>)

If loopDictionary[name(<loop-id>)] does not exist:

establish loopDictionary[name(<loop-id>)]

name(loopDictionary[name(<loop-id>)]) ← name(<loop-id>)

face(loopDictionary[name(<loop-id>)]) ← nil

Constraints: NONE

73. <edge-id> ::= <identifier>

Semantics:
name(<edge-id>) ← symbol(<identifier>)

If edgeDictionary[name(<edge-id>)] does not exist:

establish edgeDictionary[name(<edge-id>)]

name(edgeDictionary[name(<edge-id>)]) ← name(<edge-id>)

loops(edgeDictionary[name(<edge-id>)]) ← nil

Constraints: NONE

74. <vertex-id> ::= <identifier>

Semantics:
name(<vertex-id>) ← symbol(<identifier>)

If vertexDictionary[name(<vertex-id>)] does not exist:

establish vertexDictionary[name(<vertex-id>)]

name(vertexDictionary[name(<vertex-id>)]) ← name(<vertex-id>)

edges(vertexDictionary[name(<vertex-id>)]) ← nil

Constraints: NONE

75. <surface-id> ::= <identifier>

Semantics:
name(<surface-id>) ← symbol(<identifier>)

If surfaceDictionary[name(<surface-id>)] does not exist:

establish surfaceDictionary[name(<surface-id>)]

name(surfaceDictionary[name(<surface-id>)]) ← name(<surface-id>)

Constraints: NONE

76. <curve-id> ::= <identifier>

Semantics:
name(<curve-id>) ← symbol(<identifier>)

If curveDictionary[name(<curve-id>)] does not exist:

establish curveDictionary[name(<curve-id>)]

name(curveDictionary[name(<curve-id>)]) ← name(<curve-id>)

Constraints: NONE

77. <point-id> ::= <identifier>

Semantics:

name(<point-id>) ← symbol(<identifier>)

If pointDictionary[name(<point-id>)] does not exist:

establish pointDictionary[name(<point-id>)]

name(pointDictionary[name(<point-id>)]) ← name(<point-id>)

Constraints: NONE

78. <unit-vector-id> ::= <identifier>

Semantics:

name(<unit-vector-id>) ← symbol(<identifier>)

If unitVectorDictionary[name(<unit-vector-id>)] does not exist:

establish unitVectorDictionary[name(<unit-vector-id>)]

name(unitVectorDictionary[name(<unit-vector-id>)]) ← name(<unit-vector-id>)

Constraints: NONE

APPENDIX B. FEATURES GRAMMAR

1. <part-description> ::= **/PART_MODEL**
   <header-section>
   <features-section>
   <topology-section>
   <geometry-section>
   /END_PART_MODEL

   Semantics: This reduction is the top level goal of the parsing; when it is applied, the report has been completely parsed. The meaning of the report is completely contained in the various dictionaries. The name of the part is stored in headerDictionary, while topology, geometry, and features data can be accessed by following pointers, starting with shellDictionary, toleranceDictionary, and featureDictionary.

   Constraints: NONE

2. <features-section> ::= **/FEATURES** <features> **/END_FEATURES**

   Semantics: NONE

   Constraints: NONE

3. <features> ::= <feature>

   Semantics: NONE

   Constraints: NONE

4. <features>$_1$ ::= <feature> <features>$_2$

   Semantics: NONE

   Constraints: NONE

5. <feature> ::= <feature-id> ; <feature-type> ; <feature-attributes> .

Semantics:
type(featureDictionary[name(<feature-id>)]) ← type(<feature-type>)

atts(featureDictionary[name(<feature-id>)]) ← atts(<feature-attributes>)

Constraints: Constraints and additional semantics may be defined by the application, based on the feature type. For instance, if <feature-type> is the token **SIMPLE**, then <feature-attributes> should consist of a single attribute, a list of faces (attribute key FACES) exactly one long.

6. <feature-type> ::= <identifier>

Semantics: type(<feature-type>) ← symbol(<identifier>)

Constraints: NONE

7. <feature-attributes> ::= <feature-attribute>

Semantics:
atts(<feature-attributes>) ← LIST(head: att(<feature-attribute>), tail: nil)

Constraints: NONE

8. <feature-attributes>$_1$ ::= <feature-attribute> ; <feature-attributes>$_2$

Semantics:
atts(<feature-attributes>$_1$) ←
LIST(head: att(<feature-attribute>), tail: atts(<feature-attributes>$_2$))

Constraints: key(att(<feature-attribute>)) should not already occur as the key of any member of atts(<feature-attributes>$_2$).

9. <feature-attribute> ::= <feature-attribute-keyword> = <feature-attribute-value-list>

Semantics:
Let

```
v ← INTERPRET(vals: values(<feature-attribute-value-list>),
                    as: attrib(<feature-attribute-keyword>))
```

then

```
att(<feature-attribute>) ←
    ATT(key: attrib(<feature-attribute-keyword>), val: v)
```

Constraints: All attribute values in <feature-attribute-value-list> must be interpretable as values of the type indicated by <feature-attribute-keyword>. If <feature-attribute-keyword> is the name of a topology or geometry section of the part model report (one of **SHELLS**, **FACES**, **LOOPS**, **EDGES**, **VERTICES**, **SURFACES**, **CURVES**, **POINTS**, or **UNIT_VECTORS**), then each element of values(<feature-attribute-value-list>) must be either nil or an identifier naming an entity of the indicated type. The INTERPRET function maps the identifier symbol value into the dictionary entry for that entity. Similar comments apply if <feature-attribute-keyword> names a section in the functionality section (**TOLERANCES**, **DRFS**, or **DATUMS**), or names the **FEATURES** section. Otherwise, the values in <feature-attribute-value-list> are unrestricted.

10. <feature-attribute-keyword> ::= <identifier>

Semantics: attrib(<feature-attribute-keyword>) ← symbol(<identifer>)

Constraints: NONE

11. <feature-attribute-value-list> ::= <feature-attribute-value>

Semantics:

```
values(<feature-attribute-value-list>) ←
    LIST(head: value(<feature-attribute-value>), tail: nil)
```

Constraints: NONE

12. <feature-attribute-value-list>$_1$ ::=
<feature-attribute-value> , <feature-attribute-value-list>$_2$

Semantics:
values(<feature-attribute-value-list>$_1$) ←
LIST(head: value(<feature-attribute-value>),
tail: values(<feature-attribute-value-list>$_2$))

Constraints: NONE

13. <feature-attribute-value> ::= <identifier>

Semantics: value(<feature-attribute-value>) ← symbol(<identifier>)

Constraints: NONE

14. <feature-attribute-value> ::= <string>

Semantics: value(<feature-attribute-value>) ← string(<string>)

Constraints: NONE

15. <feature-attribute-value> ::= <number>

Semantics: value(<feature-attribute-value>) ← val(<number>)

Constraints: NONE

16. <feature-attribute-value> ::= <sense>

Semantics: value(<feature-attribute-value>) ← sense(<sense>)

Constraints: NONE

17. <feature-attribute-value> ::= <null>

Semantics: value(<feature-attribute-value>) ← nil

Constraints: NONE

18. <feature-id> ::= <identifier>

Semantics:

```
name(<feature-id>) ← symbol(<identifier>)

If featureDictionary[name(<feature-id>)] does not exist:

    establish featureDictionary[name(<feature-id>)]

    name(featureDictionary[name(<feature-id>)]) ← name(<feature-id>)
```

Constraints: NONE

APPENDIX C. FUNCTIONALITY GRAMMAR

1. <part-description> ::= **/PART_MODEL**
<header-section>
<functionality-section>
<features-section>
<topology-section>
<geometry-section>
/END_PART_MODEL

Semantics: This reduction is the top level goal of the parsing; when it is applied, the report has been completely parsed. The meaning of the report is completely contained in the various dictionaries. The name of the part is stored in headerDictionary, while topology, geometry, features, and functionality data can be accessed by following pointers, starting with shellDictionary, toleranceDictionary, and featureDictionary.

Constraints: NONE

2. <functionality-section> ::= **/FUNCTIONALITY**
<tolerances-section>
<drf-section>
<datum-section>
/END_FUNCTIONALITY

Semantics: NONE

Constraints: NONE

3. <tolerances-section> ::= **/TOLERANCES** <tolerances> **/END_TOLERANCES**

Semantics: NONE

Constraints: NONE

4. <tolerances> ::= <tolerance>

Semantics: NONE

Constraints: NONE

5. $\langle$tolerances$\rangle_1$ ::= <tolerance> $\langle$tolerances$\rangle_2$

   Semantics: NONE

   Constraints: NONE

6. <tolerance> ::= <tolerance-id> ; <controlled-feature> ; <tolerance-description> .

   Semantics:
   feature(toleranceDictionary[name(<tolerance-id>)]) ←
   feature(<controlled-feature>)

   mods(toleranceDictionary[name(<tolerance-id>)]) ← mods(<controlled-feature>)

   tol(toleranceDictionary[name(<tolerance-id>)]) ← tol(<tolerance-description>)

   Constraints: toleranceDictionary[name(<tolerance-id>)] should not already have feature, mods, or tol attribute values.

7. <controlled-feature> ::= <feature-id>

   Semantics:
   feature(<controlled-feature>) ← featureDictionary[name(<feature-id>)]

   mods(<controlled-feature>) ← nil

   Constraints: NONE

8. <controlled-feature> ::= <feature-id> , <modifiers>

   Semantics:
   feature(<controlled-feature>) ← featureDictionary[name(<feature-id>)]

   mods(<controlled-feature>) ← mods(<modifiers>)

   Constraints: NONE

9. <modifiers> ::= <modifier>

   Semantics:
   mods(<modifiers>) ← LIST(head: mod(<modifier>), tail: nil)

   Constraints: NONE

10. <modifiers>$_1$ ::= <modifier> , <modifiers>$_2$

    Semantics:
    mods(<modifiers>$_1$) ← LIST(head: mod(<modifier>), tail: mods(<modifiers>$_2$))

    Constraints: mods(<modifiers>$_1$) must have at most one occurrence each of a CONDITION, PROJECT, or DIMENSION modifier attribute.

11. <modifier> ::= <material-condition>

    Semantics: mod(<modifier>) ← mod(<material-condition>)

    Constraints: NONE

12. <modifier> ::= <projected-tolerance-zone>

    Semantics: mod(<modifier>) ← mod(<projected-tolerance-zone>)

    Constraints: NONE

13. <modifier> ::= <dimensional-indicator>

    Semantics: mod(<modifier>) ← mod(<dimensional-indicator>)

    Constraints: NONE

14. <material-condition> ::= **MMC**

    Semantics: mod(<material-condition>) ← CONDITION("MMC")

    Constraints: NONE

15. <material-condition> ::= **LMC**

    Semantics: mod(<material-condition>) ← CONDITION("LMC")

    Constraints: NONE

16. <material-condition> ::= **RFS**

    Semantics: mod(<material-condition>) ← CONDITION("RFS")

    Constraints: NONE

17. <projected-tolerance-zone> ::= **P** = <number>

    Semantics: mod(<projected-tolerance-zone>) ← PROJECT(val(<number>))

    Constraints: val(<number>) > 0

18. <dimensional-indicator> ::= **R**

    Semantics: mod(<dimensional-indicator>) ← DIMENSION("RADIUS")

    Constraints: NONE

19. <dimensional-indicator> ::= **SR**

    Semantics: mod(<dimensional-indicator>) ← DIMENSION("SPHERICAL RADIUS")

    Constraints: NONE

20. <dimensional-indicator> ::= **DIA**

    Semantics: mod(<dimensional-indicator>) ← DIMENSION("DIAMETER")

    Constraints: NONE

21. <dimensional-indicator> ::= **SDIA**

    Semantics: mod(<dimensional-indicator>) ← DIMENSION("SPHERICAL DIAMETER")

    Constraints: NONE

22. <tolerance-description> ::= <intrinsic-characteristic> ; <limits>

Semantics:

```
tol(<tolerance-description>) ←
    IN_TOL(char: char(<intrinsic-characteristic>), limits: limits(<limits>))
```

Constraints: If char(<intrinsic-characteristic>) is "SIZE", "LIMIT", or "PLUS_MINUS", limits(<limits>) should be a list exactly two long. All other characteristics require the limits to be a list of exactly one non-negative value.

23. <tolerance-description> ::= <extrinsic-characteristic> ; <limits> ; <drf-id>

Semantics:

```
tol(<tolerance-description>) ←
   EX_TOL(char: char(<extrinsic-characteristic>),
              limits: limits(<limits>),
                 drf: drfDictionary[name(<drf-id>)])
```

Constraints: limits(<limits>) should be a list of exactly one non-negative value.

24. <intrinsic-characteristic> ::= **SIZE**

Semantics: char(<intrinsic-characteristic>) ← CHAR("SIZE")

Constraints: NONE

25. <intrinsic-characteristic> ::= **STRAIGHTNESS**

Semantics: char(<intrinsic-characteristic>) ← CHAR("STRAIGHTNESS")

Constraints: NONE

26. <intrinsic-characteristic> ::= **FLATNESS**

Semantics: char(<intrinsic-characteristic>) ← CHAR("FLATNESS")

Constraints: NONE

27. <intrinsic-characteristic> ::= **CIRCULARITY**

Semantics: char(<intrinsic-characteristic>) ← CHAR("CIRCULARITY")

Constraints: NONE

28. <intrinsic-characteristic> ::= **CYLINDRICITY**

Semantics: char(<intrinsic-characteristic>) ← CHAR("CYLINDRICITY")

Constraints: NONE

29. <intrinsic-characteristic> ::= **LIMIT**

Semantics: char(<intrinsic-characteristic>) ← CHAR("LIMIT")

Constraints: NONE

30. <intrinsic-characteristic> ::= **PLUS_MINUS**

Semantics: char(<intrinsic-characteristic>) ← CHAR("PLUS-MINUS")

Constraints: NONE

31. <intrinsic-characteristic> ::= **INTRINSIC_LINE_PROFILE**

Semantics: char(<intrinsic-characteristic>) ← CHAR("INTRINSIC_LINE_PROFILE")

Constraints: NONE

32. <intrinsic-characteristic> ::= **INTRINSIC_SURFACE_PROFILE**

Semantics:
char(<intrinsic-characteristic>) ← CHAR("INTRINSIC_SURFACE_PROFILE")

Constraints: NONE

33. <extrinsic-characteristic> ::= **EXTRINSIC_LINE_PROFILE**

Semantics: char(<extrinsic-characteristic>) ← CHAR("EXTRINSIC_LINE_PROFILE")

Constraints: NONE

34. <extrinsic-characteristic> ::= **EXTRINSIC_SURFACE_PROFILE**

    Semantics:
    char(<extrinsic-characteristic>) ← CHAR("EXTRINSIC_SURFACE_PROFILE")

    Constraints: NONE

35. <extrinsic-characteristic> ::= **ANGULARITY**

    Semantics: char(<extrinsic-characteristic>) ← CHAR("ANGULARITY")

    Constraints: NONE

36. <extrinsic-characteristic> ::= **PERPENDICULARITY**

    Semantics: char(<extrinsic-characteristic>) ← CHAR("PERPENDICULARITY")

    Constraints: NONE

37. <extrinsic-characteristic> ::= **PARALLELISM**

    Semantics: char(<extrinsic-characteristic>) ← CHAR("PARALLELISM")

    Constraints: NONE

38. <extrinsic-characteristic> ::= **CONCENTRICITY**

    Semantics: char(<extrinsic-characteristic>) ← CHAR("CONCENTRICITY")

    Constraints: NONE

39. <extrinsic-characteristic> ::= **CIRCULAR_RUNOUT**

    Semantics: char(<extrinsic-characteristic>) ← CHAR("CIRCULAR_RUNOUT")

    Constraints: NONE

40. <extrinsic-characteristic> ::= **TOTAL_RUNOUT**

Semantics: char(<extrinsic-characteristic>) ← CHAR("TOTAL_RUNOUT")

Constraints: NONE

41. <extrinsic-characteristic> ::= **POSITION**

Semantics: char(<extrinsic-characteristic>) ← CHAR("POSITION")

Constraints: NONE

42. <limits> ::= <number>

Semantics: limits(<limits>) ← LIST(head: val(<number>), tail: nil)

Constraints: NONE

43. <limits> ::= <number>$_1$, <number>$_2$

Semantics:
limits(<limits>) ← LIST(head: val(<number>$_1$),
tail: LIST(head: val(<number>$_2$), tail: nil))

Constraints: NONE

44. <drf-section> ::= **/DRFS** <drfs> **/END_DRFS**

Semantics: NONE

Constraints: NONE

45. <drfs> ::= <drf>

Semantics: NONE

Constraints: NONE

46. <drfs>$_1$ ::= <drf> <drfs>$_2$

Semantics: NONE

Constraints: NONE

47. <drf> ::= <drf-id> ; <datum-features> .

Semantics: drf(drfDictionary[name(<drf-id>)]) ← drf(<datum-features>)

Constraints: drfDictionary[name(<drf-id>] should not already have a drf attribute value.

48. <datum-features> ::= <datum-feature>

Semantics:

drf(<datum-features>) ←

DRF(primary: datum(<datum-feature>), secondary: nil, tertiary: nil)

Constraints: NONE

49. <datum-features> ::= <datum-feature>$_1$; <datum-feature>$_2$

Semantics:

drf(<datum-features>) ←

DRF(primary: datum(<datum-feature>$_1$),

secondary: datum(<datum-feature>$_2$),

tertiary: nil)

Constraints: NONE

50. <datum-features> ::= <datum-feature>$_1$; <datum-feature>$_2$; <datum-feature>$_3$

Semantics:

drf(<datum-features>) ←

DRF(primary: datum(<datum-feature>$_1$),

secondary: datum(<datum-feature>$_2$),

tertiary: datum(<datum-feature>$_3$))

Constraints: NONE

51. <datum-feature> ::= <datum-id>

Semantics:

```
datum(<datum-feature>) ←
    DATUM_FEATURE(datum: datumDictionary[name(<datum-id>)], mod: nil)
```

Constraints: NONE

52. <datum-feature> ::= <datum-id> , <material-condition>

Semantics:

```
datum(<datum-feature>) ← DATUM_FEATURE(
        datum: datumDictionary[name(<datum-id>)],
          mod: mod(<material-condition>))
```

Constraints: NONE

53. <datum-section> ::= **/DATUMS** <datums> **/END_DATUMS**

Semantics: NONE

Constraints: NONE

54. <datums> ::= <datum>

Semantics: NONE

Constraints: NONE

55. <datums>_1 ::= <datum> <datums>_2

Semantics: NONE

Constraints: NONE

56. <datum> ::= <datum-id> ; <identifier> ; <feature-id> .

Semantics:
id(datumDictionary[name(<datum-id>)]) ← name(<identifier>)

feature(datumDictionary[name(<datum-id>)]) ←
featureDictionary[name(<feature-id>)]

Constraints: No other entry in datumDictionary should have an id attribute value of name(<identifier>).

57. <tolerance-id> ::= <identifier>

Semantics:
name(<tolerance-id>) ← symbol(<identifier>)

If toleranceDictionary[name(<tolerance-id>)] does not exist:

establish toleranceDictionary[name(<tolerance-id>)]

name(toleranceDictionary[name(<tolerance-id>)]) ← name(<tolerance-id>)

Constraints: NONE

58. <drf-id> ::= <identifier>

Semantics:
name(<drf-id>) ← symbol(<identifier>)

If drfDictionary[name(<drf-id>)] does not exist:

establish drfDictionary[name(<drf-id>)]

name(drfDictionary[name(<drf-id>)]) ← name(<drf-id>)

Constraints: NONE

59. <datum-id> ::= <identifier>

Semantics:

```
name(<datum-id>) ← symbol(<identifier>)

If datumDictionary[name(<datum-id>)] does not exist:

    establish datumDictionary[name(<datum-id>)]

    name(datumDictionary[name(<datum-id>)]) ← name(<datum-id>)
```

Constraints: NONE

APPENDIX D. DICTIONARY STRUCTURES

headerDictionary

index: header keyword string
contents:

value: header keyword value string

toleranceDictionary

index: tolerance id string
contents:

name: tolerance id string
feature: controlled feature
mods: list of feature modifiers
tol: tolerance description

A feature data structure is an entry in featureDictionary. A feature modifier can be one of CONDITION, PROJECT, or DIMENSION data structures. A tolerance description can be one of IN_TOL (intrinsic tolerance) or EX_TOL (extrinsic tolerance). An IN_TOL has a tolerance characteristic and limits, and an EX_TOL has, in addition, a datum reference frame.

drfDictionary

index: datum reference frame id string
contents:

name: datum reference frame id string
drf: datum reference frame

A datum reference frame is a data structure of type DRF.

datumDictionary

index: datum id string
contents:

name: datum id string
id: datum name string
feature: a feature that is the datum

A feature data structure is an entry in featureDictionary.

featureDictionary

index: feature id string
contents:

name: feature id string
type: feature type string
atts: list of feature attributes

A feature attribute is an ATT structure.

shellDictionary

index: shell id string

contents:

name: shell id string

faces: list of faces

A face data structure is an entry in faceDictionary.

faceDictionary

index: face id string

contents:

name: face id string

surface: a surface

sense: sense tag

loops: list of loops

shell: list of shells

A surface data structure is an entry in surfaceDictionary. A sense tag is a SENSE. A loop data structure is an entry in loopDictionary. A shell data structure is an entry in shellDictionary.

loopDictionary

index: loop id string

contents:

name: loop id string

edges: list of edges

senses: list of sense tags

face: list of faces

An edge data structure is an entry in edgeDictionary. A sense tag is a SENSE. A face data structure is an entry in faceDictionary.

edgeDictionary

index: edge id string

contents:

name: edge id string

curve: a curve

sense: a sense tag

vertices: a list of vertices

loops: a list of loops

A curve data structure is an entry in curveDictionary. A sense tag is a SENSE. A vertex data structure is an entry in vertexDictionary. A loop data structure is an entry in loopDIctionary.

vertexDictionary

index: vertex id string
contents:

name: vertex id string
point: a point
edges: a list of edges

A point data structure is an entry in pointDictionary. An edge data structure is an entry in edgeDictionary.

surfaceDictionary

index: surface id string
contents:

name: surface id string
surface: a surface description

A surface description can be one of PLANAR_SURFACE, CYLINDRICAL_SURFACE, CONICAL_SURFACE, SPHERICAL_SURFACE, or TOROIDAL_SURFACE data structures.

curveDictionary

index: curve id string
contents:

name: curve id string
curve: a curve description

A curve description can be one of LINEAR_CURVE or CIRCULAR_CURVE data structures.

pointDictionary

index: point id string
contents:

name: point id string
vec: a vector

A vector is a VECTOR data structure.

unitVectorDictionary

index: unit vector id string
contents:

name: unit vector id string
dir: a direction

A direction is a UNIT_VECTOR data structure.

APPENDIX E. TYPE DEFINITIONS

CONDITION

Enumeration type, one of "MMC", "LMC", or "RFS"

PROJECT

A real number

DIMENSION

Enumeration type, one of "RADIUS", "SPHERICAL RADIUS", "DIAMETER", or "SPHERICAL DIAMETER"

IN_TOL

A structure with two fields:

char: an intrinsic tolerance characteristic
limits: a list of limits

An intrinsic tolerance characteristic is an enumeration type, one of "SIZE", "STRAIGHTNESS", "FLATNESS", "CIRCULARITY", "CYLINDRICITY", "LIMIT", "PLUS-MINUS", "INTRINSIC LINE PROFILE", or "INTRINSIC SURFACE PROFILE". A limit is a number.

EX_TOL

A structure with three fields:

char: an extrinsic tolerance characteristic
limits: a list of limits
drf: a datum reference frame

An extrinsic tolerance characteristic is an enumeration type, one of "EXTRINSIC LINE PROFILE", "EXTRINSIC SURFACE PROFILE", "ANGULARITY", "PERPENDICULARITY", "PARALLELISM", "CONCENTRICITY", "CIRCULAR RUNOUT", "TOTAL RUNOUT", or "POSITION". A limit is a number. A datum reference frame data structure is an entry in drfDictionary.

DRF

A structure with three fields:

primary: a datum feature
secondary: a datum feature
tertiary: a datum feature

A datum feature data structure is a structure with two fields:

datum: a datum
mod: a CONDITION

A datum data structure is an entry in datumDictionary.

ATT

A structure with two fields:

key: an attribute keyword string
val: a list of attribute value interpretations

An attribute value interpretation is the result of interpreting an attribute value in terms of a keyword string. An attribute value type is the union of symbol, string, number, sense, or null. An interpretation type is the union of the attribute value types and the entries in shellDictionary, faceDictionary, loopDictionary, edgeDictionary, vertexDictionary, surfaceDictionary, curveDictionary, pointsDictionary, unitVectorDictionary, toleranceDictionary, drfDictionary, or datumDictionary.

PLANAR_SURFACE

A structure with two fields:

normal: a direction
distance: a number

A distance data structure is an entry in unitVectorDictionary

CYLINDRICAL_SURFACE

A structure with three fields:

axis-point: a point
axis: a direction
radius: a number

A vector data structure is an entry in pointDictionary. A direction data structure is an entry in unitVectorDictionary.

CONICAL_SURFACE

A structure with three fields:

vertex: a point
axis: a direction
cone-angle-cosine: a number

A point data structure is an entry in pointDictionary. A direction data structure is an entry in unitVectorDictionary.

SPHERICAL_SURFACE

A structure with two fields:

center: a point
radius: a number

A point data structure is an entry in pointDictionary.

TOROIDAL_SURFACE

A structure with four fields:

center: a point
axis: a direction
major-radius: a number
minor-radius: a number

A point data structure is an entry in pointDictionary. A direction data structure is an entry in unitVectorDictionary.

SENSE

An enumeration type, one of POSITIVE or NEGATIVE.

LINEAR_CURVE

A structure with two fields:

start: a point
direction: a direction

A point data structure is an entry in pointDictionary. A direction data structure is an entry in unitVectorDictionary.

CIRCULAR_CURVE

A structure with three fields:

center: a point
axis: a direction
start: a point

A point data structure is an entry in pointDictionary. A direction data structure is an entry in unitVectorDictionary.

VECTOR

A structure with three fields:

x : a number
y : a number
z : a number

UNIT_VECTOR

A structure with three fields:

x : a number
y : a number
z : a number

APPENDIX F. EXAMPLES

The following pages contain listings of the database report for two parts. Both parts have been made in the AMRF, and the part model database report has been used in generating the process plans and in controlling shop floor operations for both parts. The first part, named "CAMCLEVIS_FV", is a cam clevis (a holding part), the design of which was also provided to the AMRF by the Navy. (The suffix "_FV" is a naming convention that denotes that this part model is for the final workpiece after machining at the Vertical Workstation of the AMRF). The second part, named "PIPECLAMP_FV", is another holding part used by the Navy for clamping pipes in submarines.

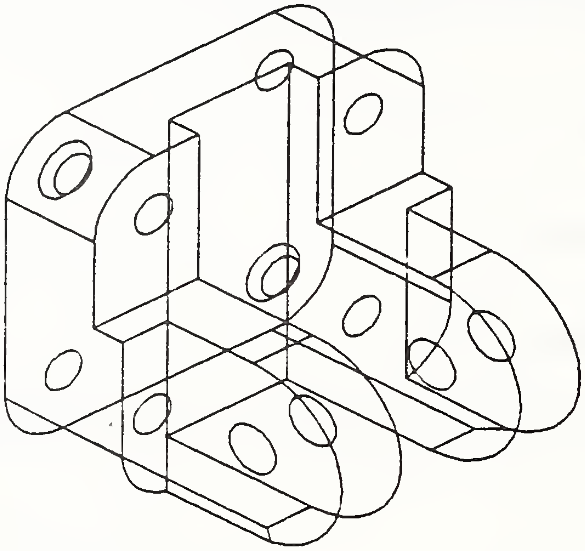

Figure F-1. Sketch of "CAMCLEVIS_FV"

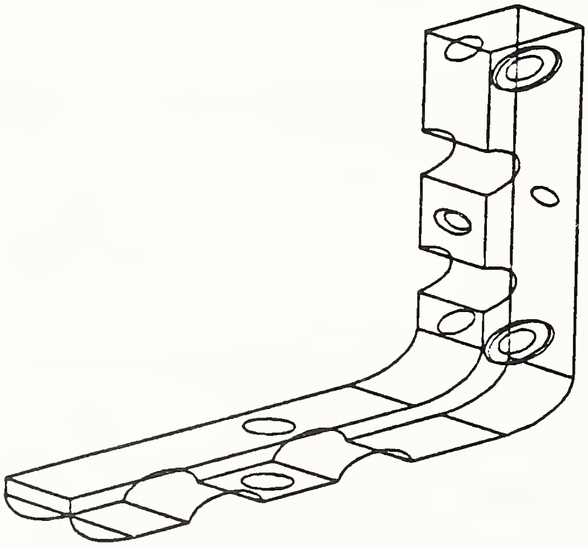

Figure F-2. Sketch of "PIPECLAMP_FV"

```
/PART_MODEL
/HEADER
        PART_NAME = 'CAMCLEVIS_FV' .
/END_HEADER
/TOPOLOGY
/SHELLS
        SHE001;
                 FAC001, FAC002, FAC003, FAC004, FAC005,
                 FAC006, FAC007, FAC008, FAC009, FAC010,
                 FAC011, FAC012, FAC013, FAC014, FAC015,
                 FAC016, FAC017, FAC018, FAC019, FAC020,
                 FAC021, FAC022, FAC023, FAC024, FAC025,
                 FAC026, FAC027, FAC028, FAC029, FAC030,
                 FAC031, FAC032 .
/END_SHELLS
/FACES
        FAC001;  LOP001, LOP002, LOP003, LOP004, LOP005; SUR001 +  .
        FAC002;  LOP006, LOP007; SUR002 +  .
        FAC003;  LOP008, LOP009; SUR003 +  .
        FAC004;  LOP010, LOP011; SUR004 +  .
        FAC005;  LOP012, LOP013; SUR005 +  .
        FAC006;  LOP014, LOP015; SUR006 +  .
        FAC007;  LOP016, LOP017; SUR007 +  .
        FAC008;  LOP018; SUR008 +  .
        FAC009;  LOP019; SUR008 +  .
        FAC010;  LOP020; SUR009 -  .
        FAC011;  LOP021; SUR010 -  .
        FAC012;  LOP022; SUR011 -  .
        FAC013;  LOP023; SUR012 -  .
        FAC014;  LOP024; SUR013 +  .
        FAC015;  LOP025; SUR014 +  .
        FAC016;  LOP026; SUR015 +  .
        FAC017;  LOP027; SUR016 +  .
        FAC018;  LOP028, LOP029; SUR017 +  .
        FAC019;  LOP030, LOP031; SUR018 +  .
        FAC020;  LOP043; SUR019 +  .
        FAC021;  LOP032; SUR019 +  .
        FAC022;  LOP033; SUR020 -  .
        FAC023;  LOP034; SUR020 -  .
        FAC024;  LOP035; SUR021 +  .
        FAC025;  LOP044; SUR021 +  .
        FAC026;  LOP036, LOP045; SUR022 +  .
        FAC027;  LOP037, LOP046; SUR022 +  .
        FAC028;  LOP038; SUR023 +  .
        FAC029;  LOP039, LOP040; SUR024 +  .
        FAC030;  LOP041, LOP042; SUR025 +  .
        FAC031;  LOP047, LOP048; SUR026 +  .
        FAC032;  LOP049, LOP050; SUR026 +  .
/END_FACES
```

```
/LOOPS
          LOP001;  EDG001 -, EDG008 -, EDG007 +, EDG006 -, EDG005 +,
                   EDG004 -, EDG003 -, EDG002 - .
          LOP002;  EDG009 + .
          LOP003;  EDG010 + .
          LOP004;  EDG011 + .
          LOP005;  EDG012 + .
          LOP006;  EDG009 - .
          LOP007;  EDG013 + .
          LOP008;  EDG012 - .
          LOP009;  EDG014 + .
          LOP010;  EDG013 - .
          LOP011;  EDG015 + .
          LOP012;  EDG010 - .
          LOP013;  EDG016 + .
          LOP014;  EDG011 - .
          LOP015;  EDG017 + .
          LOP016;  EDG014 - .
          LOP017;  EDG018 + .
          LOP018;  EDG015 - .
          LOP019;  EDG017 - .
          LOP020;  EDG008 +, EDG019 +, EDG020 +, EDG021 -, EDG022 -, EDG023 - .
          LOP021;  EDG006 +, EDG029 +, EDG030 -, EDG031 - .
          LOP022;  EDG002 +, EDG024 +, EDG025 -, EDG026 +, EDG027 -, EDG028 - .
          LOP023;  EDG004 +, EDG034 +, EDG033 -, EDG032 - .
          LOP024;  EDG005 -, EDG031 +, EDG035 +, EDG036 -, EDG037 +,
                   EDG038 +, EDG039 +, EDG034 - .
          LOP025;  EDG003 +, EDG032 +, EDG042 -, EDG041 +, EDG040 -, EDG024 - .
          LOP026;  EDG001 +, EDG028 +, EDG047 -, EDG046 -, EDG045 -,
                   EDG044 -, EDG043 -, EDG019 - .
          LOP027;  EDG007 -, EDG023 +, EDG048 +, EDG049 -, EDG050 +, EDG029 - .
          LOP028;  EDG052 +, EDG053 +, EDG051 +, EDG026 - .
          LOP029;  EDG054 - .
          LOP030;  EDG055 -, EDG021 +, EDG057 -, EDG056 - .
          LOP031;  EDG058 + .
          LOP032;  EDG063 -, EDG049 +, EDG066 +, EDG056 +, EDG065 +, EDG064 - .
          LOP033;  EDG047 +, EDG027 +, EDG051 -, EDG061 -, EDG067 + .
          LOP034;  EDG043 +, EDG068 -, EDG065 -, EDG057 +, EDG020 - .
          LOP035;  EDG025 +, EDG040 +, EDG060 -, EDG052 - .
          LOP036;  EDG054 + .
          LOP037;  EDG058 - .
          LOP038;  EDG045 +, EDG071 +, EDG037 -, EDG072 - .
          LOP039;  EDG046 +, EDG067 -, EDG062 -, EDG073 +, EDG038 -, EDG071 - .
          LOP040;  EDG069 + .
          LOP041;  EDG044 -, EDG072 +, EDG036 +, EDG075 -, EDG064 +, EDG068 + .
          LOP042;  EDG070 - .
          LOP043;  EDG059 -, EDG062 +, EDG061 +, EDG053 -, EDG060 +, EDG041 - .
          LOP044;  EDG022 +, EDG055 +, EDG066 -, EDG048 - .
          LOP045;  EDG069 - .
```

```
        LOP046;  EDG070 + .
        LOP047;  EDG059 +, EDG042 +, EDG033 +, EDG039 -, EDG073 - .
        LOP048;  EDG018 - .
        LOP049;  EDG063 +, EDG075 +, EDG035 -, EDG030 +, EDG050 - .
        LOP050;  EDG016 - .
/END_LOOPS
/EDGES
        EDG001;  VTX001, VTX002; CRV001 + .
        EDG002;  VTX002, VTX003; CRV002 + .
        EDG003;  VTX003, VTX004; CRV003 + .
        EDG004;  VTX004, VTX005; CRV004 + .
        EDG005;  VTX006, VTX005; CRV005 + .
        EDG006;  VTX006, VTX007; CRV006 + .
        EDG007;  VTX008, VTX007; CRV007 + .
        EDG008;  VTX008, VTX001; CRV008 + .
        EDG009; ; CRV009 + .
        EDG010; ; CRV010 + .
        EDG011; ; CRV011 + .
        EDG012; ; CRV012 + .
        EDG013; ; CRV013 + .
        EDG014; ; CRV014 + .
        EDG015; ; CRV015 + .
        EDG016; ; CRV016 + .
        EDG017; ; CRV017 + .
        EDG018; ; CRV018 + .
        EDG019;  VTX001, VTX009; CRV019 + .
        EDG020;  VTX009, VTX010; CRV020 + .
        EDG021;  VTX011, VTX010; CRV021 + .
        EDG022;  VTX012, VTX011; CRV022 + .
        EDG023;  VTX008, VTX012; CRV023 + .
        EDG024;  VTX003, VTX013; CRV024 + .
        EDG025;  VTX014, VTX013; CRV025 + .
        EDG026;  VTX014, VTX015; CRV026 + .
        EDG027;  VTX016, VTX015; CRV027 + .
        EDG028;  VTX002, VTX016; CRV028 + .
        EDG029;  VTX007, VTX018; CRV029 + .
        EDG030;  VTX017, VTX018; CRV030 + .
        EDG031;  VTX006, VTX017; CRV031 + .
        EDG032;  VTX004, VTX019; CRV032 + .
        EDG033;  VTX019, VTX020; CRV033 + .
        EDG034;  VTX005, VTX020; CRV034 + .
        EDG035;  VTX017, VTX021; CRV035 + .
        EDG036;  VTX022, VTX021; CRV036 + .
        EDG037;  VTX022, VTX023; CRV037 + .
        EDG038;  VTX023, VTX024; CRV038 + .
        EDG039;  VTX024, VTX020; CRV035 + .
        EDG040;  VTX013, VTX026; CRV039 + .
        EDG041;  VTX025, VTX026; CRV040 + .
        EDG042;  VTX025, VTX019; CRV041 + .
        EDG043;  VTX009, VTX027; CRV042 + .
        EDG044;  VTX028, VTX027; CRV043 + .
        EDG045;  VTX028, VTX029; CRV044 + .
        EDG046;  VTX029, VTX030; CRV045 + .
        EDG047;  VTX030, VTX016; CRV042 + .
        EDG048;  VTX012, VTX032; CRV046 + .
        EDG049;  VTX031, VTX032; CRV047 + .
        EDG050;  VTX031, VTX018; CRV048 + .
        EDG051;  VTX034, VTX015; CRV049 + .
        EDG052;  VTX014, VTX033; CRV050 + .
        EDG053;  VTX033, VTX034; CRV051 + .
        EDG054; ; CRV052 + .
        EDG055;  VTX011, VTX035; CRV053 + .
        EDG056;  VTX035, VTX036; CRV054 + .
        EDG057;  VTX036, VTX010; CRV055 + .
        EDG058; ; CRV056 + .
        EDG059;  VTX037, VTX025; CRV057 + .
        EDG060;  VTX033, VTX026; CRV058 + .
        EDG061;  VTX038, VTX034; CRV059 + .
        EDG062;  VTX037, VTX038; CRV060 + .
        EDG063;  VTX031, VTX039; CRV057 + .
        EDG064;  VTX039, VTX040; CRV061 + .
        EDG065;  VTX036, VTX040; CRV059 + .
        EDG066;  VTX032, VTX035; CRV058 + .
        EDG067;  VTX038, VTX030; CRV062 + .
        EDG068;  VTX040, VTX027; CRV063 + .
        EDG069; ; CRV064 + .
        EDG070; ; CRV065 + .
        EDG071;  VTX029, VTX023; CRV066 + .
        EDG072;  VTX028, VTX022; CRV067 + .
        EDG073;  VTX037, VTX024; CRV068 + .
        EDG075;  VTX039, VTX021; CRV070 + .
/END_EDGES
/VERTICES
        VTX001; PNT010 .
        VTX002; PNT011 .
        VTX003; PNT012 .
        VTX004; PNT013 .
        VTX005; PNT014 .
        VTX006; PNT015 .
        VTX007; PNT016 .
        VTX008; PNT017 .
        VTX009; PNT034 .
        VTX010; PNT035 .
        VTX011; PNT036 .
        VTX012; PNT037 .
        VTX013; PNT038 .
        VTX014; PNT039 .
        VTX015; PNT040 .
        VTX016; PNT041 .
        VTX017; PNT042 .
        VTX018; PNT043 .
        VTX019; PNT044 .
        VTX020; PNT045 .
        VTX021; PNT046 .
        VTX022; PNT047 .
        VTX023; PNT048 .
        VTX024; PNT049 .
        VTX025; PNT050 .
        VTX026; PNT051 .
        VTX027; PNT052 .
        VTX028; PNT053 .
        VTX029; PNT054 .
        VTX030; PNT055 .
        VTX031; PNT056 .
        VTX032; PNT057 .
        VTX033; PNT058 .
        VTX034; PNT059 .
        VTX035; PNT062 .
        VTX036; PNT063 .
        VTX037; PNT065 .
        VTX038; PNT066 .
        VTX039; PNT067 .
        VTX040; PNT068 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETRY
/SURFACES
        SUR001; PLANE; UNV003; 0.00000 .
        SUR002; CONE; PNT007; UNV006; 0.70711 .
        SUR003; CONE; PNT008; UNV006; 0.70711 .
        SUR004; CYLINDER; PNT001; UNV003; 0.15625 .
        SUR005; CYLINDER; PNT002; UNV003; 0.15625 .
        SUR006; CYLINDER; PNT003; UNV003; 0.15625 .
        SUR007; CYLINDER; PNT004; UNV003; 0.15625 .
        SUR008; PLANE; UNV006; -0.37500 .
        SUR009; CYLINDER; PNT001; UNV003; 0.25000 .
        SUR010; CYLINDER; PNT002; UNV003; 0.25000 .
        SUR011; CYLINDER; PNT003; UNV003; 0.25000 .
        SUR012; CYLINDER; PNT004; UNV003; 0.25000 .
        SUR013; PLANE; UNV002; 1.37500 .
        SUR014; PLANE; UNV001; 1.37500 .
        SUR015; PLANE; UNV005; 0.00000 .
        SUR016; PLANE; UNV004; 0.00000 .
        SUR017; PLANE; UNV001; 1.18750 .
        SUR018; PLANE; UNV004; -0.18750 .
        SUR019; PLANE; UNV002; 0.75000 .
        SUR020; CYLINDER; PNT009; UNV001; 0.37500 .
        SUR021; PLANE; UNV003; 0.50000 .
        SUR022; CYLINDER; PNT009; UNV001; 0.19660 .
        SUR023; PLANE; UNV003; 0.25000 .
        SUR024; PLANE; UNV004; -0.93750 .
        SUR025; PLANE; UNV001; 0.43750 .
        SUR026; PLANE; UNV002; 0.37500 .
/END_SURFACES
/CURVES
        CRV001; LINE; PNT010; UNV001 .
        CRV002; CIRCLE; PNT003; UNV003; PNT011.
        CRV003; LINE; PNT012; UNV002 .
        CRV004; CIRCLE; PNT004; UNV003; PNT013.
        CRV005; LINE; PNT015; UNV001 .
        CRV006; CIRCLE; PNT002; UNV003; PNT015.
        CRV007; LINE; PNT017; UNV002 .
        CRV008; CIRCLE; PNT001; UNV003; PNT010.
        CRV009; CIRCLE; PNT001; UNV003; PNT018.
        CRV010; CIRCLE; PNT002; UNV003; PNT019.
        CRV011; CIRCLE; PNT003; UNV003; PNT020.
        CRV012; CIRCLE; PNT004; UNV003; PNT021.
        CRV013; CIRCLE; PNT022; UNV003; PNT023.
        CRV014; CIRCLE; PNT024; UNV003; PNT025.
        CRV015; CIRCLE; PNT026; UNV003; PNT030.
        CRV016; CIRCLE; PNT027; UNV003; PNT031.
        CRV017; CIRCLE; PNT028; UNV003; PNT032.
        CRV018; CIRCLE; PNT029; UNV003; PNT033.
        CRV019; LINE; PNT010; UNV003 .
        CRV020; LINE; PNT034; UNV007 .
        CRV021; LINE; PNT036; UNV003 .
        CRV022; CIRCLE; PNT073; UNV003; PNT037.
        CRV023; LINE; PNT017; UNV003 .
        CRV024; LINE; PNT012; UNV003 .
        CRV025; CIRCLE; PNT074; UNV003; PNT038.
        CRV026; LINE; PNT039; UNV003 .
        CRV027; LINE; PNT041; UNV008 .
        CRV028; LINE; PNT011; UNV003 .
        CRV029; LINE; PNT016; UNV003 .
        CRV030; CIRCLE; PNT027; UNV003; PNT042.
        CRV031; LINE; PNT015; UNV003 .
        CRV032; LINE; PNT013; UNV003 .
        CRV033; CIRCLE; PNT029; UNV003; PNT044.
        CRV034; LINE; PNT014; UNV003 .
        CRV035; LINE; PNT042; UNV001 .
        CRV036; LINE; PNT047; UNV003 .
        CRV037; LINE; PNT047; UNV001 .
        CRV038; LINE; PNT048; UNV003 .
        CRV039; LINE; PNT038; UNV002 .
        CRV040; LINE; PNT050; UNV003 .
        CRV041; LINE; PNT050; UNV002 .
```

```
            CRV042; LINE; PNT034; UNV001 .
            CRV043; LINE; PNT053; UNV003 .
            CRV044; LINE; PNT053; UNV001 .
            CRV045; LINE; PNT054; UNV003 .
            CRV046; LINE; PNT037; UNV002 .
            CRV047; LINE; PNT056; UNV003 .
            CRV048; LINE; PNT056; UNV002 .
            CRV049; CIRCLE; PNT060; UNV001; PNT059.
            CRV050; LINE; PNT039; UNV002 .
            CRV051; LINE; PNT058; UNV003 .
            CRV052; CIRCLE; PNT060; UNV001; PNT061.
            CRV053; LINE; PNT036; UNV002 .
            CRV054; LINE; PNT062; UNV003 .
            CRV055; CIRCLE; PNT009; UNV001; PNT063.
            CRV056; CIRCLE; PNT009; UNV001; PNT064.
            CRV057; LINE; PNT056; UNV001 .
            CRV058; LINE; PNT057; UNV001 .
            CRV059; LINE; PNT063; UNV001 .
            CRV060; LINE; PNT065; UNV003 .
            CRV061; LINE; PNT067; UNV003 .
            CRV062; CIRCLE; PNT069; UNV001; PNT066.
            CRV063; CIRCLE; PNT070; UNV001; PNT068.
            CRV064; CIRCLE; PNT069; UNV001; PNT071.
            CRV065; CIRCLE; PNT070; UNV001; PNT072.
            CRV066; LINE; PNT054; UNV002 .
            CRV067; LINE; PNT053; UNV002 .
            CRV068; LINE; PNT065; UNV002 .
            CRV070; LINE; PNT067; UNV002 .
/END_CURVES
/POINTS
            PNT001; 0.25000, 0.25000, 0.00000 .
            PNT002; 0.25000, 1.12500, 0.00000 .
            PNT003; 1.12500, 0.25000, 0.00000 .
            PNT004; 1.12500, 1.12500, 0.00000 .
            PNT005; 0.25000, 0.25000, 0.32375 .
            PNT006; 1.12500, 1.12500, 0.32375 .
            PNT007; 0.25000, 0.25000, 0.11050 .
            PNT008; 0.25000, 0.25000, 0.11050 .
            PNT009; 0.18750, 0.37500, 0.85900 .
            PNT010; 0.25000, 0.00000, 0.00000 .
            PNT011; 1.12500, 0.00000, 0.00000 .
            PNT012; 1.37500, 0.25000, 0.00000 .
            PNT013; 1.37500, 1.12500, 0.00000 .
            PNT014; 1.12500, 1.37500, 0.00000 .
            PNT015; 0.25000, 1.37500, 0.00000 .
            PNT016; 0.00000, 1.12500, 0.00000 .
            PNT017; 0.00000, 0.25000, 0.00000 .
            PNT018; 0.25000, 0.13950, 0.00000 .
            PNT019; 0.25000, 1.04688, 0.00000 .
```

```
PNT020; 1.12500, 0.17188, 0.00000 .
PNT021; 1.12500, 1.01450, 0.00000 .
PNT022; 0.25000, 0.25000, 0.03237 .
PNT023; 0.25000, 0.17188, 0.03237 .
PNT024; 1.12500, 1.12500, 0.03237 .
PNT025; 1.12500, 1.04688, 0.03237 .
PNT026; 0.25000, 0.25000, 0.37500 .
PNT027; 0.25000, 1.12500, 0.37500 .
PNT028; 1.12500, 0.25000, 0.37500 .
PNT029; 1.12500, 1.12500, 0.37500 .
PNT030; 0.25000, 0.17188, 0.37500 .
PNT031; 0.25000, 1.04688, 0.37500 .
PNT032; 1.12500, 0.17188, 0.37500 .
PNT033; 1.12500, 1.04688, 0.37500 .
PNT034; 0.25000, 0.00000, 0.85900 .
PNT035; 0.18750, 0.00794, 0.93575 .
PNT036; 0.18750, 0.00794, 0.50000 .
PNT037; 0.00000, 0.25000, 0.50000 .
PNT038; 1.37500, 0.25000, 0.50000 .
PNT039; 1.18750, 0.00794, 0.50000 .
PNT040; 1.18750, 0.00794, 0.93575 .
PNT041; 1.12500, 0.00000, 0.85900 .
PNT042; 0.25000, 1.37500, 0.37500 .
PNT043; 0.00000, 1.12500, 0.37500 .
PNT044; 1.37500, 1.12500, 0.37500 .
PNT045; 1.12500, 1.37500, 0.37500 .
PNT046; 0.43750, 1.37500, 0.37500 .
PNT047; 0.43750, 1.37500, 0.25000 .
PNT048; 0.93750, 1.37500, 0.25000 .
PNT049; 0.93750, 1.37500, 0.37500 .
PNT050; 1.37500, 0.75000, 0.37500 .
PNT051; 1.37500, 0.75000, 0.50000 .
PNT052; 0.43750, 0.00000, 0.85900 .
PNT053; 0.43750, 0.00000, 0.25000 .
PNT054; 0.93750, 0.00000, 0.25000 .
PNT055; 0.93750, 0.00000, 0.85900 .
PNT056; 0.00000, 0.75000, 0.37500 .
PNT057; 0.00000, 0.75000, 0.50000 .
PNT058; 1.18750, 0.75000, 0.50000 .
PNT059; 1.18750, 0.75000, 0.85900 .
PNT060; 1.18750, 0.37500, 0.85900 .
PNT061; 1.18750, 0.27670, 0.85900 .
PNT062; 0.18750, 0.75000, 0.50000 .
PNT063; 0.18750, 0.75000, 0.85900 .
PNT064; 0.18750, 0.27670, 0.85900 .
PNT065; 0.93750, 0.75000, 0.37500 .
PNT066; 0.93750, 0.75000, 0.85900 .
PNT067; 0.43750, 0.75000, 0.37500 .
PNT068; 0.43750, 0.75000, 0.85900 .
```

```
          PNT069; 0.93750, 0.37500, 0.85900 .
          PNT070; 0.43750, 0.37500, 0.85900 .
          PNT071; 0.93750, 0.27670, 0.85900 .
          PNT072; 0.43750, 0.27670, 0.85900 .
          PNT073; 0.25000, 0.25000, 0.50000 .
          PNT074; 1.12500, 0.25000, 0.50000 .
/END_POINTS
/UNIT_VECTORS
          UNV001; 1.00000, 0.00000, 0.00000 .
          UNV002; 0.00000, 1.00000, 0.00000 .
          UNV003; 0.00000, 0.00000, 1.00000 .
          UNV004; -1.00000, 0.00000, 0.00000 .
          UNV005; 0.00000, -1.00000, 0.00000 .
          UNV006; 0.00000, 0.00000, -1.00000 .
          UNV007; -0.62943, 0.07996, 0.77293 .
          UNV008; 0.62943, 0.07996, 0.77293 .
/END_UNIT_VECTORS
/END_GEOMETRY
/FEATURES
          FEAT1;  SIMPLE;  FACES =  FAC005  .
          FEAT2;  SIMPLE;  FACES =  FAC004  .
          FEAT3;  SIMPLE;  FACES =  FAC006  .
          FEAT4;  SIMPLE;  FACES =  FAC007  .
          FEAT5;  PATTERN;  FEATURES =  FEAT2 ,  FEAT3  .
          FEAT6;  PATTERN;  FEATURES =  FEAT4 ,  FEAT3  .
          FEAT7;  SIMPLE;  FACES =  FAC026  .
          FEAT8;  SIMPLE;  FACES =  FAC027  .
          FEAT9;  SIMPLE;  FACES =  FAC001  .
          FEAT10;  SIMPLE;  FACES =  FAC032  .
          FEAT11;  SIMPLE;  FACES =  FAC031  .
          FEAT12;  PATTERN;  FEATURES =  FEAT10 ,  FEAT11  .
          FEAT13;  SIMPLE;  FACES =  FAC008  .
          FTR014;  PATTERN;  FEATURES =  FEAT13 ,  FEAT9  .
          FTR015;  PATTERN;  FEATURES =  FEAT8 ,  FEAT7  .
          FTR016;  HOLE; FACES = FAC001; FEATURES = FEAT2, FEAT3 .
/END_FEATURES
/FUNCTIONALITY
/TOLERANCES
          TOL001;  FTR014; PLUS_MINUS; 0.06200 0.00000 .
          TOL002;  FTR015; PARALLELISM; 0.00100 ;  DRF001 .
/END_TOLERANCES
/DATUMS
          DAT001; 'A';  FEAT9 .
/END_DATUMS
/DRFS
          DRF001;  DAT001 .
/END_DRFS
/END_FUNCTIONALITY
/END_PART_MODEL
```

```
/PART_MODEL
/HEADER
        PART_NAME = 'PIPECLAMP_FV' .
/END_HEADER
/TOPOLOGY
/SHELLS
        SHL001; FAC001, FAC002, FAC003, FAC004, FAC005,
                FAC006, FAC007, FAC008, FAC009, FAC010,
                FAC011, FAC012, FAC013, FAC014, FAC015,
                FAC016, FAC017, FAC018, FAC019, FAC020,
                FAC021, FAC022, FAC023, FAC024, FAC025,
                FAC026, FAC027, FAC028 .
/END_SHELLS
/FACES
        FAC001; LOP001; SUR001 - .
        FAC002; LOP002; SUR002 + .
        FAC003; LOP003; SUR003 + .
        FAC004; LOP004; SUR004 - .
        FAC005; LOP005; SUR005 + .
        FAC006; LOP006, LOP030; SUR006 + .
        FAC007; LOP007; SUR007 + .
        FAC008; LOP008; SUR008 + .
        FAC009; LOP009; SUR009 + .
        FAC010; LOP010; SUR010 + .
        FAC011; LOP011; SUR011 + .
        FAC012; LOP012, LOP032; SUR012 - .
        FAC013; LOP013; SUR013 - .
        FAC014; LOP014, LOP031; SUR014 + .
        FAC015; LOP015, LOP033; SUR015 + .
        FAC016; LOP016, LOP034; SUR016 + .
        FAC017; LOP017, LOP037; SUR017 + .
        FAC018; LOP018, LOP038; SUR018 + .
        FAC019; LOP019, LOP039; SUR019 + .
        FAC020; LOP020, LOP042; SUR020 + .
        FAC021; LOP021, LOP044; SUR021 + .
        FAC022; LOP022, LOP036, LOP041; SUR022 - .
        FAC023; LOP023, LOP040, LOP045; SUR023 + .
        FAC024; LOP024, LOP029; SUR001 - .
        FAC025; LOP025; SUR001 - .
        FAC026; LOP026, LOP035; SUR008 + .
        FAC027; LOP027; SUR008 + .
        FAC028; LOP028, LOP043; SUR018 + .
/END_FACES
/LOOPS
        LOP001; EDG001 +, EDG041 +, EDG025 -, EDG040 - .
        LOP002; EDG002 -, EDG042 +, EDG026 +, EDG041 - .
        LOP003; EDG003 -, EDG044 +, EDG027 +, EDG043 - .
        LOP004; EDG004 +, EDG046 +, EDG028 -, EDG045 - .
        LOP005; EDG005 +, EDG047 +, EDG029 -, EDG046 - .
        LOP006; EDG006 -, EDG048 +, EDG030 +, EDG047 - .
        LOP007; EDG007 -, EDG049 +, EDG031 +, EDG048 - .
        LOP008; EDG008 +, EDG050 +, EDG032 -, EDG049 - .
        LOP009; EDG009 -, EDG051 +, EDG033 +, EDG050 - .
        LOP010; EDG010 -, EDG053 +, EDG034 +, EDG052 - .
        LOP011; EDG011 -, EDG055 +, EDG035 +, EDG054 - .
        LOP012; EDG012 -, EDG056 +, EDG036 +, EDG055 - .
        LOP013; EDG013 +, EDG040 +, EDG037 -, EDG056 - .
        LOP014; EDG016 - .
        LOP015; EDG018 - .
        LOP016; EDG019 - .
        LOP017; EDG014 - .
        LOP018; EDG022 + .
        LOP019; EDG022 - .
        LOP020; EDG015 - .
        LOP021; EDG024 - .
        LOP022; EDG001 -, EDG013 -, EDG012 +, EDG011 +, EDG060 -,
                EDG010 +, EDG059 -, EDG009 +, EDG008 -, EDG007 +,
                EDG006 +, EDG005 -, EDG004 -, EDG058 -, EDG003 +,
                EDG057 -, EDG002 + .
        LOP023; EDG025 +, EDG026 -, EDG061 +, EDG027 -, EDG062 +,
                EDG028 +, EDG029 +, EDG030 -, EDG031 -, EDG032 +,
                EDG033 -, EDG063 +, EDG034 -, EDG064 +, EDG035 -,
                EDG036 -, EDG037 + .
        LOP024; EDG057 +, EDG043 +, EDG061 -, EDG042 - .
        LOP025; EDG058 +, EDG045 +, EDG062 -, EDG044 - .
        LOP026; EDG059 +, EDG052 +, EDG063 -, EDG051 - .
        LOP027; EDG060 +, EDG054 +, EDG064 -, EDG053 - .
        LOP028; EDG024 + .
        LOP029; EDG016 + .
        LOP030; EDG017 - .
        LOP031; EDG017 + .
        LOP032; EDG018 + .
        LOP033; EDG019 + .
        LOP034; EDG020 + .
        LOP035; EDG020 - .
        LOP036; EDG014 + .
        LOP037; EDG021 + .
        LOP038; EDG021 - .
        LOP039; EDG038 + .
        LOP040; EDG038 - .
        LOP041; EDG015 + .
        LOP042; EDG023 + .
        LOP043; EDG023 - .
        LOP044; EDG039 + .
        LOP045; EDG039 - .
/END_LOOPS
/EDGES
        EDG001; VTX001, VTX002; CRV001 + .
```

```
EDG002; VTX004, VTX002; CRV002 + .
EDG003; VTX007, VTX005; CRV003 + .
EDG004; VTX008, VTX010; CRV004 + .
EDG005; VTX010, VTX011; CRV005 + .
EDG006; VTX012, VTX011; CRV006 + .
EDG007; VTX014, VTX012; CRV007 + .
EDG008; VTX014, VTX015; CRV008 + .
EDG009; VTX017, VTX015; CRV009 + .
EDG010; VTX020, VTX018; CRV010 + .
EDG011; VTX022, VTX021; CRV011 + .
EDG012; VTX023, VTX022; CRV012 + .
EDG013; VTX023, VTX001; CRV013 + .
EDG014; ; CRV014 + .
EDG015; ; CRV015 + .
EDG016; ; CRV016 + .
EDG017; ; CRV017 + .
EDG018; ; CRV018 + .
EDG019; ; CRV019 + .
EDG020; ; CRV020 + .
EDG021; ; CRV021 + .
EDG022; ; CRV022 + .
EDG023; ; CRV023 + .
EDG024; ; CRV024 + .
EDG025; VTX039, VTX040; CRV025 + .
EDG026; VTX042, VTX040; CRV026 + .
EDG027; VTX045, VTX043; CRV027 + .
EDG028; VTX046, VTX048; CRV028 + .
EDG029; VTX048, VTX049; CRV029 + .
EDG030; VTX050, VTX049; CRV030 + .
EDG031; VTX052, VTX050; CRV031 + .
EDG032; VTX052, VTX053; CRV032 + .
EDG033; VTX055, VTX053; CRV033 + .
EDG034; VTX058, VTX056; CRV034 + .
EDG035; VTX060, VTX059; CRV035 + .
EDG036; VTX061, VTX060; CRV036 + .
EDG037; VTX061, VTX039; CRV037 + .
EDG038; ; CRV038 + .
EDG039; ; CRV039 + .
EDG040; VTX001, VTX039; CRV040 + .
EDG041; VTX002, VTX040; CRV041 + .
EDG042; VTX004, VTX042; CRV042 + .
EDG043; VTX005, VTX043; CRV043 + .
EDG044; VTX007, VTX045; CRV044 + .
EDG045; VTX008, VTX046; CRV045 + .
EDG046; VTX010, VTX048; CRV046 + .
EDG047; VTX011, VTX049; CRV047 + .
EDG048; VTX012, VTX050; CRV048 + .
EDG049; VTX014, VTX052; CRV049 + .
EDG050; VTX015, VTX053; CRV050 + .
```

```
        EDG051; VTX017, VTX055; CRV051 + .
        EDG052; VTX018, VTX056; CRV052 + .
        EDG053; VTX020, VTX058; CRV053 + .
        EDG054; VTX021, VTX059; CRV054 + .
        EDG055; VTX022, VTX060; CRV055 + .
        EDG056; VTX023, VTX061; CRV056 + .
        EDG057; VTX004, VTX005; CRV001 + .
        EDG058; VTX007, VTX008; CRV001 + .
        EDG059; VTX017, VTX018; CRV008 + .
        EDG060; VTX020, VTX021; CRV008 + .
        EDG061; VTX042, VTX043; CRV025 + .
        EDG062; VTX045, VTX046; CRV025 + .
        EDG063; VTX055, VTX056; CRV032 + .
        EDG064; VTX058, VTX059; CRV032 + .
/END_EDGES
/VERTICES
        VTX001; PT001 .
        VTX002; PT002 .
        VTX004; PT004 .
        VTX005; PT005 .
        VTX007; PT007 .
        VTX008; PT008 .
        VTX010; PT010 .
        VTX011; PT011 .
        VTX012; PT012 .
        VTX014; PT014 .
        VTX015; PT015 .
        VTX017; PT017 .
        VTX018; PT018 .
        VTX020; PT020 .
        VTX021; PT021 .
        VTX022; PT022 .
        VTX023; PT023 .
        VTX026; PT026 .
        VTX028; PT028 .
        VTX030; PT030 .
        VTX032; PT032 .
        VTX034; PT034 .
        VTX036; PT036 .
        VTX038; PT038 .
        VTX039; PT039 .
        VTX040; PT040 .
        VTX042; PT042 .
        VTX043; PT043 .
        VTX045; PT045 .
        VTX046; PT046 .
        VTX048; PT048 .
        VTX049; PT049 .
        VTX050; PT050 .
        VTX052; PT052 .
        VTX053; PT053 .
        VTX055; PT055 .
        VTX056; PT056 .
        VTX058; PT058 .
        VTX059; PT059 .
        VTX060; PT060 .
        VTX061; PT061 .
        VTX064; PT064 .
        VTX066; PT066 .
        VTX068; PT068 .
        VTX069; PT069 .
        VTX071; PT071 .
        VTX072; PT072 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETRY
/SURFACES
        SUR001; PLANE; UNV002; 0.0 .
        SUR002; CYLINDER; PT073; UNV003; 0.281 .
        SUR003; CYLINDER; PT074; UNV003; 0.281 .
        SUR004; CYLINDER; PT075; UNV003; 0.281 .
        SUR005; PLANE; UNV001; 3.812 .
        SUR006; PLANE; UNV002; 0.703 .
        SUR007; CYLINDER; PT076; UNV003; 0.500 .
        SUR008; PLANE; UNV001; 0.703 .
        SUR009; CYLINDER; PT077; UNV003; 0.281 .
        SUR010; CYLINDER; PT078; UNV003; 0.281 .
        SUR011; PLANE; UNV002; 5.000 .
        SUR012; PLANE; UNV001; 0.0 .
        SUR013; CYLINDER; PT079; UNV003; 0.500 .
        SUR014; CYLINDER; PT029; UNV002; 0.281 .
        SUR015; CYLINDER; PT033; UNV001; 0.203 .
        SUR016; CONE; PT080; UNV001; 0.707 .
        SUR017; CYLINDER; PT025; UNV003; 0.1405 .
        SUR018; PLANE; UNV003; 0.46875 .
        SUR019; CYLINDER; PT067; UNV003; 0.250 .
        SUR020; CYLINDER; PT027; UNV003; 0.1405 .
        SUR021; CYLINDER; PT070; UNV003; 0.250 .
        SUR022; PLANE; UNV003; 0.0 .
        SUR023; PLANE; UNV003; 0.500 .
/END_SURFACES
/CURVES
        CRV001; LINE; PT001; UNV001 .
        CRV002; CIRCLE; PT073; UNV003; PT002 .
        CRV003; CIRCLE; PT074; UNV003; PT005 .
        CRV004; CIRCLE; PT075; UNV003; PT010 .
        CRV005; LINE; PT010; UNV002 .
        CRV006; LINE; PT012; UNV001 .
```

```
CRV007; CIRCLE; PT076; UNV003; PT012 .
CRV008; LINE; PT014; UNV002 .
CRV009; CIRCLE; PT077; UNV003; PT015 .
CRV010; CIRCLE; PT078; UNV003; PT018 .
CRV011; LINE; PT022; UNV001 .
CRV012; LINE; PT023; UNV002 .
CRV013; CIRCLE; PT079; UNV003; PT001 .
CRV014; CIRCLE; PT025; UNV003; PT026 .
CRV015; CIRCLE; PT027; UNV003; PT028 .
CRV016; CIRCLE; PT029; UNV002; PT030 .
CRV017; CIRCLE; PT031; UNV002; PT032 .
CRV018; CIRCLE; PT033; UNV001; PT034 .
CRV019; CIRCLE; PT035; UNV001; PT036 .
CRV020; CIRCLE; PT037; UNV001; PT038 .
CRV021; CIRCLE; PT067; UNV003; PT068 .
CRV022; CIRCLE; PT067; UNV003; PT069 .
CRV023; CIRCLE; PT070; UNV003; PT071 .
CRV024; CIRCLE; PT070; UNV003; PT072 .
CRV025; LINE; PT039; UNV001 .
CRV026; CIRCLE; PT081; UNV003; PT040 .
CRV027; CIRCLE; PT082; UNV003; PT043 .
CRV028; CIRCLE; PT083; UNV003; PT048 .
CRV029; LINE; PT048; UNV002 .
CRV030; LINE; PT050; UNV001 .
CRV031; CIRCLE; PT084; UNV003; PT050 .
CRV032; LINE; PT052; UNV002 .
CRV033; CIRCLE; PT085; UNV003; PT053 .
CRV034; CIRCLE; PT086; UNV003; PT056 .
CRV035; LINE; PT060; UNV001 .
CRV036; LINE; PT061; UNV001 .
CRV037; CIRCLE; PT087; UNV003; PT039 .
CRV038; CIRCLE; PT063; UNV003; PT064 .
CRV039; CIRCLE; PT065; UNV003; PT066 .
CRV040; LINE; PT001; UNV003 .
CRV041; LINE; PT002; UNV003 .
CRV042; LINE; PT004; UNV003 .
CRV043; LINE; PT005; UNV003 .
CRV044; LINE; PT007; UNV003 .
CRV045; LINE; PT008; UNV003 .
CRV046; LINE; PT010; UNV003 .
CRV047; LINE; PT011; UNV003 .
CRV048; LINE; PT012; UNV003 .
CRV049; LINE; PT014; UNV003 .
CRV050; LINE; PT015; UNV003 .
CRV051; LINE; PT017; UNV003 .
CRV052; LINE; PT018; UNV003 .
CRV053; LINE; PT020; UNV003 .
CRV054; LINE; PT021; UNV003 .
CRV055; LINE; PT022; UNV003 .
```

```
        CRV056; LINE; PT023; UNV003 .
/END_CURVES
/POINTS
        PT001; 0.5, 0.0, 0.0 .
        PT002; 1.22072, 0.0, 0.0 .
        PT004; 1.77928, 0.0, 0.0 .
        PT005; 2.34572, 0.0, 0.0 .
        PT007; 2.90428, 0.0, 0.0 .
        PT008; 3.312, 0.0, 0.0 .
        PT010; 3.812, 0.5, 0.0 .
        PT011; 3.812, 0.703, 0.0 .
        PT012; 1.203, 0.703, 0.0 .
        PT014; 0.703, 1.203, 0.0 .
        PT015; 0.703, 1.67372, 0.0 .
        PT017; 0.703, 2.23228, 0.0 .
        PT018; 0.703, 3.17372, 0.0 .
        PT020; 0.703, 3.73228, 0.0 .
        PT021; 0.703, 5.0, 0.0 .
        PT022; 0.0, 5.0, 0.0 .
        PT023; 0.0, 0.5, 0.0 .
        PT025; 0.406, 1.141, 0.0 .
        PT026; 0.5465, 1.141, 0.0 .
        PT027; 0.406, 4.641, 0.0 .
        PT028; 0.5465, 4.641, 0.0 .
        PT029; 2.062, 0.0, 0.25 .
        PT030; 2.2025, 0.0, 0.25 .
        PT031; 2.062, 0.703, 0.25 .
        PT032; 2.2025, 0.703, 0.25 .
        PT033; 0.0, 2.703, 0.25 .
        PT034; 0.0, 2.8045, 0.25 .
        PT035; 0.664, 2.703, 0.25 .
        PT036; 0.664, 2.8045, 0.25 .
        PT037; 0.703, 2.703, 0.25 .
        PT038; 0.703, 2.8435, 0.25 .
        PT039; 0.5, 0.0, 0.5 .
        PT040; 1.22072, 0.0, 0.5 .
        PT042; 1.77928, 0.0, 0.5 .
        PT043; 2.34572, 0.0, 0.5 .
        PT045; 2.90428, 0.0, 0.5 .
        PT046; 3.312, 0.0, 0.5 .
        PT048; 3.812, 0.5, 0.5 .
        PT049; 3.812, 0.703, 0.5 .
        PT050; 1.203, 0.703, 0.5 .
        PT052; 0.703, 1.203, 0.5 .
        PT053; 0.703, 1.67372, 0.5 .
        PT055; 0.703, 2.23228, 0.5 .
        PT056; 0.703, 3.17372, 0.5 .
        PT058; 0.703, 3.73228, 0.5 .
        PT059; 0.703, 5.0, 0.5 .
```

```
          PT060; 0.0, 5.0, 0.5 .
          PT061; 0.0, 0.5, 0.5 .
          PT063; 0.406, 1.141, 0.5 .
          PT064; 0.656, 1.141, 0.5 .
          PT065; 0.406, 4.641, 0.5 .
          PT066; 0.656, 4.641, 0.5 .
          PT067; 0.406, 1.141, 0.46875 .
          PT068; 0.5465, 1.141, 0.46875 .
          PT069; 0.656, 1.141, 0.46875 .
          PT070; 0.406, 4.641, 0.46875 .
          PT071; 0.5465, 4.641, 0.46875 .
          PT072; 0.656, 4.641, 0.46875 .
          PT073; 1.500, -0.031, 0.0 .
          PT074; 2.625, -0.031, 0.0 .
          PT075; 3.312, 0.500, 0.0 .
          PT076; 1.203, 1.203, 0.0 .
          PT077; 0.734, 1.953, 0.0 .
          PT078; 0.734, 3.453, 0.0 .
          PT079; 0.500, 0.500, 0.0 .
          PT080; 0.5625, 2.703, 0.250 .
          PT081; 1.500, -0.031, 0.5 .
          PT082; 2.625, -0.031, 0.5 .
          PT083; 3.312, 0.500, 0.5 .
          PT084; 1.203, 1.203, 0.5 .
          PT085; 0.734, 1.953, 0.5 .
          PT086; 0.734, 3.453, 0.5 .
          PT087; 0.500, 0.500, 0.5 .
/END_POINTS
/UNIT_VECTORS
          UNV001; 1.0, 0, 0 .
          UNV002; 0, 1.0, 0 .
          UNV003; 0, 0, 1.0 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_PART_MODEL
```

REFERENCES

[ANSI81] American National Standards Institute, **Digital Representation for Communication of Product Definition Data**, American National Standard ANSI Y14.26M-1981, American Society of Mechanical Engineers, New York, 1981.

[ANSI82] American National Standards Institute, **Dimensioning and Tolerancing**, American National Standard ANSI Y14.5M-1982, American Society of Mechanical Engineers, New York, 1982.

[CLAR87] Clark, S. and Ressler, S., **The Geometry Modeling System User's Guide**, NBSIR 87-3508, National Bureau of Standards, Gaithersburg, Maryland, January 1987.

[RESS87] Ressler, S., **Using the AMRF Part Model Report**, NBSIR 87-3531, National Bureau of Standards, Gaithersburg, Maryland, February 1987.

[SMIT86] Smith, B., **Product Data Exchange Specification; The PDES Project: Objectives, Plans, and Schedules**, Internal Memorandum, National Bureau of Standards, Gaithersburg, Maryland, June, 1986.

[TU87] Tu, J. and Hopp, T., **Part Geometry Data in the AMRF**, NBSIR 87-3551, National Bureau of Standards, Gaithersburg, Maryland, April 1987.

READER COMMENT FORM

AMRF DATABASE REPORT FORMAT: PART MODEL

This document is one in a series of publications which document research done at the National Bureau of Standards' Automated Manufacturing Research Facility from 1981 through March, 1987.

You may use this form to comment on the technical content or organization of this document or to contribute suggested editorial changes.

Comments: ____________________

If you wish a reply, give your name, company, and complete mailing address: ____________________

What is your occupation? ____________________

NOTE: This form may not be used to order additional copies of this document or other documents in the series. Copies of AMRF documents are available from NTIS.

Please mail your comments to:

AMRF Program Manager
National Bureau of Standards
Building 220, Room B-111
Gaithersburg, MD 20899

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBSIR 87-3672	2. Performing Organ. Report No.	3. Publication Date NOVEMBER 1987

4. TITLE AND SUBTITLE

AMRF Database Report Format: Part Model

5. AUTHOR(S)

Theodore H. Hopp

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

National Bureau of Standards
Gaithersburg, Maryland 20899

10. SUPPLEMENTARY NOTES

□ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

This document specifies the format of Part Model database reports. These reports are used throughout the AMRF to communicate part model data between application processes and the global AMRF database. Part model data consists of basic shape data (topology and geometry), features, and functionality (tolerances). This document is organized in five sections. This document is intended for use by programmers implementing systems that make use of AMRF part model data. It is also intended as an introduction to the capabilities of the part model for systems analysts who must decide what applications can be supported by the AMRF part model.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

AMRF, boundary representation, features, formal languages, geometry, IGES/PDES, solid models, tolerance models, topology

13. AVAILABILITY

☒ Unlimited

□ For Official Distribution. Do Not Release to NTIS

□ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

78

15. Price

$13.95