TM-20a

PART AND ASSEMBLY DESCRIPTION LANGUAGES--II

by

A. A. G. Requicha
N. M. Samuel
H. B. Voelcker

1974
TECHNICAL MEMORANDUM 20a

PRODUCTION AUTOMATION PROJECT
College of Engineering & Applied Science
The University of Rochester
Rochester, NY 14627

TM-20a

PART AND ASSEMBLY DESCRIPTION LANGUAGES -- II

Proposed Specifications for Definititional Facilities in PADL-1.n
&
Tentative Specifications for Command Facilities

by
A. A. G. Requicha
N. M. Samuel
H. B. Voelcker

August 30, 1974
Level "a" Revision:
November 30, 1974

The work described in this Memo was supported largely by the National Science Foundation under Grant GI-34274X. Secondary support was provided by the Xerox Corporation, Gleason Works, and the University of Rochester.
PART AND ASSEMBLY DESCRIPTION LANGUAGES -- II

Contents:

1. Preliminaries
   1.1 Evolution of PADL
   1.2 Purpose of this summary specification
   1.3 Definitional and command facilities
   1.4 Definitional philosophy

2. Definitional Facilities in PADL-1.n
   2.1 Coordinate system
   2.2 Definitions as sequences of statements
   2.3 Elements of the language
   2.4 Entities
      2.4.1 Names
      2.4.2 Numbers and conventional tolerances
      2.4.3 Distances and locations
      2.4.4 Objects
      2.4.5 Primitive solids
      2.4.6 Features
      2.4.7 Datum systems
   2.5 Operators
      2.5.1 Translational operator
      2.5.2 Rotational operator
      2.5.3 Combinational operators
      2.5.4 Assembly operator
      2.5.5 Operator precedence
   2.6 Definitional statements for nominal objects
      2.6.1 Object definitional statements
      2.6.2 Value assignment statements
   2.7 Functions
      2.7.1 Feature defining statements and functions
      2.7.2 Datum defining statements and functions
   2.8 Attributes
      2.8.1 Distance tolerances
      2.8.2 Attributes of form
      2.8.3 Attributes of geometric relationship
      2.8.4 Attributes of position
      2.8.5 Attributes of size
      2.8.6 Run-out tolerances
      2.8.7 Attributes of finish
      2.8.8 Geometric details
   2.9 Structure of part definitions
   2.10 Enhancements
3. Command Facilities in PADL Systems
   3.1 Command syntax
   3.2 Classes of commands
   3.3 Tasks and priorities
   3.4 Graphic inputs
   3.5 Graphic outputs
   3.6 Graphic commands for PADL-1.0

4. Remarks and Examples
   4.1 Tolerancing facilities in PADL
   4.2 "Textbook" examples
   4.3 Other examples

Appendix: BNF notation

* * *

LEVEL "a" REVISIONS

October/November 1974

The following sections of TM-20 were modified to create TM-20a. (Minor editorial changes were made in various other sections.)

   2.4.1 - 2.4.3, 2.4.5
   2.5.1, 2.5.2
   2.6.1, 2.6.2
   2.7.1
   2.8.1 - 2.8.6
   3.1, 3.2, 3.5, 3.6
   4.1, 4.2 (Examples 1-3)

Most of the modifications are aimed at simplifying and generalizing the definition and handling of distances, locations, and tolerances. Others condense the discussion of graphic command facilities, and present an exemplary repertoire of simplified commands for PADL-1.0.
1. PRELIMINARIES

PADL is an acronym for Part and Assembly Description Language. Suffixixed numbers denote various versions of the language.

1.1 EVOLUTION OF PADL

PADL is a major extension of PDL, our initial and tentative foray into geometrically complete description systems for "simple" mechanical parts. The primary 1974-75 objective in the PADL project is a prototype description language for simple 3-D parts, and nominal assemblies thereof. This language should be extensible (at least in principle), and it should serve as a host vehicle for both increasingly advanced description facilities and experimental manufacturing planning systems.

PADL and its successors (if successors are needed) will be evolutionary in at least four senses.

1) PADL's nominal descriptive power (roughly, the variety of objects that PADL can accommodate) almost surely will be improved. This can be done by expanding the set of primitive solid building blocks in higher number versions of the language, and by expanding the set of operators. (PADL-1.0 will contain only $B$ (block) and $C$ (cylinder), and it will not contain a general rotational operator.)

2) PADL's specification facilities (roughly, facilities for dimensioning and tolerancing) are likely to be increasingly powerful in higher number versions.

3) PADL's output facilities, e.g. for generating various kinds of hard and soft graphic displays, will be expanded, as will the range and power of the "conveniences" it offers.

4) Because PADL systems are intended to be hosts for a range of automatic manufacturing and design software systems, PADL's characteristics probably will evolve somewhat unpredictably to match applications beyond object description per se.

* * *

structure to facilitate our new
1.2 PURPOSE OF THIS SUMMARY SPECIFICATION:

This summary specification proposes facilities and characteristics for the PADL family of languages which are intended to --

1) guide preparation of a User's Manual, which will enhance and complete these summary specifications;

2) guide the design and implementation of experimental versions of PADL-1.n;

3) raise issues for further R&D.

* * *

1.3 DEFINITIONAL AND COMMAND FACILITIES

Languages of the PADL family have two generic types of facilities:

1) definitional (declarative) facilities for defining "simple" mechanical parts and similar solid objects, and

2) command (imperative) facilities for directing a PADL system to perform various tasks, such as generating graphic displays of defined objects.

This memo is concerned mainly with definitional facilities in the PADL language, but Section 3 proposes tentative specifications for command facilities in PADL systems.

* * *

1.4 DEFINITIONAL PHILOSOPHY

The PADL family of languages is based on constructive solid geometry, which the User's Manual will explain to neophyte users with some care.

Languages of the PADL family offer three types of definitional facilities, as summarized in Section 4.1. The first, wherein constructive solid geometry plays the predominant role, enables nominal (roughly, "ideal") parts to be defined. The second and third types enable parts to be dimensioned and toleranced in accordance with conventional and/or modern (feature-oriented) standards.

Dimensioning and tolerancing are not well understood in a theoretical sense. Our current views on the subject, which underlie the proposals set forth below, are contained in TM-19, the first memo in this series on the PADL family. Briefly, we view nominal parts as (models of) ideal parts:
they have perfect form and exact dimensions. The introduction of tolerances provides a mechanism for defining classes of real parts which are "acceptable", i.e. which approximate ideal parts sufficiently closely to be functionally useful. This dichotomy -- ideal parts, and classes of real parts defined by variances on ideal parts -- is reflected in the PADL definitional facilities noted in the preceding paragraph.

PADL reflects a very utilitarian view of parts, and of definitional facilities for parts. This view is manifest in PADL's facilities for handling "specially finished" part features -- threaded and knurled surfaces, for example -- and also for handling such "discretionary geometric details" as bevels and fillets. As explained or implied in the sequel (c.f. 2.8.7, 2.8.8), special finishes and geometric details are tedious and expensive to describe with explicit detailed geometry, and so PADL treats them as "feature attributes" definable by procedures.

* * *

Many readers may wish to scan the examples of PADL part definitions in Section 4 before proceeding into Section 2.

* * *

2. DEFINITIONAL FACILITIES IN PADL-1.n

2.1 COORDINATE SYSTEM

A master coordinate system is provided to define directions and locations for definitional and display purposes. It is the right-handed system shown in Fig. 1 below.

* * *

2.2 DEFINITIONS AS SEQUENCES OF STATEMENTS

PADL is a vehicle for defining and displaying objects, which can be either single solids or collections of solids (assemblies). Objects are defined by SEQUENCES OF STATEMENTS that are usually organized into two blocks of statements. The first block defines a nominal object, with or without conventional "plus/minus" tolerances, while the second provides precise modern tolerancing information. The two blocks are separated by BEGIN and END, as explained in Section 2.9 below.

Two kinds of primary statements are used to define NOMINAL OBJECTS.
Three other kinds of statements:

- \( \text{odds} \): feature definitional statement
- \( \text{vas} \): datum definitional statement
- \( \text{as} \): attribute assignment statement

These can be used to qualify definitions of nominal objects, e.g., to assign (declare) tolerance criteria.

The notation used just above, i.e., \( \ldots \), is a sample of BNF, the meta language used to define PADL's syntax. A tutorial summary of BNF notation is provided in the Appendix.

* * *

2.3 ELEMENTS OF THE LANGUAGE

Individual statements in PADL are strings of language elements. There are four kinds of elements:

1) **Entities** ("things"): objects, numbers, features such as particular surfaces, etc.
2) **Operators**: for positional and angular transformation of objects, and for combining and collecting objects.
3) **Functions**: for defining datums, for "naming" features, etc.
4) **Attributes:** properties that are to be associated with features of objects, e.g. finish specifications for particular surfaces.

The four kinds of elements are discussed individually in Sections 2.4, 2.5, 2.7 and 2.8 below.

* * *

### 2.4 ENTITIES

#### 2.4.1 NAMES

Names are the primary means of denoting or referring to objects, features, and other entities in PADL. PADL names include prefix symbols so that the various kinds of entities can be distinguished easily. The basic rule is:

\[
\text{<entity name> ::= <predefined name> | <type string><name>}
\]

where

\[
\text{<predefined name> ::= SX | SY | SZ | SB | ...}
\]

\[
\text{<type string> ::= null | 8 | 9 | 99 | 999 | 9999}
\]

\[
\text{<name> ::= \{<letter><digit>\}^n 0}
\]

\[
\text{<letter> ::= A | B | C | D | E | ... | X | Y | Z}
\]

\[
\text{<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
\]

(The notation used above is BNF. As noted earlier, the Appendix contains a brief summary of BNF usage.)

The semantics of `<type string>` symbols are as follows.

- `null` ==> distance
- `8` ==> object
- `#` ==> surface
- `##` ==> edge
- `###` ==> point
- `###` ==> datum system

Note that the single `8` prefix always signifies a predefined entity, i.e. a coordinate axis or primitive solid.

**Examples:**

- `A15` (distance)
- `SB` (primitive block)
- `SY` (Y-axis of master coordinate system)
It is important to note that entity names in PADL represent ("name") geometric entities that are often expressed as relations or functions involving other geometric entities, e.g., a solid defined as a combination of other solids. In PADL, the relation and/or "value" associated with a name cannot be modified within a part definition; this means that a PADL name cannot be reused within a definition to represent different entities, i.e., different relations or values.

As later examples will show, each name in a part definition must appear once and only once as the left side of a statement in the part-defining sequence, whereas a name may appear many times on the right side of the sequence of statements. (Predefined names such as $B$ or $X$ must not appear on the left side of statements.) PADL is quite different in this respect from Fortran and similar arithmetically oriented languages.

* * *

2.4.2 NUMBERS AND CONVENTIONAL TOLERANCES

Numbers in PADL are unsigned decimal numbers with or without conventional tolerances. Integer numbers in part definitions are converted to their decimal equivalents by the PADL system.

```
<int>  ::=  <digit> | <digit><int>
<dec>  ::=  <int>.<int> | .<int> | <int>.
<no.>  ::=  <int> | <dec>
<t-no.> ::=  <int>:<tolspec>

<tolspec> ::=  B | REF | PM(<no.>,<no.>)
```

Semantics of the tolerancing notation:

- $B$ ===> BASIC
- REF ===> REFERENCE
- $PM(<no.>)$ ===> $+<no.>$
- $PM(<no.>)$ ===> $-<no.>$
- $PM(<no.1>,<no.2>)$ ===> $+<no.1>$
- $PM(<no.1>,<no.2>)$ ===> $-<no.2>$

Examples:
2
2.00
25.10: B
0.75: REF
4.25: PM(.01)
100.00: PM(.02, 0.)

***

2.4.3 DISTANCES AND LOCATIONS

A distance is an unsigned measure of

1) the separation between two physically distinct features of an object, or between an object feature and the origin of the master coordinate system,

or

2) the size of a primitive entity.

"Distance" and "dimension" are synonymous for the purposes of this memo and PADL-1.n. However, "distance" is defined precisely in TM-19 in a specific mathematical context, whereas "dimension" is a more intuitive or traditional concept for which we cannot find a precise definition in existing literature.

---

FIGURE 2
In PADL-1.n, Type 1 distances are measured in directions parallel to principal coordinate axes. They designate separations between parallel faces of objects and/or axes (centerlines) of cylindrical entities. Type 2 distances designate diameters of cylindrical entities.

Distances of either type are represented in PADL by numbers or unprefixed names, with or without conventional tolerances.

\[ \text{dist} ::= \text{no.} \mid \text{t-no.} \mid \text{name} \mid \text{name}:\text{tolspec} \]

Fig. 2 above provides examples. Distances denoted by names must be assigned numeric values via \( \text{vas}'s \), as explained later.

* * *

Locations (of features of objects) are defined in PADL by DISTANCE CHAINS (\( \text{d-chains}'s \)), where

\[ \text{d-chain} ::= \text{null} \mid + \mid - \text{dist} \mid (\text{d-chain})\{+ \mid -\}\text{dist} \]

Examples of \( \text{d-chains}'s \):

1) \(-\text{XLOC}\)
2) \(\text{XLOC} + 7.0:\text{PM}(.01)\)
3) \(\text{A} - \text{B}:\text{PM}(.01) - 10.0:\text{B} + \text{C}\)

Rules Governing \( \text{d-chains}'s \):

1) A \( \text{d-chain} \) defines a location relative to another location. In PADL-1.0, this reference location is the origin except for a special convention associated with certain primitive solids (see 2.4.5) and transformations (see 2.5.1, 2.5.2).

2) A \( \text{d-chain} \) has a strict left-to-right ordering which must be preserved.

3) The symbols "+" and "-" in \( \text{d-chains}'s \) assign directions (with respect to a principal axis) to immediately following \( \text{dist}'s \), and thus \( \text{d-chain}'s \) can be read from the left as "movement rules". In example 3 above (see the following illustration), the location of "a" relative to the origin is defined by, "move in the (+) direction by A, then move (-) by B:PM(.01), then move (-) by 10.0:B, then (+) by C."
4) A $\langle d\text{-}chain\rangle$ cannot be tolerated, but its component $\langle \text{dist}\rangle$'s can be tolerated as shown in the examples above.

5) There is but one rule for condensing $\langle d\text{-}chain\rangle$'s: a component distance and its immediate successor cancel one another if and only if they are identical and of opposite direction.

**EXAMPLES** (where "$==>$" means "condenses to" and "$=/=$" means "does not condense to"):

- $-A + A ===> 0$
- $A + B - B ===> A$
- $B + A - B /=/> A$
- $10. - 5. - 10. /=/> -5.$
- $10. - 10. - 5. ===> -5.$
- $10. : PM(0.01) - 10. : PM(0.02) /=/> 0$

6) The nominal (i.e. untoleranced) numeric value of a $\langle d\text{-}chain\rangle$ can be found by applying "+" and "-" as arithmetic operators to the nominal numeric values of the component distances, but a $\langle d\text{-}chain\rangle$'s value must not be confused with the chain itself. The operators "+,-" in $\langle d\text{-}chain\rangle$'s are monadic direction-assignment operators rather than dyadic arithmetic operators.

7) A $\langle d\text{-}chain\rangle$ cannot be assigned a numeric value (e.g. by a $\langle \text{vas}\rangle$, as in 2.6.2) unless it is a $\langle \text{dist}\rangle$. 
Page TM-20a-10

<d-chain>'s are used in PADL as parameters of primitive solids and MOV and ROT transformations, as explained below. The PADL User's Manual presents simple rules for composing <d-chain>'s which impose specific dimensioning and tolerancing patterns on parts. The theory of <d-chain>'s and dimensional trees is discussed in TM-19.

***

2.4.4 OBJECTS

PADL admits three kinds of objects.

1) Primitive solids, which are the basic "ideal building blocks" available to users. These are covered in 2.4.5.

2) Composite solids, which are solids (integral or whole, "one-piece") defined constructively via <ods>'s from other solids.

3) Assemblies, which are collections of primitives and/or composite solids which are disjoint or adjoint. These are defined via <ods>'s using the operator .ASB. of 2.5.4.

The names of composite solids and assemblies are always prefixed by the symbol 3.

***

2.4.5 PRIMITIVE SOLIDS

Two primitive solids, block and cylinder, are available in PADL-1.0. Thus

<s-prim> ::= <bloc-prim> | <cyl-prim>.

BLOCK PRIMITIVE -- Syntax

<bloc-prim> ::= &B(<d-chain>,<d-chain>,<d-chain>)
AT(<d-chain>,<d-chain>,<d-chain>)

Fig. 3 illustrates the following instance of the block primitive:

&B(L,H,D) AT (XB,YB,ZB).

BLOCK PRIMITIVE -- "Semantics"

The arguments of AT in the block primitive are locations relative to the origin, in the X, Y, Z directions, of three "reference faces" of the block. These are usually the left, lower, back faces, as in the illustration just
above.

The arguments of $\$B$ itself are locations of the remaining faces RELATIVE TO THE REFERENCE FACES. These latter arguments can be thought of as "sizes" of the block in the $X$, $Y$, $Z$ directions, with AT specifying the block's location.

Rules for constructing <d-chain> arguments for block primitives are as follows.

1) Select the reference faces (or reference corner) and construct the AT arguments as ordered chains of directed distances starting from the origin.

2) Construct the $\$B$ arguments as ordered chains of directed distances starting from the reference faces and proceeding to the opposite faces.

3) (Check) The locations relative to the origin of the non-reference faces can be found by appending the $\$B$ arguments to the AT arguments and applying the condensation rule (Rule 5) of 2.4.3.

EXAMPLE: the block in Fig. 3 can be described alternatively as

$\$B(-L,-H,-D) AT (XB+L,YB+H,ZB+D)$
if one uses the right, top, front faces as references. The locations of the left, lower, back faces are:

\[
\begin{align*}
\text{XB} + L - L & \implies \text{XB} \\
\text{YB} + H - H & \implies \text{YB} \\
\text{ZB} + D - D & \implies \text{ZB}.
\end{align*}
\]

** CYLINDER PRIMITIVE -- Syntax **

\[
\text{Y} \\
\langle \text{cyl-prim} \rangle ::=: \text{SCX}(\langle \text{dist} \rangle,\langle \text{d-chain} \rangle) \text{AT}(\langle \text{d-chain} \rangle,\langle \text{d-chain} \rangle,\langle \text{d-chain} \rangle)
\]

z

Fig. 4 illustrates the following instance of the cylinder primitive:

\[
\text{SCX}(D,L) \text{ AT } (XC,YC,ZC).
\]

** CYLINDER PRIMITIVE -- "Semantics" **

In the illustration, the "X" in \text{SCX} indicates that the axis of the cylinder is parallel to the X-axis. The arguments of AT define the location of the center of the "reference end-face" of the cylinder. The remaining
arguments define the diameter, which must be a <dist> rather than a <d-chain>, and the location of the non-reference end-face relative to the reference end-face. Thus the arguments of $C\times$ (or $C\times$ or $CZ$) can be thought of as "sizes", i.e. the diameter and length of the cylinder.

Rules for constructing <d-chain> arguments for primitive cylinders can be inferred easily from those cited above for blocks. An alternative representation of the cylinder in Fig. 4 is

$$C(X,-L)\ AT\ (XB+L, YC, ZC).$$

* *

Remarks:

1) When a primitive's reference point is at the origin, the suffix AT (0,0,0) can be omitted.

2) An alternative syntax for the primitive cylinder which might prove more congenial to users is

$$C(<dir>, <dist>, <d-chain>)\ AT\ (\ ... \)$$

or

$$CYL(<dir>, <dist>, <d-chain>)\ AT\ (\ ... \)$$

where <dir> ::= $X$ | $Y$ | $Z$.

* *

2.4.6 FEATURES

The primary features of objects are surfaces, which are designated ("named") as

``<name>``.

Secondary features are edges designated

``<name>``,

and tertiary features are points designated by

```<name>``.

Special functions (see 2.7) and <fda>'s are used to specify and name features.

* * *
2.4.7 DATUM SYSTEMS

Modern manufacturing practices require the use of datums to specify tolerances unambiguously. PADL contains facilities for defining datums which are essentially equivalent to those described in the ANSI Standard Y14.5-1973. Datum systems may consist of a single datum or a set of up to three ordered datums. Each datum may be defined in terms of named part features via <dds>'s, as explained in 2.7.2 below.

Datum systems are given names of the form

$$<\text{name}>$$

and are referred to by name in certain <ass>'s, as described in the sequel.

* * *

2.5 OPERATORS

Operators are of two kinds, monadic and dyadic. PADL's monadic operators modify the positions of objects by translation, and perhaps (see 2.5.2) by rotation; also, a version of PADL-1.n almost surely will contain a monadic operator for global scaling. The dyadic combinational and assembly operators are quite powerful; the former are unlikely to be modified or enhanced in future versions.

It should be noted that none of the operators "destroy" or otherwise modify the objects they operate upon. New objects are created to represent the effects of operators on "old" objects.

* * *

2.5.1 TRANSLATIONAL OPERATOR

The translational operator MOV is used to form translational expressions of the form

$$<\text{trans-e}>: = \text{MOV(<obj>)} \text{ BY (<d-chain>, <d-chain>, <d-chain>)}$$

where <obj> is defined in 2.6.1. The <d-chain>'s specify location of a reference feature in the new (i.e. MOV'd) object RELATIVE TO the corresponding feature of the original object.

Example: MOV(SPART) BY (XM, YM, 2.0:PM(.01)-ZM)

The MOV operator is convenient when a "sub-object", 
e.g. a counterbored hole, recurs in several different locations in a more complicated object. One need only define the "sub-object" once, in some convenient reference location, and then MOV (copy) it to other final locations before applying combinational operators.

MOV affects the conventional dimensioning and tolerancing patterns of (output) orthographic drawings in simple, sensible ways when MOV'd objects are locally defined, i.e. have <d-chain>'s that involve only a single external reference location. Formally improper dimensioning may arise when an object to be MOV'd has more than one external reference location. For further discussion, see TM-19 and the PADL User's Manual.

* * *

2.5.2 ROTATIONAL OPERATOR

The rotational operator will NOT be present in PADL-1.0. Its inclusion in later versions of PADL will be influenced strongly by industrial part survey information. This section, therefore, serves mainly as a reference for future implementations.

Rotational expressions are defined as follows.

<rot-e> ::= ROT(<obj>) BY (<a-chain>) ABT (<axis>)

where <obj> is defined in 2.6.1. <a-chain> is an angle chain analogous to a distance chain, and <axis> specifies the axis of rotation. The <axis> specification can be accomplished in a variety of ways. For example,

<axis> ::= ($X$, $Y$, $Z$),<d-chain>,<d-chain>

defines an axis parallel to a named principal axis that contains the point (<d-chain>,<d-chain>). Fig. 5 illustrates this convention for the expression

ROT(SPART) BY (90.) ABT ($Z$, A, B).

* * *

2.5.3 COMBINATIONAL OPERATORS

Combinational expressions (<comb-e>'s) are of the form

<comb-e> ::= <obj><comb-op><obj>
<comb-op> ::= .UN. | .INT. | .DIF.

where <obj> is defined in 2.6.1.
Combinational expressions are "evaluated" on a precedence basis from left to right. Union has lower precedence than intersection or difference. Parentheses can be used to enforce a desired order of evaluation. (The situation is similar to that encountered in evaluating arithmetic expressions, .UN. being analogous to arithmetic sum or difference and .INT. and .DIF. being treated as multiplication and division.) Thus

\[ SA .UN. SB .INT. SC .DIF. SD .UN. SE \]

is equivalent to

\[ (SA .UN. ((SB .INT. SC) .DIF. SD)) .UN. SE. \]

* * *

2.5.4 ASSEMBLY OPERATOR

Only one assembly operator -- .ASB., for "assemble" -- is provided in PADL-1.n, and it is present mainly for experimental purposes. It is used to create "collective" expressions, viz.

\[ \langle coll-e \rangle ::= \langle obj \rangle .ASB. \langle obj \rangle. \]
The precedence of the operator .ASB. is lower than the precedences of the combinational operators.

In essence, .ASB. merely allows the user to refer to two (or more) distinct objects by a single name. To use .ASB. for practical assembly modelling, the user must position the various objects properly via MOV statements (AT's with primitives). The semantics of .ASB. should include an intersection test, to check interference between objects in a collection on a nominal basis.

Notes on Extensions for Higher PADL's:

1) The ability to perform intersection tests under various MMC, RFS, etc. conditions would be very useful, but such a capability is too ambitious for PADL-1.0.

2) CONDITIONAL MOV's and ROT's would allow users to adjust the positions of objects being assembled automatically, subject to the null interference condition of many assembly operations.

3) A .DAS. (dis-assemble) operator might be useful, but it raises a number of issues with non-obvious "right" answers.

* * *

2.5.5 OPERATOR PRECEDENCE

PADL's operators are listed below in order of decreasing precedence.

MOV, ROT
.INT., .DIF.
.UN.
.ASB.

The sequencing of the application of MOV and ROT is determined by their parenthesized syntax. As noted earlier, parentheses can be used with the dyadic operators to force a desired order of evaluation.

* * *

2.6 DEFINITIONAL STATEMENTS FOR NOMINAL OBJECTS

As noted in 2.2, there are two primary types of statements provided in PADL-1.n for defining nominal objects.

<ode> : object definitional statements
<vas> : value assignment statements
These are discussed individually in the next two subsections.

* * *

2.6.1 OBJECT DEFINITIONAL STATEMENTS

These statements are the means whereby nominal objects are defined, with or without conventional tolerances, and named. The format of <ods>'s has the effect of assigning a (new) name to an object which can be a primitive solid, a transformed primitive, an already named object, or a combination or collection of other objects. Specifically,

<ods> ::= &<name> = <obj>

where

<obj> ::= <s-prim> | &<name> | <trans-e> | <rot-e> | <comb-e> | <coll-e>

The following examples illustrate the definitional rules.

SBLOC = S$B((1.0, 2.0, 0) A + 40.) AT (0., 0., 0.)
SNUBL = MOV(SBLOC) BY (A, B, C)
SNEXT = (MOV(SNUBL) BY (D, E, F+C).UN. SNUBL). DIF. SHOLE

* * *

2.6.2 VALUE ASSIGNMENT STATEMENTS

An object defined as above contains <dist>'s which determine, through <d-chain>'s, the "sizes" and locations of the object's component entities. <dist>'s can be numeric, with or without conventional tolerances, as shown above in examples.

Symbolic distances must be assigned numeric nominal values within the nominal block of a part definition. (The block structure of part definitions is explained in 2.9.) Value assignment statements -- <vas>'s -- provide means to do this. <vas>'s also can be used to assign tolerances to distances, although tolerance values are not required in a nominal definition.

The syntax of <vas>'s is given below.

<vas> ::= <name> = <nom-val> | <nom-val>:<tol-spec>
<nom-val> ::= <no.> | <name> | <arith-e>
<arith-e> ::= <nom-val>(+ | - | * | /)<nom-val>
The semantics of \(<\text{vas}\)'s is as follows.

1) A \(<\text{vas}\>' of the type

\[
<\text{name}> = <\text{nom-val}>
\]

assigns a nominal value to the named \(<\text{dist}\>' on
the left. The tolerance that may be associated with
the \(<\text{dist}\>' is not affected.

2) The \(<\text{vas}\>'

\[
<\text{name1}> = <\text{name2}>
\]

assigns the nominal value associated with
\(<\text{name2}>' to \(<\text{name1}>' . Such a statement is legal
even through the \(<\text{vas}\>' endowing \(<\text{name2}>' with an
explicit value may occur later in the
definitional sequence.

3) A \(<\text{vas}\>' of the type

\[
<\text{name}> = <\text{nom-val}>:<\text{tol-spec}>
\]

assigns both a nominal value and a tolerance to
a distance. One can view the ":" (colon) as
having the function of separating two
essentially distinct assignments. Indeed, this
type of \(<\text{vas}\>' is equivalent to the following two
statements: a \(<\text{vas}\>'

\[
<\text{name}> = <\text{nom-val}>
\]

and a tolerance attribute assignment (explained
in 2.6.1) of the form

\[
\text{TOL}(<\text{name}>) = <\text{tol-spec}>
\]

4) PADL does not allow re-assignment of "values" to
entities. This implies that the same \(<\text{name}>'
cannot be used twice on the left side of
\(<\text{vas}>'s. It also implies that symbolic \(<\text{dist}>'s
given explicit \(<\text{tol-spec}>'s in \(<\text{d-chain}>'s
cannot be re-toleranced in \(<\text{vas}>'s. Attempts to
do any of the above will generate warning
messages and will be ignored otherwise.

5) The dyadic operators "+", "-", "\times", "\div" in
\(<\text{arith-e}>'s have their traditional arithmetic
meanings and precedences. (Recall that "+" and
"-" are used differently, as monadic operators
to designate directions, in \(<\text{d-chain}>'s.\)
\(<\text{arith-e}>'s specify operations on nominal
values. Thus, e.g.
A = B + C

may be viewed as an abbreviation of

A = NOM(B) + NOM(C)

where NOM is a function which extracts the nominal numeric value of a <dist>. Should the present syntax prove confusing to users, we may introduce the NOM function explicitly.

Examples:

1) C = 10:PM(.01)
   Legal if C has had neither nominal nor tolerance values assigned previously.

2) SP = $B(C:PM(.01),...)
   
   C = 20.0
   Legal.

3) SP = $B(C:PM(.01),...)
   
   C = 20.0:PM(.02),...).
   Illegal; C is toleranced in the first statement.

4) C = B:PM(.01) + 10.0:PM(.02).
   Illegal; <arith-e>'s cannot be vehicles for assigning tolerances to their component entities.

5) HITE = 8.0:PM(.01)
   LENGTH = 2 * (1.0 + HITE/2.0):PM(.002).
   Legal. The tolerance of HITE is ignored when computing the nominal value of 10.0 for LENGTH. A tolerance of PM(.002) is assigned to LENGTH. Note that this tolerance is unrelated to the tolerance of HITE.

   * *
Notes on Extensions

1) An unprefixed \langle name\rangle in PADL-1.0 denotes a \langle dist\rangle and is associated with a 3-tuple of numeric values: a nominal value, a plus tolerance value, and a minus tolerance value. PADL-1.0 does not provide facilities for symbolic (named) tolerances, but these may prove desirable in the future.

One way to accommodate symbolic tolerances is to use unprefixed names to represent values (either nominal or tolerance) rather than \langle dist\rangle's, and to introduce a new data-type to represent distances. One might also introduce an explicit data-type for \langle d-chain\rangle's, thereby enabling them to be named and manipulated. Each such enhancement, however, makes the language more complicated to learn and (perhaps) to use for practical purposes.

2) Later versions of PADL may contain "wedge" and "cone" primitives and the ROT operator of 2.5.2. PADL's arithmetic facilities should be extended correspondingly to include trigonometric calculations.

* * *

2.7 FUNCTIONS

Functions in PADL-1.n are provided as a means for designating features, which are typically planar faces normal to a principal axis or more complicated surfaces which are parallel to a principal axis. They are also used to define datums in terms of features. Some versions of PADL-1.n may provide facilities for designating edges and points. Higher versions of PADL that include a general rotational operator or primitives such as "wedge" will require more extensive feature-defining capabilities than those described below.

Functions can occur only in statements within the tolerance definition block, as explained in 2.9. Therefore, it is always clear which part is being referenced.

Some PADL systems may provide interactive graphic "pointing" facilities as an optional means for designating features of (displayed) objects. The role of graphic inputs and their relationship to conventional PADL statements are discussed in Section 3.4.

* * *
2.7.1 FEATURE DEFINING STATEMENTS AND FUNCTIONS

The basic form of an <fds> (feature defining statement) is

\[ <fds> ::= (\# | \#\# | \#\#\#)\langle name\rangle = \langle feature function\rangle \]

(One \# is used for surfaces, two for edges, and three for points.) The \langle feature function\rangle's in PADL-1.n are as follows.

**FACES:**

\[ X \]
\[ YFACE(\langle cond\rangle,\langle cond\rangle,\langle cond\rangle) \]
\[ Z \]

where

\[ \langle cond\rangle ::= \text{null} | \langle d\text{-chain}\rangle \]

For example, the statement

\[ \#TOP = YFACE(\quad,10\quad,\quad) \]

in the tolerance block of a part called SP finds the face (or the aggregate of all faces) of SP which is perpendicular to the Y-axis and contains \( Y = 10 \). To avoid ambiguities one can specify as \( \langle cond\rangle \)'s the three coordinates of a point through which the face passes.

For the part defined in Fig. 6, the statement above would return a surface consisting of the two top faces. If only the shaded face were desired, one would write

\[ \#TOP1 = YFACE(0\quad,10\quad,\quad), \]

where the name \#TOP1 is arbitrary.

An alternative face definition could be:

\[ FACE(\langle normal\rangle,\langle cond\rangle,\langle cond\rangle,\langle cond\rangle) \]

where

\[ \langle normal\rangle = X | Y | Z. \]

"LATERAL" SURFACES:

These are ruled surfaces, not necessarily planar, which are PARALLEL, rather than perpendicular, to a principal axis. The generic form is

\[ X \]
\[ YSURF(\&\langle name\rangle), \]
\[ Z \]
For example:

\#BASESIDES = XSURF(\$BASE)

in the tolerance block of a part called \$PART finds ALL surfaces of \$PART which are parallel to X and which arose from a component of \$PART named \$BASE.

FACE AGGREGATES:

Aggregates of plane faces parallel to a principal plane can be designated by means of the function

XY
XZSURF(\$<name>)
YZ

This function is used in \$fds's mainly to give a single name to two parallel planes, e.g. the opposite faces of a keyway. For example,

\#SLOTSIDES = XYSURF(\$SLOT)

in the tolerance block of a part called \$SHAFT10 finds all the faces in \$SHAFT10 which are parallel to the X-Y plane and arose from a component called \$SLOT.

EDGES:

\$fds's for edges parallel to a principal axis take the form

\#<name> = YEDGE(\$cond, \$cond, \$cond),
Z
where the \(<\text{cond}>\)'s are either null or are \(<\text{d-chain}>\)'s used to designate the desired edge.

Alternative definitions could include the following.

1) \(\text{EDGE}(\langle\text{dir}\rangle, \langle\text{cond}\rangle, \langle\text{cond}\rangle, \langle\text{cond}\rangle)\)

where \(\langle\text{dir}\rangle := \#X | \#Y | \#Z.\)

2) \(\text{EDGE}(*\langle\text{name}\rangle, *\langle\text{name}\rangle)\)

where the specified edge is the intersection of the named surfaces. Note that this definition caters for curved edges as well as straight-line edges.

POINTS:

\(<\text{fds}>\)'s for points have the form

\[**\langle\text{name}\rangle = \text{POINT}(\langle\text{cond}\rangle, \langle\text{cond}\rangle, \langle\text{cond}\rangle).\]

Points can also be defined as the intersection of named edges, and by other methods as well.

** *

2.7.2 DATUM DEFINING STATEMENTS AND FUNCTIONS

Datum systems are defined via datum definitional statements \(<\text{dds}>\)'s of the generic form

\[\begin{align*}
\langle\text{dds}\rangle & ::= *\langle\text{name}\rangle = \text{DAT}(\langle\text{dat-def}\rangle[, , \langle\text{dat-def}\rangle]) \\
\langle\text{dat-def}\rangle & ::= \langle\text{feat}\rangle | (\langle\text{feat}\rangle[, , \langle\text{feat}\rangle])
\end{align*}\]

and \(<\text{feat}\>"feature") is explained below.

Datums are NOT part features; they are ideal entities (e.g., planes) defined in terms of part features. The function \(\text{DAT} \"constructs\"\) (more properly, specifies the construction of) the appropriate datums from part features indicated in the \(<\text{dat-def}>\).

A datum system may consist of a single datum or of up to three ordered datums. The datum order is implied in the \(<\text{dds}>\) as shown in the examples to follow. We shall not define \(<\text{feat}>\) formally; we merely state that a \(<\text{dat-def}>\) must include an appropriate (per ANSI Standards) set of NAMED part features, which may be qualified by conditions such as RFS, MMC, V (virtual condition).
Examples:

1) $$$1 = DAT(#A)

This defines a datum system $$$1 consisting of a single datum associated with a feature #A which could be, for example, a plane or the lateral surface of a hole.

2) $$$SYS3 = DAT((###P1,###P2,###P3),#A,#B).

The datum system $$$SYS3 has a primary datum defined by three points, a secondary datum defined on a surface #A, and a tertiary datum defined on another surface #B.

3) $$$S1 = DAT(#C(RFS),#A).

$$S1 has a primary datum defined by an RFS feature #C, and a secondary datum #A.

* *

When target areas of circular shape are used to define datums, we introduce an additional function

TARG(#<name>,###<name>,<no.>)

to denote the circular area. It is specified by the name of the (planar) feature in which it lies, its center as designated by a named point, and a numeric diameter. This function is used as an argument in the <dds>, viz.

$$D15 = DAT(#A,#B,TARG(#C,###P7,0.5)).

* *

There are no operational rules to combine named datum systems, but features can be used in the definition of several datum systems if necessary.

Finally, note that <dds>'s occur only within the tolerance definition block.

* * *

2.8 ATTRIBUTES

Attributes are properties of objects that are specified by assigning values to special function-like descriptors. Examples of attributes include

flatness
perpendicularity
position tolerance
surface finish.
Attribute assignment is done in the tolerance block of a part definition via attribute assignment statements -- `aas' s -- whose generic form is

```
<aas> ::=<at-name>(<arg-list>) = <numeric spec>.
```

The various `at-name' s needed to cover nearly all modern dimensioning and tolerancing practices are introduced below.

```
* * *
```

### 2.8.1 DISTANCE TOLERANCES

**Statements of the form**

```
TOL(<name>) = <tolspec>
```

can be used to assign numeric tolerances to symbolic distances not previously tolerated.

A default tolerance statement is provided to assign a common tolerance to all distances defined symbolically and not tolerated explicitly. It is,

```
DEFTOL(b) = <tolspec>
```

where `b' denotes a null or blank argument.

A fuller range of default tolerances, e.g. to cover angles, corner bevels, fillets, etc., will be needed in a fully engineered version of PADL.

```
* * *
```

### 2.8.2 ATTRIBUTES OF FORM

Higher versions of PADL should cater for all of the form, relational, positional, size, ... tolerancing practices prescribed in ANSI Standard Y14.5-1973. We shall indicate here and in the next several subsections the generic forms we propose to specify this information.

**Flatness:**

```
FLAT(#<name>) = <no.>
```

**Roundness:**

```
RND(#<name>) = <no.>
```

**Straightness:**

```
X
STR(Y(#<name>) = <no.>,
Z
```

where the direction of straightness measurement is indicated
by the letter X, Y, or Z.

Cylindricity, surface profiles, and "line" profiles can be specified similarly. (Profiles may require a datum specification; see 2.8.3.)

* * *

2.8.3 ATTRIBUTES OF GEOMETRIC RELATIONSHIP

Parallelism:

\[ \text{PAR}(\#\langle\text{name}\rangle, \#\#\langle\text{name}\rangle) = \langle\text{no.}\rangle \]

This specifies a parallelism tolerance between a feature and a datum.

Perpendicularity:

\[ \text{PERP}(\#\langle\text{name}\rangle, \#\#\langle\text{name}\rangle) = \langle\text{no.}\rangle(\langle\text{cond}\rangle) \]

where \(\langle\text{cond}\rangle\) signifies an appropriate RFS, MMC, ... specification. The \(\langle\text{cond}\rangle\) applies of course to the feature being tolerated, \#\langle\text{name}\rangle, rather than to the datum (system).

Angularity, concentricity, and symmetry can be specified similarly.

* * *

2.8.4 ATTRIBUTES OF POSITION

Modern positional tolerances can be introduced as follows.

\[ \text{POSTOL}(\#\langle\text{name}\rangle, \#\#\langle\text{name}\rangle) = \langle\text{no.}\rangle(\langle\text{cond}\rangle) | (\langle\text{no.}\rangle | \langle\text{null}\rangle, (\langle\text{no.}\rangle | \langle\text{null}\rangle, (\langle\text{no.}\rangle | \langle\text{null}\rangle))(\langle\text{cond}\rangle) \]

Thus one may specify either a single all-around tolerance, or up to three tolerances for the X, Y, Z directions. The \(\langle\text{cond}\rangle\) applies to \#\langle\text{name}\rangle and can include all necessary information, e.g., projected height specifications.

Because cylindrical holes are usually true-position tolerated and our POSTOL refers to features, one is forced to name explicitly each lateral surface of a hole. This may be quite inconvenient, and one might invent a convention to help the user in this regard. For example, the PADL system might automatically name the lateral surface of \#HOL as \#HOL, which would save one \#fds per hole.

* * *
2.8.5 ATTRIBUTES OF SIZE

These attributes specify the size of "features of size", e.g. holes, pockets, and slots, especially when such features are POSTOL'd. If holes are cylindrical it is natural to interpret a diameter tolerance as a size tolerance, but there is no natural way to extend this simple convention to other shapes of holes or features.

A statement of the following character seems necessary.

\[
\text{SIZTOL(#<name#)} = <\text{tolspec}> | ([<\text{tolspec}> | \text{null}]), ([<\text{tolspec}> | \text{null}]), ([<\text{tolspec}> | \text{null}])
\]

The single \text{tolspec} specification obtains over the whole feature, whereas the 3-tuple permits different tolerances to be specified in different directions.

The attribute assignment

\[
\text{SIZTOL(#POC)} = (\text{PM(.02)}, \text{PM(.01)})
\]

is illustrated in Fig. 7, which shows the top view of a pocket whose lateral sides are a feature named #POC.

* * *

( TOP VIEW )

![Figure 7](image-url)

2.8.6 RUN-OUT TOLERANCES

Circular and composite run-outs are specified as follows.

Circular run-out:

\[
\text{RUNC(#<name#>, #<name#>) = <no.>}
\]
Total or composite run-out:

\[ \text{RUNT}(<\text{name}>,<\text{name}>) = <\text{no.}> \]

* * *

### 2.8.7 Attributes of Finish

Finish attributes include specifications of surface quality and, importantly, various special operations such as threading, tapping, and knurling. Our treatment of such special operations as attributes within the tolerance block of a part definition, rather than by detailed geometry within the nominal block, is consistent with modern drafting practices. It also reflects modern manufacturing practices, in that threading, tapping, knurling, etc. almost always are final operations on particular features which are performed in standardized ways.

Suitable `<spec>`'s are illustrated below.

- THRD(<name>) = `<spec`
- TAP(<name>) = `<spec`
- FIN(<name>) = `<spec`

The forms of the `<spec>`'s should follow current industrial standards.

* * *

### 2.8.8 Geometric Details

Many basically simple parts have a surprising number of geometric "details" which are loosely specified, and which sometimes contribute more to the aesthetic character of a part than to its functionality. The most common "details" of this nature are bevels and rounds on convex corners, fillets in concave corners, and draft angles on surfaces of castings.

Such non-critical "details" are more grossly geometric or volumetric in character than are the precise tolerance specifications dealt with above, in that fillets, bevels, etc. can be readily seen with the naked eye and touched. Indeed, they can be represented with explicit solid geometry in the nominal block of a part definition, but to do so is both tedious and unnecessarily expensive.

We propose, therefore, that such geometric details be treated as attributes in the following exemplary style.

- BEV(<name>) = `<spec`
- FIL(<name>) = `<spec`
- DRANG(<name>,<name>) = `<spec>`
Note that bevels, fillets, and rounds are edge attributes, whereas a draft angle ("DRANG") is a surface attribute that requires, by convention, a datum or equivalent reference citation to remove ambiguity.

* * *

2.9 STRUCTURE OF PART DEFINITIONS

A part definition in PADL is composed of two blocks of statements, as shown below.

\[
\begin{align*}
\text{BEGIN(DEF(8<name>))} \\
\text{BEGIN(NOMDEF(8<name>))} \\
\text{Nominal Block} \\
\text{END(NOMDEF(8<name>))} \\
\text{BEGIN(TOLDEF(8<name>))} \\
\text{Tolerancing Block} \\
\text{END(TOLDEF(8<name>))} \\
\text{END(DEF(8<name>))}
\end{align*}
\]

The \(8<\text{name}>\) in the various statements above is the part name, which can be a (part) number headed by a single letter. Comments prefixed with a semicolon can be placed anywhere, as can certain commands, e.g. for display.

It may prove useful to provide multi-block definitional facilities for some classes of parts, e.g. CAST PARTS which are subsequently machined. In many companies such parts are defined by separate casting drawings and machining drawings which have distinctive systems of features and datums. Preliminary studies imply that the machined forms of such parts can be defined conveniently by appending one or more additional definitional blocks to the basic (NOMDEF, TOLDEF) blocks that define the casting.

* *

One cannot refer to entities other than \(d<\text{name}>\)’s unless they have been previously defined. Thus

\[
\begin{align*}
SB &= \#B(1.0,2.0,3.0) \\
SP &= \#A . \text{UN.} \ SB
\end{align*}
\]
$SA = CX(D,L)
D = 10.2$

is invalid because $SA$ is used in the second statement before it is defined as a cylinder in the third statement. Because this restriction does not apply to $\langle \text{dist} \rangle$'s, the third and fourth statements above form a valid $\langle \text{sub} \rangle$ sequence.

The nominal block of a definition consists of $\langle \text{ods} \rangle$'s and $\langle \text{vas} \rangle$'s. Distance tolerances may be included in $\langle \text{vas} \rangle$'s; the system merely saves such information. Users may display any named object in a definition before (and after) the END(NOMDEF(.)) statement, provided that nominal numeric values have been assigned to all distances involved in the object’s definition.

The tolerancing block may contain $\langle \text{fda} \rangle$'s, $\langle \text{dda} \rangle$'s, and $\langle \text{asa} \rangle$'s ... and of course comments and certain commands. Named features and datums must not be cited in $\langle \text{asa} \rangle$'s before they have been defined.

* *

More refined facilities for assigning numeric values to distances and attributes are likely to be wanted in later versions of PADL. Specifically, PADL's efficiency in a Group Technology environment will be enhanced if one can define "generic parts" symbolically, and then read-in (e.g. from computer files) blocks of numeric parameter values to define specific parts within a genre. Facilities for doing this should be easy to provide as part of a file-handling or macro-definition package (see 2.10), but some stylistic changes, e.g. declaration of parameters to be acquired externally, may be necessary.

* * *

2.10 ENHANCEMENTS

Some definitional enhancements to make PADL more convenient and/or more powerful are listed below, in ascending order of sophistication. Certain of these almost surely will be implemented in higher versions of PADL if industrial feasibility tests of Version 1.n and industrial part surveys indicate their desirability.

1) ENHANCED SETS OF PRIMITIVES, OPERATORS, AND DATA-TYPES

These are self-explanatory. A rotational operator or, at the minimum, a "wedge" primitive appear to be almost essential for industrial practicability. Additional explicit data types, e.g. for values and $\langle \text{d-chain} \rangle$'s, may also prove useful as noted in 2.6.2.