

Personal Designer
User Programming Language
(UPL)

Revision 6.0

User Reference Guide

Appendix G

Database Format

Database Format

Disclaimer

This appendix describes the part or drawing database format and a set of routines used to manipulate it at a subrecord level. This appendix is included for reference purposes only and is not supported as part of the released UPL product. The response center does not have the resources to answer questions on this appendix. Computervision reserves the right to change the database format and any of these routines without prior notification of our users.

CAUTION:

- Computervision cannot assume responsibility for a part or drawing which is modified by a program using any of these routines.
- The database format and the routines described in this appendix are subject to change without notice. Incorrect use of this information or the included intrinsic procedures can damage or destroy part databases.
- If this information changes in a new revision, programs which use this information or the intrinsic procedures may not work and could possibly damage your database.

Introduction To This Appendix

The first part of this appendix describes the database format. Next is a description of the different methods of database access, followed by a more detailed description of the parts of the database: the subrecords which make up each kind of entity. The last portion of the appendix details the intrinsic routines which are used to access the subrecords.

Overview of the Database Format

A Personal Designer part or drawing database file is made up of three parts:

- 1) a Header
- 2) the Master Index Block (MIB) portion and
- 3) the Part Data File (PDF) portion.

What Happens When You Activate a Part

When you activate a part in Personal Designer, the part file is opened and the database is put into Part Temporary Files. The MIB portion is put into the CVMIB.TMP file. The PDF portion is put into the CVPDRTMP file.

Database Format

When you save a part, it writes the new database information to the named part file. Otherwise, the data is lost. You can save a part using the Personal Designer commands FILE, or EXIT with the save option. You can also set the automatic save feature in the configurator.

The File Header

The file Header is 128 bytes long and contains various bookkeeping information needed by Personal Designer when a part is activated. You will never need to access this portion of the database directly because Personal Designer and UPL do this for you.

Each entity in Personal Designer is defined by a database record. These records are made up of subrecords. Each entity has one subrecord in the MIB file, and *at least* one subrecord in the PDF file.

The MIB File

The MIB file consists of 16 byte subrecords, one for each entity in the part. This subrecord holds attributes which are common to all types of entities – such as entity type, layer, view of visibility, group, font, and color. Also included is the “PDF pointer.” This is an offset into the PDF file where the rest of the information about the entity resides. Entities are identified by the MIB number which is assigned when the entity is created. This unique record number does not change randomly. It remains the same until the part is exited and/or filed in conjunction with the database packing option.

The Part Data File (PDF)

The PDF file is made up of subrecords of varying lengths, *at least* one for each entity in the part. These subrecords hold information which is specific to each type of entity. For example, the data defining a circle will be much different than the data which defines a line. All PDF subrecords for a given entity are contiguous. *For each entity type, a specific subrecord ordering must be maintained.* Some subrecords are *optional*. Some subrecords may occur *one or more* times. In addition to entity specific subrecords, there are subrecords for properties, tags, and multiple views (MViews) which may be added to *any* entity’s PDF record.

Database Format

The list below describes entity subrecord types and their contents:

Entity Subrecord Types and Their Contents

AC –	Arc information
AI –	Interval appearance information for an entity in an MView
AP –	Partial arc information – for angular dimensions
AS –	Associativity record
BI –	General purpose character data (byte)
CP –	Toolpath Cut vectors
CT –	Centerline data
di –	Dimensioning integer data
dl –	Dimensioning logical data
dr –	Dimensioning real data
D1 –	Dimensioning general information
D2 –	Auxiliary text string for dimensions
D3 –	Auxiliary text string for dimensions
D4 –	Dimension entity association
D5 –	Dimension prefix text string
D6 –	Reference entity for parallel or perpendicular orientation of a dimension
DS –	Display Image information
DV –	MView transform data
DX –	MView viewing data
EP –	Ellipse information
EX –	Extents information
gp –	General parameters
GX –	NURB curve and surface image data
I2 –	General purpose integer data (2 byte, signed)
I4 –	General purpose integer data (4 byte)
IL –	Image list (figure name)
IU –	Nsurface trim boundary 2D control points
L1 –	Dimension extension line endpoint coordinates
L2 –	Dimension extension line endpoint coordinates
LP –	Plane information
MA –	Entity appearance information in an MView
MG –	Drawing MView list
mv –	MView definitions
MV –	MView visibility assignment of an entity
NC –	NURB curve header

Database Format

NM –	Name of an entity
NS –	NURB surface header
NT –	NURB surface trim boundary header
NV –	NURB knot vector
OF –	Offset for all dimensions
PA –	Entity property information
PC –	Bezier curve and surface header
PH –	NURB control points
PU –	NURB trim boundary control points
PX –	Coordinates of a point
R4 –	General purpose single-precision real data
R8 –	General purpose double-precision real data
SP –	Evaluation point information
TB –	NURB trim boundaries
TC –	Text color for all dimensions
TD –	Text format and orientation information
TF –	Transform matrix
TG –	Entity tag information
TI –	Toolpath integer data
TR –	Toolpath real data
TS –	Toolpath spin data
TX –	Text character string and Bezier extents data
U2 –	General purpose integer data
VN –	Working View Transformation matrix
WD –	Wide string information (vertices)
XD –	Nurbs interpolation points
XF –	Exploded entity type list
XH –	Cross – hatching information
XN –	Coordinates of vertices in a string
XP –	Cross hatching information
XT –	NURB extents data
XZ –	Endpoints of a line

Accessing the Database

There are three ways of directly accessing the database: Reading, Modifying, and Writing. There are intrinsic procedures for each method.

Reading involves simply retrieving entity data from the database. The intrinsic routines simply return the information in the parameters supplied.

Database Format

Modifying involves changing already existing subrecord data. You should first read the existing data, then change that data, and finally, modify the subrecords. This will minimize the chances of damaging your part. Be sure that the new data is valid for that type of entity. Invalid data can cause the system to hang or damage your part.

Note that if UNDO is selected on, or if the new PDF subrecord data occupies more storage than the original data, the following happens: The entity's PDF subrecords are copied to the end of the PDF file to allow for expansion. The subrecords are then modified and the PDF pointer in the MIB subrecord is updated to point to them, instead of the old ones.

Writing involves adding new data to an entity. This is generally done in two situations. The first is when you are *adding* an additional PDF subrecord, such as a property subrecord, to an entity. The second is when you are *creating* a new entity in the database.

When *adding* a PDF subrecord to an existing entity, the following happens: All of the entity's PDF subrecords are copied to the end of the PDF file to allow for expansion. The new subrecords are then added and the PDF pointer in the MIB subrecord is updated to point to them, instead of the old ones.

When *creating* a new entity, the MIB subrecord is first added to the end of the MIB file. The new PDF subrecords are then added by writing them to the end of the PDF file. Finally, the PDF pointer (in the MIB subrecord) is updated to point to the new PDF subrecords.

Be sure the data to be written to or modified in the subrecords are valid. Check the database description in this appendix and be sure to follow the given ordering for the subrecords. Invalid data or subrecords which are out of order can cause the system to hang or damage your part.

Exiting or filing with the pack database option removes all the unused entity records in the database. Subrecords abandoned due to the PDF expansion mentioned above are removed. Also removed are deleted entities. When an entity has been deleted, its entity type field of its record is negated.

Database Format

Based Variables

Accessing certain subrecord types requires the use of based variables. A based variable acts for the programmer as a drafting template would for a draftsman. It superimposes a structure or form on an unstructured block of data. For example, to modify an ellipse, you must use based variables to base the contents of the EP subrecords to a REAL array and pass that array to the GetSrR, PutSrR, and/or ModSrR routines. See the examples at the end of this appendix.

Based variables are similar to normal variables except that their addresses or storage locations are based on the storage location of a previously defined variable. This is similar in concept to the FORTRAN EQUIVALENCE statement. The syntax for declaring a based variable is:

<based var> @ <var>[+<iconst>]

Replace *<based var>* with the based variable's name. Replace *<var>* with the name of the previously defined variable. Replace *<iconst>* with an integer constant giving the number of bytes from the beginning of *<var>* to place *<based var>*. Variables based to an array usually start with an offset of 2 to allow for the array's internal size descriptor. Appendix E gives the internal data storage for variables. It should be used to help determine *<iconst>*.

Based variables are declared in the variable declaration section of a procedure or function or in the Group section. Those declared in a procedure or function can be based to variables defined in the global variable Group section. Both based and non – based variables can be declared on a single variable declaration line.

Based variables cannot be arrays, however they are usually based to an array variable. When basing variables to an array, as in the first example, no parentheses or element numbers can be used in the based variable. The offset should usually start at two to allow for the length field of the array structure.

Database Format

Examples:

```
REAL A(20), A1 @ A+2, A2 @ A+6, A20 @ A+78

INTEGER BUFFER(82), BUFFER_LENGTH @ BUFFER
REAL X1 @ BUFFER+2, X2 @ BUFFER+6
COORD C1 @ BUFFER+10
```

Entity Format

The following is a description of the database format of each entity used by Personal Designer. The list describes all the supported entity types – their name, type, and subrecords used.

MIB Subrecord

Every entity has an MIB portion with exactly the same format. It is a 16-byte subrecord which contains the following fields in the following order: entity type, PDF pointer, layer number, view of visibility, group number, line font, and color number. Each of these fields is a 2-byte integer excepting the PDF pointer, which is a 4-byte integer. Note that the high order byte of the line font field contains flags related to the use of multiple views, as follows:

Bit	Meaning
7	MV subrecord exists in PDF portion
6	MA subrecord exists in PDF portion
5	AI subrecord exists in PDF portion
4 – 0	(Reserved)

where bit 7 is the most significant bit of the byte.

Use the intrinsic procedures ReadEnt, WriteEnt, and AddEnt to access the MIB subrecord. See Direct Access Intrinsic (below) for more information.

Database Format

PDF Subrecord

PDF Subrecord NOTE: The PDF subrecords are listed in the EXACT order that they must be written to the database.

Most PDF subrecord types have direct access intrinsic procedures written especially for them. Those which do not are listed at the end of the section with an appropriate based variable template. These templates may be entered and stored in a text file. The compiler directive **\$Include** may then be used to insert that text file into your program. Then use the appropriate **GetSR***, **ModSR***, or **PutSR*** routine to access the data, where * represents either **C**, **I**, **R**, or **S**. See the Direct Access Intrinsics section (below) for more information.

All coordinate data is stored in model space coordinates unless otherwise noted.

Type	Entity
1	Line XZ - line endpoints
2	String (or Arrow) XN - coordinates of string vertices WD - coordinates of vertices if a wide string (optional); If a WD subrecord is present, XN is used for digitizing. Note: If the AS subrecord is present, the string is a nodal line. TX - character string (optional); The TX subrecord is present only if the string is being used with the PIXL property for specialized repaint (for build menus and dialog boxes).
3	Arc AC - view transform, origin, radius, start angle, end angle
4	Text TD - X/Y view transform, justification, origin, height, width, line spacing TX - character string; if the TX subrecord has no characters, it is a text node, which is displayed as a small triangle and is a placeholder for text to be added later.

Database Format

Type Entity

5 Point

PX - coordinates of point
SP - evaluate nsurface data

6 Linear Dimension

TD – X/Y view transform, justification, origin, height, width, line spacino,
TX - dimension (stored as a character string)
L1 - endpoints of first witness line (optional)
L2 - endpoints of second witness line (optional)
XN - arrow coordinates (one or more)
D1 - dimension and tolerance information
D2 - overriding text string (optional)
D3 - alternate text strin- (optional)
D4 - dimension entity association
D5 - prefix text string (optional)
D6 - dimension direction (parallel or perpendicular)
TC - dimension text color (optional)
OF - dimension offsets (optional)

Note: The vertex fields in the D4 subrecord may contain integers from 1 - 1000. A - 1 value means the association is to the entity's origin. A positive value means the association is to the nth vertex of the entity. Zero means no association. Linear dimensions without a D4 subrecord are associative to the two dimension points in the D1 subrecord (if D1 is present). If one of the vertex numbers in the D4 subrecord is zero, that dimension is associated to the corresponding dimension point.

7 Label (Point Dimension)

TD - X/Y view transform, justification, origin, height, width, line spacinc,
TX - dimension (stored as a charaeter string)
XN - arrow string coordinates (one or more)
D1 - dimension and tolerance information
AP - balloon arc data (optional; one or more)
D2 - overriding text string (optional)
D3 - auxiliary text string (optional)
D5 - prefix text strin- (optional)
TC - dimension text color (optional)

Database Format

Type Entity

Label (continued)

Note: This entity type may be referred to as a balloon if the AP subrecord is present, or a PDIM (point dimension), in which the text of the label is the XYZ coordinate at the point of the arrowhead.

The linear dimension text should be center justified.

8 Radius Dimension

TD - X/Y view transform, justification, origin, height, width, line spacing

TX - dimension (stored as a character string)

XN - arrow string coordinates (one or more)

D1 - dimension and tolerance information

D2 - overriding text string (optional)

D3 - alternate text string (optional)

D5 - prefix text string (optional)

TC - dimension text color (optional)

9 Angular Dimension

TD - X/Y view transform justification, origin, height, width, line spacing

Note: Justification varies according to the standard in use; ANSI is left justified, while ISO and JIS are center justified.

TX - dimension (stored as a character string)

L1 - endpoints of first witness line (optional)

L2 - endpoints of second witness line (optional)

XN - arrow coordinates (one or more)

AP - partial arcs' oriein, beginning and ending angles (optional; one or more)

D1 - dimension and tolerance information

D2 - overriding text string (optional)

D3 - alternate test string (optional)

D5 - prefix text string (optional)

TC - dimension text color (optional)

OF - dimension offsets (optional)

Database Format

Type Entity

- 10 Cross-hatch
- XH - endpoints of cross-hatch lines (one or more)
 - XP - cross-hatch angle, distance, offset, pattern, number of boundaries
 - XN - vertices of boundaries (one or more)
- Note: These subrecords may appear in any order, and multiple
- XH - subrecords may be used for patterns, but the total PDF record size cannot exceed 32K.
 - The solid fill option is a function of the graphics display device driver only, and does not plot.
- 11 Figure Instance
- EX - extents of figure (transformed from figure definition)
 - TF - transformation matrix, MIB of figure's image list entity (entity type 147)
 - XF - exploded entity type list
 - The XF subrecord is present only on an XFIGURE or NFIGURE. Also, if the first word in a XF subrecord is "-1", all entity types are exploded.
 - AS - exploded entity association pointers
 - The AS subrecord is present only on an XFIGURE or NFIGURE.
- 12 Diameter Dimension
- TD - X/Y view transform, justification, origin, height, width, line spacing
 - TX - dimension (stored as a character string)
 - L1 - endpoints of first witness line (optional)
 - L2 - endpoints of second witness line (optional)
 - XN - arrow coordinates (one or more)
 - D1 - dimension and tolerance info
 - D2 - overriding text string (optional)
 - D3 - alternate text string (optional)
 - D5 - prefix text string (optional)
- Note: The text of a type 2 diameter dimension should be center justified.
- TC - dimension text color (optional)
 - OF - dimension offsets (optional)

Database Format

Type Entity

- 13 MView
- TD - X/Y view transform, justification, origin, height, width, line spacing
 TX - MView frame or fold line text
 XN - MView frame coordinates or fold line endpoints
 DX - MView's rotation vector, depth, originating view, MView number
 DV - MView's rotation transform, translation vector
- Note: MViews are 3D views based on the techniques of descriptive geometry. MViews are active only when in view 1, and further depend on the status flag in the PPE. Each entity that appears in an MView has a DG subrecord containing the numbers of the MView in which it should be displayed.
- 14 Ellipse
- EP - ellipse's view transform, origin, radius 1, radius2, starting angle, ending angle
- 15 Construction Line
- XZ - Two points defining infinite line
- Note: Although a construction line contains two endpoints, these define an infinite length vector which is not considered in calculating the drawing extents.
- 16 Cpole (Bezier curve)
- PC - number of poles points in polygon (NP), number of vertices in string representing curve (NV), polygon display flags
 XN - coordinates of pole points (one per pole)
 WD - vertices of string representing curve
 TX - Polygon minimum, polygon maximum, curve minimum, curve maximum (stored as 4 coordinates based to a text string)
- 17 Spole (Bezier surface)
- PC - number of poles points in U direction (NU), number of vertices in string representing curve, polygon display flags, number of pole points in V direction (NV), number of mesh lines in U direction (MU), number of mesh lines in V direction (MV)

Database Format

Type Entity

Spole (continued)

- XN - pole coordinates (NV records of NU poles)
- WD - vertices of mesh strings (one for each U strinc, followed by one for each V string)
- TX - Polygon minimum, polygon maximum, curve minimum, curve max., (stored as 4 coords. based to a text string)

18 Plane

- LP - origin point, normal vector, bounds
- WD - vertices of string representing plane (5 vertices: 4 corners of square. vert #1 = vert#5)

21 Connect node

- PX - coordinates of node.

22 Centerline

- XZ - line endpoints (see Note with CT below)
- AC - arc data (see Note with CT below)
- CT - centerline data

Note: Centerline types are: linear (1), radial (3), and composite (10). If the type specified in the CT subrecord is linear, then the first subrecord must be XZ and no AC will be present. If the specified type is radial, then the first subrecord must be AC and no XZ will be present. If the specified type is composite, then neither an XZ or AC subrecord will be present.

- XN - locations of overriding dash locations (optional)
- XH - centerline image

23 Ordinate Dimension

- TD - X/Y view transform. Justification associativity flag, MView no., text font no., slant angle.
- TX - dimension (stored as character string)
- L1 - endpoints of first witness line (optional)
- L2 - endpoints of second witness line (optional)
- XN - arrow coordinates (one or more)
- D1 - dimension and tolerance info
- D2 - overriding text string (optional)
- D3 - alternate text string (optional)

Database Format

Type Entity

Ordinate Dimension (continued)

D4 - dimension entity association.

D5 - prefix text string (optional)

TC - dimension text color (optional)

OF - dimension offsets

Note: Ordinate Dimension text should be center justified. Vertex fields in the D4 subrecord may contain integers ranging from - 1 to 1000 . A - 1 means the association is to the entity's origin. A positive value means the association is to the entity's nth vertex. A zero means no entity association. If one of the vertex numbers in the D4 subrecord is zero, then that end of the dimension is associative to the corresponding dimension point.

30 NURB curve (Nspline)

NC - NURB curve header, degree of curve, curve flags

NV - knot vector (present only if 'uniform' flag is set)

PH - control points

GX - TGF index currently must be set to 0,0.

XT - 3D extents, polygon minimum, polygon maximum, curve minimum, curve maximum

TB - curve trim boundaries, parametric start and end

XD - interpolation data points

31 NURB surface (Nsurface)

NS - Nsurface header degrees (dU, and dV) of surface, count of control points (mu and mv), surface flags, mesh of surface (ku and kv), surface trim data, interpolation points.

NV - knot vector in U direction (mu + du + 1 reals)

NV - knot vector in V direction (mv + dv+1 reals)

PH - polygon points (mv subrecords)

GX - mesh image 3D points currently must be set to 0,0.

XT - 3D extents; polygon minimum, polygon maximum, surface minimum, surface maximum

NT - 2D trim boundary-, nspline curve header - degree, count polygon points, count of critical points

Database Format

Type Entity

NURB surface (continued)

PU - trim boundary polygon points

TB - knot vector for trim curve

TC - knot vector for critical points (not currently being used)

IU - boundary interpolation points

XD - coordinates of data points (pv subrecords) if 1!=0

35 3-axis Toolpath

XN - Toolpath coordinates (one or more subrecords)

TI - Toolpath inte-er data

TR - Toolpath real data

PA - property (used for character data)

The PA subrecord appears only on the first entity in the path.

36 2 1/2-axis Toolpath

TI - Toolpath integer data

TR - Toolpath real data

PA - output file name, toolpath comment

CP - number of Z cuts, CPL origin

TS - tool spin direction and feed rate

XN - coordinates of tool path

AC - arc information (TS, XN, AC may repeat one or more times)

145 Display (Image)

DS - display extents, associated view, xmin, xmax, ymin, ymax, screen scale, view number.

Note: The MIB record is non-standard, and stores the image number in the layer field. Also, the image entity may be referred to as a display.

146 View

VN - working view transform

PX - origin of CPL

Note: The view number is stored in the MIB subrecord's layer field.

Note: If the view is defined as a CPL, a PX subrecord will be present. This gives the origin of the CPL.

Database Format

Type Entity

147 Figure Definition

IL - figure name
EX - figure extents

Note: The MIB subrecord holds figure information:

- a) The view of visibility (VVIS) field holds the MIB number of the figure's first entity
- b) The group field holds the number of entities in the figure.
- c) The line font field holds the time the figure was last filed.
- d) The color field holds the date the figure was filed.

The figure definition entity may also be referred to as a 'Figure Image List.'

Each of the entities in a figure is loaded into the current part; 1.000 is added to the type field of the MIB record of the figure's entities, to differentiate them from the part's entities. Although you may change the color, layer, etc. of your figure instance, the figure image itself will still show up with its original colors on its original layers. When a part is activated, Personal Designer checks the time and date stored here against the time and date of the source drawing-, if the drawing has been changed, the user is optionally prompted to update the figure. The source drawing file name may contain a drive and path designator if the user specified an explicit path when the figure was inserted. This may cause problems when a part is moved from one operating system to another. If no path is specified, the environment variable "FIGPATH" locates the source part.

148 Part Parameter Entity

EX - Drawing extents
dl - Dimensioning logical parameters
di - Dimensioning integer parameters
dr - Dimensioning real parameters
gp - general parameters
R4 - additional real dimension data
I2 - additional integer dimension data

Database Format

Type Entity

Part Parameter Entity (continued)

R4 - dual dimension real data
I2 - dual dimension integer data
B1 - dual dimension prefix
B1 - dual dimension suffix
R4 - centerline real data
B1 - color-by-layer data
I2 - color-by-layer table
mv - MView definitions

Note: There should never be more than one PPE entity in a part. Also, the PPE entity is created or modified each time a ZOOM ALL command is executed. Personal Designer tries to make this entity number 1, but some circumstances cause this entity to appear later in the database. With the exception of the drawing extents, the data in this entity reflect the state of the part when it was activated. Any commands issued when the part is open will not change the subrecord data, but instead change the corresponding system memory variables. The subrecord is updated only when the part is filed. To obtain the current values for the data contained in this entity, use the SYSVARI and SYSVARR UPL intrinsic functions.

149 Appended Datafile

TX - data (one or more subrecords)

Note: The data stored in the TX subrecord can be of any type. The subrecords should not exceed 1000 bytes in each length. If the data you want to store exceeds 1000 bytes, carry over the data to another TX subrecord. For this reason, it is important that the order of the subrecords be maintained.

Note: The MIB record is non-standard, and it stores the file name or data type name in the Vvis, Group, Font, Flags, and Color fields. The file name or data type name is case-sensitive and should be padded with nulls (0) if necessary.

Database Format

Type Entity

Appended Data file (continued)

Note: The following data file types are currently defined:

<u>Type number</u>	<u>Description</u>
-2.....-32.768	User-defined non-specific data
-1	User-defined ASCII text data
0	Unknown data
1	ASCII text data
2	Non-specific data
3	Shaded image
4	TIFF
5	Sun Raster
6	String macro file
7	Keyboard macro file
8	Color palette definition file
9.....32.767	Reserved

Note: You can use the DISPLAY APPENDEDFILE command to display data in an appended data file. If the absolute value of the file type number is ≤ 1 or > 10.000 , the data is displayed on the screen as ASCII text. If the absolute value of the file type number is > 1 and ≤ 10.000 , the DISPLAY APPENDEDFILE command does not display the data.

Note: API programs can be "registered" to gain control of the DISPLAY APPENDEDFILE command (prior to the application attempting any display) and may display non-ASCII data using whatever means the API program provides. For more information, refer to the API Programmer's Reference.

Database Format

Type	Entity
------	--------

150	Drawing sheet
-----	---------------

MG - drawing MV list

Note: The MIB record is non-standard, and it stores the sheet name in the Vvis, Group, Font, Flags, and Color fields. The sheet name is case-sensitive and should be padded with spaces if necessary.

Other Subrecords

The following Part Data File (PDF) subrecords are not supported with the **Rsubrec__**, **Msubrec__**, **Wsubrec__** intrinsic routines. Instead, enter the subrecord templates given below into your program and use the **GetSR__**, **ModSR__**, **PutSR__** intrinsics. You can enter these as text files and put in your program with the **\$Include** compiler directive.

Database Format

```
-----
-- AI Subrecord definition -----
-----
-- Multiple View (MView) interval appearance info.
-- This is color and font appearance info that is
-- generated by the Personal Designer command CHANGE
-- APPEARANCE INTERVAL. (Note: whole-entity MView
-- appearance info is held in the MA subrecord). AI
-- subrecords occur on each entity placed in an
-- MView AND selected in a CHANGE APPEARANCE
-- INTERVAL command. One AI subrecord occurs for
-- each interval in the MView in which the entity
-- appearance is changed. Also, each entity may
-- have its appearance changed in different MViews,
-- resulting in many AI subrecords on an entity.
-- 'MVnumberAI' is the MView number in which the
-- appearance is changed. 'ColorFontMA' holds the
-- font in two consecutive bytes. The order of these
-- bytes may be implementation dependant. On
-- machines such as Intel and DEC MIPS the font is
-- in low order byte and the color is in the high
-- order byte. On Sun SPARC machines the reverse is
-- true. A value of zero in either byte means no
-- change: use font or color in MIB subrecord.
-- ParamStartAI and ParamEndAI are the parametric
-- start and finish of the interval.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access desired variables in 'AIsubrec'
-- 2) Substitute for the 'ibuf' parameter
-----

integer AIsubrec(6)
integer MVNumberAI    @ AIsubrec + 2
integer ColorFontAI   @ AIsubrec + 4
real    ParamStartAI  @ AIsubRec + 6
real    ParamEndAI    @ AIsubRec + 10
```

Database Format

```
-----
-- AS Sub-record definition -----
-----
-- Entity associativity subrecord.
-- The associativity type and MIB number of the
-- associated entity. For each associated entity
-- there is a type and MIB number entry.
-- With parent-child associativity, one entity is
-- associated to another (one way). With peer
-- associativity, both entities point to each other.
-- Association types are:
--     Move or delete (MOVD): 1
--     Delete only    (DELO): 2
--     Move only      (MOVO): 3
--     Nodal line end 1: 101
--     Nodal line end 2: 102
-- NOTE: AS sub-record has a variable length.
-- You may have to adjust size of the 'ASsubrec'
-- array for your needs. The maximum size for
-- AS subrecord is 3000 assoc. (12000 bytes).
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access variables named below.
-- 2) Substitute 'ASsubrec' for the 'ibuf'
--     parameter.
-----
integer ASsubrec(101)
integer AssoType1AS @ ASsubrec + 2
integer AssoEnt1AS  @ ASsubrec + 4
integer AssoType2AS @ Assubrec + 6
integer AssoEnt2AS  @ Assubrec + 8
integer AssoType3AS @ Assubrec + 10
integer AssoEnt3AS  @ Assubrec + 12
integer AssoType4AS @ Assubrec + 14
integer AssoEnt4AS  @ Assubrec + 16
--etc
--etc
-----
```

Database Format

```
-----
-- 'CP' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Cut Vector information.
-----
-- Holds Number of Z Cuts,
--      Z Normal Vector / Incremental Cut Vector
--      CPL origin.
--
-- Appears as set of 4 subrecords TI, TR, PA, CP
-- on the first toolpath entity in a multiple
-- toolpath series.
--
-- The Normal Vector is the Z axis of the CPL.
-- Upon creation, the length is the depth of
-- the cut if Z Cut Number > 1.
-----
-- real CPsubrec(7)
-- real  NumZCutsCP      @ Cpsubrec + 2
-- coord ZnrmIncCutVecCP @ Cpsubrec + 6
-- coord CPLOriginCP    @ Cpsubrec + 18
-----

-----
-- 'di' Sub-record definition -----
-----
-- Dimension integer information.
-- Integer values stored on Part Parameter entity
-- (PPE). There is usually no need to access these
-- directly. Instead, use the SysVarI intrinsic to
-- access integer system variables #2001-2004, 2031.
-- See SysVarI pages in this manual for more info.
-----
-- If you must access directly:
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Assign desired integers in 'disubrec'
-- 2) Substitute for the 'ibuf' parameter
-----
integer disubrec(5)
-----
```

Database Format

```
-----
-- 'dl' Sub-record definition -----
-----
-- Dimension logical information. True/false values
-- stored on Part Parameter entity (PPE). There is
-- generally no need to access these directly.
-- Instead, use the SysVarI intrinsic to access
-- integer system variables #2005-2038. See SysVarI
-- pages in this manual for more information.
-----
-- If you must access directly:
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access desired bits in 'dlsubrec'
-- 2) Substitute for the 'ibuf' parameter
-- Use UPL intrinsics SetBit and GetBit to
-- manipulate the 1-bit logical values.
-----
integer dlsubrec(20)
-----
```

```
-----
-- 'dr' Sub-record definition -----
-----
-- Dimension real information. Real values stored
-- on Part Parameter entity (PPE). There is
-- generally no need to access these directly.
-- Instead, use the SysVarR intrinsic to access
-- real system variables #2001-2011. See SysVarR
-- pages in this manual for more information.
-----
-- If you must access directly:
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access desired reals to 'drsubrec'
-- 2) Substitute for the 'rbuf' parameter
-----
real drsubrec(11)
-----
```


Database Format

```
-----
-- D1 Sub-record definition -----
-----
-- Dimension parameter information.
-- Used on all dimension types, however format
-- of record is different for each dimension type.
-- Information below is entity specific, followed
-- by that which is common to all entity types.
-- For Linear Dimensions (LDIMS):
--   The coordinates of the ends of the entities
--   digitized when creating the dimension.
--   Dimension orientation (1=horz, 2=vert, 3=ptpt)
--   Dimension centering (1=centered, 2=not)
-- For Ancrular Dimensions (ADIMS):
--   MIB numbers and end numbers of the entities
--   digitized when creating the dimension.
-- Labels, Point Dimensions, Balloons (LDIMS):
--   MIB of the dimensioned arc or circle,
--   multiple view used when inserting dim.
-- For Diameter (DDIMS), Radius Dimensions (RDIMS):
--   MIB of the dimensioned arc or circle,
--   multiple view used when inserting dim.
--   Dimension orientation (1=horz, 2=vert, 3=ptpt)
--   diam notation type (1='DIA', 2= symbol),
--   symbol position (1=suffix, 2= prefix, 3=none)
-- For ALL dimensions:
--   Dimension angle, offset, arrow size, plus
--   tolerance, minus tolerance, scale, arrowhead
--   direction (1=in, 2=out), dimension standard
--   (1=ISO/JIS, 2=ANSI), text align=ent (1=aligned
--   2=no), arrowhead type, tolerance type,
--   tolerance precision, construction plane used to
--   create dimension, dimension precision.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
--   1) Assign desired integers in 'dlsbrec,
--   2) Substitute for the 'ibuf, parameter
-----
integer D1SubRec(42)
-----
```

Database Format

```
-- For LDIMS:
coord    End1D1      @ D1SubRec + 2
coord    End2D1      @ D1SubRec + 14
integer  LDimOrientD1 @ D1SubRec + 62
integer  CenterD1    @ D1SubRec + 80

--For ADIMS:
integer  MibLine1D1  @ D1Subrec + 2
integer  MibLine2D1  @ D1Subrec + 4
coord    EndLine1D1  @ D1Subrec + 6
coord    EndLine2D1  @ D1Subrec + 18

-- For Label:
integer  MibLabD1     @ D1Subrec + 2
integer  LabMViewD1   @ D1Subrec + 4

--For DDIMS, RDIMS:
integer  MibArcD1     @ D1Subrec + 2
integer  DimMViewD1   @ D1Subrec + 4
integer  DimOrientD1  @ D1SubRec + 62
integer  DWordSymD1   @ D1Subrec + 80
integer  IDummyD1     @ D1Subrec + 82
integer  PreSuffD1    @ D1Subrec + 84

--For All DIMS
real     AngleD1      @ D1SubRec + 26
real     OffsetD1     @ D1SubRec + 30
real     ArrowSizeD1  @ D1SubRec + 34
real     PlusTolD1    @ D1SubRec + 38
real     MinusTolD1   @ D1SubRec + 42
real     TolHgtD1     @ D1SubRec + 46
real     ScaleD1      @ D1SubRec + 50
real     RdummyD1(2)  -- reserved -do not use
integer  ArrOrientD1  @ D1SubRec + 64
integer  DimTypeD1    @ D1SubRec + 66
integer  AlignmentD1  @ D1SubRec + 68
integer  ArrowTypeD1  @ D1SubRec + 70
integer  TolTypeD1    @ D1SubRec + 72
integer  TolPrecD1    @ D1SubRec + 74
integer  CPLusedD1    @ D1SubRec + 76
integer  PrecD1       @ D1SubRec + 78
```

Database Format

```
-----
-- D2 Sub-record definition -----
-----
-- Use with GetSrS, ModSrS, and PutSrS intrinsics
--   1) Assign desired string to 'D2subrec',
--   2) Substitute for the 'sbuf' parameter
-- Note: D2 sub-record has a variable length.
-- You may have to adjust size of string for
-- your needs.
-----
string D2subrec:256 -- Overriding text string
-----

-----
-- D3 Sub-record definition -----
-----
-- Use with GetSrS, ModSrS, and PutSrS intrinsics
--   1) Assign desired string to 'D3subrec'
--   2) Substitute for the 'sbuf' parameter
-- Note: D3 sub-record has a variable length.
-- You may adjust the size of the string for
-- your needs.
-----
string D3subrec:256 -- Auxiliary text string
-----

-----
-- D4 Sub-record definition -----
-----
-- Dimension entity association.
-- The entity and vertex number (within the
-- entity) which make up the endpoints of the
-- dimension. Allows for updating of the dims
-- if either entity is moved.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
--   1) Access variables named below.
--   2) Substitute 'D4subrec' for the 'ibuf'
--      parameter.
-----
integer D4subrec(4)
integer Ent1MIB   @ D4subrec + 2
integer Ent1Vert @ D4subrec + 4
integer Ent2MIB   @ D4subrec + 6
integer Ent2Vert  9 D4subrec + 8
-----
```

Database Format

```
-----
-- D5 Sub-record definition -----
-----
-- Use with GetSrS, ModSrS, and PutSrS intrinsics
--   1) Assign desired string to 'D5subrec'
--   2) Substitute for the 'sbuf' parameter
-- Note: D5 sub-record has a variable length.
-- You may have to adjust size of string for
-- your needs.
-----
string D5subrec:256 -- Auxiliary text string
-----

-----
-- D6 Sub-record definition -----
-----
real      D6subrec(30)
coord    pnt1 @ D6subrec + 6
coord    pnt2 @ D6subrec + 18
-- pnt1/pnt2 endpoints of orientation line
-----
```

Database Format

```
-----
-- DV Sub-record definition -----
-----
-- Multiple View transform information.
-- Rotation vectors (directional cosines) and
-- translation from origin.
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Substitute DVsubrec for the 'rbuf' parameter.
-- 2) TransformDV is start address of transform
    array. Use with NullTransform, GetCPL,
    GetView, MapTo, MapFrom, etc. See App. E
    Interanl Data Storage Format under
    Transform array for more info.
-- 3) TranslationDV is the offset from origin.
-----
real    DVsubrec(12)
real    TransformDV    @ DVsubrec + 2
coord   TranslationDV @ DVsubrec + 38
-----

-----
-- DX Sub-record definition -----
-----
-- Multiple View view orientation data:
-- Rotation vector, Z-depth, View used to define
-- the MView, and defined MView number.
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Substitute DXsubrec for the 'rbuf' parameter.
-- 2) If CPL number is used to define an MView
    FromMVNumDX should be negative.
-----
real    DXsubrec(5)
coord   RotVectDX      @ DXsubrec + 2
real    ZDepthDX       @ DXsubrec + 14
integer FromMVNumDX    @ DXsubrec + 18
integer ThisMVNumDX    @ DXsubrec + 20
-----
```

Database Format

```
-----
-- EP Sub-record definition -----
-----
-- Ellipse (entity type #14) information. Rotation
-- vectors (directional cosines), origin, major
-- radius, minor radius, starting and ending angles.
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Substitute D5subrec for the 'rbuf' parameter.
-- 2) TransformEP is start address of transform
--    array. Use with NullTransform, GetCPL, GetView,
--    MapTo, MapFrom, etc. See App. E, Internal Data
--    Format under Transform array for more
--    information.
-- 3) OriginEP is the offset from model space origin.
-----
-- real Epsubrec(16)
-- real  TransformEP @ Epsubrec + 2
-- coord TranslateEP @ Epsubrec + 38
-- real  MajorRadEP  @ Epsubrec + 50
-- real  MinorRadEP  @ Epsubrec + 54
-- real  StartAngEP  @ Epsubrec + 58
-- real  EndAngEP    @ Epsubrec + 62
-----

-----
-- 'gp' Sub-record definition -----
-----
-- General parameter information. System values
-- stored on Part Parameter entity (PPE). There is
-- generally no need to access these directly.
-- Instead, use the SysVarI, SysVarR intrinsics to
-- access general system variables. See SysVarI,
-- SysVarR pages in this manual for more
-- information.
-----
integer gpsubrec(65)
-----
```

Database Format

```
-----
-- GX Sub-record definition -----
-----
-- Graphics Index information.
-----
-- Holds start of and size of curve image in
-- graphical display lists (known as TGFs in
-- CV CADDs world). It is currently not
-- implemented and should be present but ignored.
-- Both fields should be equal to zero. Used in
-- both Nspline and Nsurface.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access variables named below.
-- 2) Substitute 'GXsubrec' for 'ibuf'param.
-----
-- integer GXsubrec(4)
-- integer4 CurveImageOffGX @ GXsubrec + 2
-- integer4 BCntCurveImageGX @ GXsubrec + 6
-----
```

Database Format

```
-----
-- IU Sub-record definition -----
-----
-- Trim Bounds (Nspline) interpolation info.
-----
-- Holds coordinates u & v of interpolation
-- points of the NSurface trim bounds (which is
-- an Nspline in U & V).
--
-- This subrecord is only present if the NSurface
-- has been tri=ed ('t' flag in NS subrecord
-- is set to 1). It is #4 of 4 subrecords
-- representing the trimming Nspline.
-- (see NT subrecord for more information)
--
-- The number of interpolation points
-- will be:
--     m + d + 1
--
-- where d = degree of trimming Nspline or
-- m = # of control pnts in trimming Nspline
-- For Personal Designer Rev. 5, number of
-- interpolation points must equal the number of
-- control points in the trim boundaries.
-----
-- Use with the GetSrR, ModSrR, PutSrR intrinsics
-- 1) Access knot values in 'IUsubrec'
-- 2) Substitute 'IUsubrec' for 'rbuf' param.
-- This is a variable length subrecord. You may
-- need to change the dimension of 'IUsubrec'
-- for your needs.
-----
real IUsubrec(100)
-----
```


Database Format

```
-----
-- LP Sub-record definition -----
-----
-- Plane data:
--   Center point, Normal vector, Boundary size.
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Substitute LPSubrec for 'rbuf' parameter.
-----
-- real LPSubrec(7)
-- coord CenterPtLP 9 LPSubrec + 2
-- coord NormVectLP @ LPSubrec + 14
-- real BoundSizeLP @ LPSubrec + 26
-----

-----
-- MA Sub-record definition -----
-----
-- Multiple View (MView) appearance information.
-- This is color and font appearance info gene-
-- rated by the CHANGE APPEARANCE INTERVAL command.
-- (Note: MView interval appearance information
-- is held in the AI subrecord) MA subrecords occur
-- on each entity which is put in an MView AND
-- selected in a CHANGE APPEARANCE command. One MA
-- subrecord occurs for each MView in which the
-- appearance is changed. Thus, there may be more
-- than one MA sub-record on an entity.
-- 'MVnumberMA' is the MView number in which the
-- appearance is changed. 'ColorFontMA' holds
-- the font in two consecutive bytes. The order of
-- these bytes may be implementation dependant. On
-- machines such as Intel and DEC MIPS, the font is
-- in low order byte and color is in the high order
-- byte. On Sun SPARC machines it is the opposite.
-- A value of zero in either byte means no
-- change: use font or color in MIB sub-record.
-----
```

Database Format

```
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access desired variables in 'masubrec'
-- 2) Substitute for the 'ibuf' parameter
-----
integer MASubrec(2)
integer MVNumberMA @ MASubrec + 2
integer ColorFontMA @ MASubrec + 4
-----

-----
-- 'mv' Sub-record definition -----
-----
-- Multiple View (MView) Definition information.
-- (Note: lower-case mv)
-- This always occurs on Part Parameter Entity.
-- Holds MView number, rotation and translation
-- transformations, clipping rectangle, attribute
-- bit table and clipping status.
--
-- It is a variable length subrecord. For each
-- MView that is defined, 80 bytes of data are
-- added to the subrecord. A max of 64 MViews are
-- allowed. You may shorten 'mv-subrec1' array
-- if you use less MViews. 'mv-subrec' is currently
-- dimensioned to the maximum size of 2560=
-- ((80 bytes * 64 views) / 2 bytes per integer).
-- You may wish to change this for your needs.
--
-- MView 1 always exists.
-- You will have to add declarations (as shown
-- below for MView 2 and MView 7). Note that MView
-- information is added to the subrecord in the
-- order in which MViews are CREATED. This is not
-- necessarily in sequence of the MV numbers,
-- although it may happen that way.
-- The ordering of the bits in the bit table
-- 'MVxAttrBitTab' may be implementation dependant.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
```

Database Format

```
-- 1) Access desired variables in 'mv-subrec'
-- 2) Substitute for the libuf' parameter
-----
integer mv-subrec(2560)

-- offsets start at 2 for MV 1
integer4 MV1MVNumber      @ mv-subrec + 2
real      MV1TransformAr   @ mv-subrec + 6
coord     MV1ClipRectLL    @ mv-subrec + 54
coord     MV1ClipRectUR    @ mv-subrec + 66
integer   MV1AttrBitTab    @ mv-subrec + 78
integer   MV1ClipStatFlg   @ mv-subrec + 80

-- offsets start at 82 for next defined MV
integer4 MV2MVNumber      @ mv-subrec + 82
real      MV2TransformAr   @ mv-subrec + 86
coord     MV2ClipRectLL    @ mv-subrec + 134
coord     MV2ClipRectUR    @ mv-subrec + 146
integer   MV2AttrBitTab    @ mv-subrec + 158
integer   MV2ClipStatFlg   @ mv-subrec + 160

-- offsets start at 162 for next defined MV
integer4 MV7MVNumber      @ mv-subrec + 162
real      MV7TransformAr   @ mv-subrec + 166
coord     MV7ClipRectLL    @ mv-subrec + 214
coord     MV7ClipRectUR    @ mv-subrec + 226
integer   MV7AttrBitTab    @ mv-subrec + 238
integer   MV7ClipStatFlg   @ mv-subrec + 240

-- offsets start at ? for next defined MV
--[where ? = ((order-MV-was-defined * 80) + 2)]
--integer4 MVxMVNumber      @ mv-subrec + (? + 0)
--real      MvxTransformAr   @ mv-subrec + (? + 4)
--coord     MvxClipRectLL    @ mv-subrec + (? + 52)
--coord     MvxClipRectUR    @ mv-subrec + (? + 64)
--integer   MvxAttrBitTab    @ mv-subrec + (? + 76)
--integer   MvxClipStatFlg   @ mv-subrec + (? + 78)
-----
```

Database Format

```
-----
-- MV Sub-record definition -----
-----
-- MView visibility information. This is a
-- bit-table flagging the MViews in which the
-- entity appears. It occurs on each entity
-- which is put in a multiple view.
-- MV is a variable length subrecord. For each
-- defined MView, there is one bit in the table.
-- The table is allocated 8 bits at a time.
-- MVsubrec is dimensioned here for all 64 bits.
-- You may wish to change this for your needs.
-- The ordering of the bits in the bit table
-- 'BitTableMV' may be implementation dependant.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
--   1) Access desired variables in 'mv-subrec'
--   2) Substitute for the libuf' parameter
-- Use UPL intrinsics GetBit & SetBit to
-- manipulate bit-tables.
-----
integer MVsubrec(4)
integer BitTableMV @ MVsubrec + 2
-----
```

Database Format

```
-----
-- NC Sub-record definition -----
-----
-- NURB Spline header information.
-----
-- Contains parameters/flags about an NSPLINE.
-- Nspline degree (d) (1<d<=10)
-- Number polygon control points (1<M<=500)
-- Uniform flag (0=non-uniform,1=uniform)
-- Rational flag (0=non-rational, 1=rational)
-- Closed flag (0=open,1=closed)
-- Periodic flag (0=non-periodic,1=periodic)
-- Curve (0=general NURB, 1=line, 2=circle,
-- 3=ellipse, 4=parabola, 5=hyperbola)
-- Planar flag (0=non-planar, 1=planar)
-- Display flags (bit table)
-- Polygon display flags (bit-table:16)
-- Bit 15 is a high order bit (Intel byte order)
-- bit 15: polygon displayed
-- bit 14: direction mark (positive U)
-- bit 13: normal vector displayed
-- bit 12-0: (reserved)
-- Display quality (q). # of lines per segment
--          (2<=q<=20)
-- Interpolation points flag (I)
--      (0=points not saved,
--       1=saved in XD subrecords end of entity;
--       see XD subrecord for more info.)
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access variables named below.
-- 2) Substitute 'NCSubrec' for 'ibuf'param.
-----
integer NCsubrec(11)
```

Database Format

Integer	DegreeNC	@	Ncsubrec	+	2
Integer	NumCtrlPtsNC	@	Ncsubrec	+	4
Integer	UniformFlagNC	@	Ncsubrec	+	6
Integer	RationalFlagNC	@	Ncsubrec	+	8
Integer	ClosedFlagNC	@	Ncsubrec	+	10
Integer	PeriodicFlagNC	@	Ncsubrec	+	12
Integer	SplineTypeNC	@	Ncsubrec	+	14
Integer	PlanarFlagNC	@	Ncsubrec	+	16
Integer	DisplayFlagNC	@	Ncsubrec	+	18
Integer	DisplayQualNC	@	Ncsubrec	+	20
Integer	InterPntsSavNC	@	Ncsubrec	+	22

Database Format

```
-----
-- NS Sub-record definition -----
-----
-- NURB Surface header information.
-----
-- Contains parameters/flags about an Nsurface.
-- Nsurface degree in U (dU) (0<du<=10)
-- Nsurface degree in V (dv) (0<dv<=10)
-- Number poly. control pnts in U (mu) (1<mu<=500)
-- Number poly. control pnts in V (mv) (1<mv<=500)
-- Uniform in U flag (0=non-uniform,1=uniform)
-- Uniform in V flag (0=non-uniform,1=uniform)
-- Rational flag (0=non-rational, 1=rational)
-- Closed in U flag (0=open,1=closed)
-- Closed in V flag (0=open,1=closed)
-- Periodic in U flag(0=non-periodic,1=periodic)
-- Periodic in V flag(0=non-periodic,1=periodic)
-- Surface (0=general NURB, 1=plane,
--           2=right circle, 3= cone, 4=sphere
--           5=torus, 6=surface of revolution,
--           7=tabulated cylinder,
--           8=ruled surface
--           9=general quadratic surface)
-- Planar flag (0=non-planar, 1=planar)
-- Display flags (bit table)
-- Bit 15 is a high order bit (Intel byte order)
--   bit 15: polygon displayed
--   bit 14: direction mark (positive U)
--   bit 13: mesh displayed
--           (must be set to 1 to display surface)
--   bit 12: normal vector displayed
--   bit 11-0: (reserved)
-- Display quality (q). # of lines per segment
--           (1<q<=20)
-- Number of mesh lines in U (ku) (1<ku<20)
-- Number of mesh lines in V (kv) (1<kv<20)
-- Number of trim bounds in surface (t)
--   (see TB subrecord for more info.)
-- Outer boundary trimmed flag (B)
--   (0=outer bound same as surface edges,
```

Database Format

```
-- 1=outer bound is trimmed and saved)
-- (see TB subrecord for more info.)
-- Interpolation points flag (I)
-- (0=points not saved,
-- 1=saved in XD subrecords end of entity;
-- see XD subrecord for more info.)
```

Use with GetSrI, ModSrI, and PutSrI intrinsics

- 1) Access variables named below.
- 2) Substitute 'NSsubrec' for 'ibuflparam'.

integer NSsubrec(20)

integer DegreesInUNS	@ Nssubrec + 2
integer DegreesInVNS	@ Nssubrec + 4
integer NumCtrlPtsInUNS	@ Nssubrec + 6
integer NumCtrlPtsInVNS	@ Nssubrec + 8
integer UniformInUFlagNS	@ Nssubrec + 10
integer UniformInVFlagNS	@ Nssubrec + 12
integer RationalFlagNS	@ Nssubrec + 14
integer ClosedInUFlagNS	@ Nssubrec + 16
integer ClosedInVFlagNS	@ Nssubrec + 18
integer PeriodicInUFlagNS	@ Nssubrec + 20
integer PeriodicInVFlagNS	@ Nssubrec + 22
integer SurfaceTypeNS	@ Nssubrec + 24
integer PlanarFlagNS	@ Nssubrec + 26
integer DisplayFlagNS	@ Nssubrec + 28
integer DisplayQualNS	@ Nssubrec + 30
integer NumMeshInUNS	@ Nssubrec + 32
integer NumMeshInVNS	@ Nssubrec + 34
integer NumTrimBndsNS	@ Nssubrec + 36
integer OutBndTrimPlagNS	@ Nssubrec + 38
integer InterPntsSavNS	@ Nssubrec + 40

Database Format

```
-----
-- NT Sub-record definition -----
-----
-- Nsurface Trim Boundary information.
-----
-- This subrecord acts as the header for
-- Nsurface Trim Boundaries. It holds the curve
-- degree, count of polygon points, and count
-- of critical points in the trimming boundary.
-- The trimming boundary is a group of 2-D
-- (in U & V) Nsplines which form a closed loop.
-----
-- Nsurface trim bounds are present if the
-- surface has been trimmed by commands such as
-- CHANGE NSURFACE TRIM or CHANGE NSURFACE HOLE.
--
-- If the Nsurface has not been trimmed, the 't'
-- flag in the NS subrecord will be set to 0 and
-- the trim bound subrecords will not be present.
--
-- If the Nsurface has been trimmed, the 't' flag
-- in the NS subrecord is set to 1 and a set of
-- 4 subrecords (NT, PU, TB, IU) will be present
-- for each trim boundary effecting the surface.
--
-- If all trim boundaries lay inside the
-- outer boundary of the Nsurface, the 'B' flag
-- in the NS subrecord is set to 0. There will
-- be one set of trim bound subrecords for each
-- inner trim boundary (holes). This is the case
-- for commands like CHANGE NSURF HOLE.
--
-- Commands like CHANGE NSURF TRIM will often
-- make a trim boundary cross an outer bound
-- such that they form a closed loop. This
-- closed loop is stored as the new outer
-- boundary for the Nsurface:
-- The 'B' flag in the NS subrecord is set to 1.
-- The first set of trim boundary subrecords
-- represents the (new trimmed) outer boundaries
```

Database Format

```
-- of the Nsurface. Subsequent sets of trim
-- boundary subrecords represent any 'holes'
-- within the new bounds.
-- Note: the trim boundary must form a
-- closed loop with all outer boundaries it
-- crosses and, it may not enclose more than
-- one portion of the surface.
-- That is, the enclosed portion of the outer
-- boundary combined with the inner intersecting
-- portion of the trim boundary must completely
-- surround (only) one piece of the surface.
--
-- If the trim boundary cannot form a closed
-- loop with outer boundary, the Nsurface will
-- be split into 2 or more Nsurfaces.
--
-- For Personal Designer Rev. 5, number of
-- control points in the trim boundaries
-- must equal the number of interpolation points.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access variables named below.
-- 2) Substitute 'NTsubrec' for 'ibuf'param.
-----
integer NTsubrec(3)

integer NsplineDegNT @ NTsubrec + 2
integer NumPolyPntsNT @ NTsubrec + 4
integer NumCritPntsNT @ NTsubrec + 6
-----
```

Database Format

```
-----
-- NV Sub-record definition -----
-----
-- NURB Knot Vector information.
-----
-- Holds knot vector for NURB Spline and Surface.
-- This subrecord is only present if the Spline
-- or Surface is non-rational. The Uniform flag
-- in the NC or NV subrecord must be set to 0.
--
-- Nsplines:
-- Holds 'n' real values giving the knots for
-- the Nspline.
-- Nsurfaces:
-- There will be one NV subrecord to hold the
-- knot vector in the U direction followed by
-- another NV subrecord to hold the knot vector
-- in the V direction. Both knot vectors must be
-- present even when the surface is uniform in
-- U and V direction.
--
-- In both cases the number of knot values (n)
-- will be:  $n = m + d + 1$ 
--
-- where d = degree of curve (d) or
--           degree of surface (du) or (dv).
--           m = # of control pnts of curve (m) or
--           of the surface (mu) or (mv).
-----
-- Use with the GetSrR, ModSrR, PutSrR intrinsics
-- 1) Access knot values in 'NVsubrec'
-- 2) Substitute 'NVsubrec' for 'rbuf' param.
-- This is a variable length subrecord. You may
-- need to change the dimension of 'NVsubrec'
-- for your needs.
-----
real NVsubrec(100)
-----
```

Database Format

```
-----
-- PC Sub-record definition -----
-----
-- Bezier curve and surface parameters.
-----
-- NOTE: PC sub-record is different for
--       Curves vs Surfaces. Use appropriate one.
-----
-- For Bezier curves (PC sub-record):
-- Number of poles (2 <= np <= 8)
-- Number of vert in curve image (2<=nv<=51)
-- Polygon display flags (bit-table:16)
-- Bit 15 is a high order bit (Intel byte order)
--   bit 15 : polygon visible
--   bit 14-0: (reserved)
-- Next is XN sub-rec. holding 'np' pole coords
-- Next is WD sub-rec. holding 'nv' image coords
-- Last is TX sub-rec. described below.
-----
-- For Bezier surfaces (PC sub-record):
-- Number of poles in U direction (NU: 2<=NU<=8)
-- Number vertices in curve image(nv: 2<=nv<=51)
-- Polygon display flags (bit-table:16)
--   bit 15: polygon displayed
--   bit 14: direction mark displayed (positive U)
--   bit 13: mesh displayed
--   bit 12: normal vector displayed
--   bit 11-0: (reserved)
-- Number of poles in V direction (NV: 2<=NV<=8)
-- Mesh in U direction (MU)
-- Mesh in V direction (MV)
-- Next are NV XN sub-rec.s holding NU pole coords
-- Next are MU+MV WD sub-rec.s holding NV coords
-- Last is TX sub-rec. described below.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
--   1) Access variables named below.
--   2) Substitute 'PCSubrecC' (curve) or
--      or 'PCsubrecS' (surface) the 'ibuf'param.
```

Database Format

```
-- A special version of TX subrecord is
-- used in curves and surfaces for polygon
-- and curve/surface minimum and maximums.
-- Access appropriate variables listed
-- below and substitute 'CurveExtTX' or
-- 'SurfExtTX' for the 'textstring'
-- parameter in calls to RsubrecTX,
-- MsubrecTX, and WsubrecTX.
```

```
-----
integer PCsubrecC(3)
```

```
integer CNumPolesPC @ PCsubrecC + 2
integer CNumVertsPC @ PCsubrecC + 4
integer CDispFlagPC @ PCsubrecC + 6
```

```
string CurveExtTX:48
```

```
coord CPolyMinTX @ CurveExtTX + 4
coord CPolyMaxTX @ CurveExtTX + 16
coord CMinPointTX @ CurveExtTX + 28
coord CMaxPointTX @ CurveExtTX + 40
```

```
-- Surface version -----
```

```
integer PCsubrecS(6)
```

```
integer SNumPolesUPC @ PCsubrecS + 2
integer SnumVertsPC @ PCsubrecS + 4
integer SdispFlagPC @ PCsubrecS + 6
integer SNumPolesVPC @ PCsubrecS + 8
integer SNumMeshUPC @ PCsubrecS + 10
integer SNumMeshVPC @ PCsubrecS + 12
```

```
string SurfExtTX:48
```

```
coord SpolyMinTX @ SurfExtTX + 4
coord SpolyMaxTX @ SurfExtTX + 16
coord SminPointTX @ SurfExtTX + 28
coord SmaxPointTX @ SurfExtTX + 40
-----
```

Database Format

```
-----
-- PH Sub-record definition -----
-----
-- Polygon control points.
-- The homogenous coordinates of the control
-- points of the Nspline or Nsurface.
-----
-- Each control point is represented by 4 real
-- values for the x, y, z, and h coordinates,
-- respectively.
-- Nsplines:  one PH subrecord of 'm' points.
-- Nsurfaces: 'mv' PH subrecords of 'mu' points.
-----
-- Use with the GetSrR, ModSrR, PutSrR intrinsics
-- 1) Access homogeneous coordinates using
--    'PHsubrec' array.
-- 2) Substitute 'PHsubrec' for 'rbuf' param.
-- This is a variable length record. There
-- is no theoretical limit on control points,
-- however, the system may limit you to 500. We
-- use 100 points here. You may need to adjust the
-- size of 'PHsubrec' for your needs.
-----
real PHsubrec(100)
-----
```

Database Format

```
-----
-- PU Sub-record definition -----
-----
-- Trim Bounds (Nspline) Polygon information.
-----
-- Holds controlling polygon points for NSurface
-- trim bounds (which is an Nspline in U & V).
--
-- For each controlling polygon point, this
-- subrecord holds the homogeneous coordinates:
-- (u, v, h). The number of polygon points is
-- contained in the NT subrecord.
-- (see NT subrecord for more information)
--
-- This subrecord is only present if the Nsurface
-- has been trimmed ('t' flag in NS subrecord
-- is set to 1). It is #2 of 4 subrecords
-- representing the tri=ing Nspline.
-- (see NS subrecord for more information)
--
-----
-- Use with the GetSrR, ModSrR, PutSrR intrinsics
-- 1) Access knot values in 'PUsubrec'
-- 2) Substitute 'PUsubrec' for 'rbuf' param.
-- This is a variable length subrecord. You may
-- need to change the dimension of 'PUsubrec'
-- for your needs.
-----
real PUsubrec(100)
-----
```

Database Format

```
-----  
-- TB Sub-record definition -----
```

```
-----  
-- Trim Bounds (Nspline) Knot Vector information.  
-----
```

Holds knot vector for NSurface trim bounds (which is an Nspline in U & V). A TB subrecord may also be used to hold the knot vector for critical points in the trimming Nspline.

This subrecord is only present if the NSurface has been tri=ed ('t' flag in NS subrecord is set to 1). It is #3 of 4 subrecords representing the trimming Nspline. (see NT subrecord for more information)

The number of knot values (n)
will be:

$$n = m + d + 1$$

where d = degree of trimming Nspline or
 m = # of control pnts in trimming
 Nspline

If used to hold the critical points it will hold 'lc' real values for the critical point knot vector.

```
-----  
Use with the GetSrR, ModSrR, PutSrR intrinsics  
1) Access knot values in 'TBsubrec'  
2) Substitute 'TBsubrec' for 'rbuf' param.  
This is a variable length subrecord. You may  
need to change the dimension of 'TBsubrec'  
for your needs.  
-----
```

```
real TBsubrec(100)  
-----
```


Database Format

```
-----
-- TG Sub-record definition -----
-----
-- Tag Fields.
-- Non-graphical data. Multiple fields.
-- one 4 byte tag number followed by variable
-- length tag fields. Each field consists of
-- a 2-byte integer byte-count followed by data.
-----
-- NOTE: It is far easier and highly recommended
--       that you use the following UPL intrinsics
--       INSTEAD of directly accessing tag fields:
--       TagMib, MibTag, SetTagField, GetTagField.
-- If you must, use with GetSrI, ModSrI, and
-- PutSrI intrinsics
--   1) Access variables named below.
--   2) Substitute 'TGsubrec' for the 'ibuf'
--       parameter.
-----
integer TGsubrec(1000)

integer4 TagNumTG      @ Tgsubrec + 2
integer TagField1TG    @ Tgsubrec + 6
integer TagField1Data @ Tgsubrec + 8
--integer TagField2TG   @ Tgsubrec + ??
--integer TagField2Data @ Tgsubrec + ??
--etc
-----
```

Database Format

```
-----
-- 'TI' Sub-record definition - (3 Axis) -----
-----
-- 3-Axis Toolpath integer information.
-----

integer VLimitInXTI3      @ TI3axis + 2
integer VLimitInYTI3      @ TI3axis + 4
integer VLimitInZTI3      @ TI3axis + 6
integer ToolTypeTI3       @ TI3axis + 8
--      (0=ball, 1=flat, 2=flat corner)
integer NumCutPIsTI3      @ TI3axis + 10
integer CutPlaneTI3       @ TI3axis + 12
--      (0=X, 1=Y, 2=Z, 3=other)
integer VarStepFlgTI3     @ TI3axis + 14
--      (0=off, 1=on)

integer CrossCutFlgTI3    @ TI3axis + 16
integer SpecialModeTI3    @ TI3axis + 18
--      (1=recut, 2=IntAuto, 3=IntSeries, 4=Proj)
integer ToolTypeRecTI3    @ TI3axis + 20
integer CutDirectTI3      @ TI3axis + 22
--      (0=zigzag, 1=unidirl, 2=unidirl2)
integer RoughmodeTI3      @ TI3axis + 24
--      (0=constant, 1=Zrough, 2=Variable)
integer CompFlgTI3        @ TI3axis + 26
--      (0=off, 1=on)
integer AutoZFlgTI3       @ TI3axis + 28
--      (0=off, 1=on)
integer StopOnSurfTI3     @ TI3axis + 30
integer StopOnRestrTI3    @ TI3axis + 32
integer MagPosTI3         @ TI3axis + 34
integer OptRecInAutoTI3   @ TI3axis + 36
integer AutoZTypeTI3      @ TI3axis + 38
--      (0=RetAuto, 1=RetAbs, 2=RetInc)
integer BorderSmoothTI3   @ TI3axis + 40
--      (always 1)
integer SeparatesTPTI3    @ TI3axis + 42
--      (for mapping)
integer WarningLevelTI3   @ TI3axis + 44
--      (for twisted surfaces)
integer AStockPropFlgTI3  @ TI3axis + 46
```

Database Format

```
Integer NumberAxesTI3    @ TI3axis + 62
--      (3, 4, or 5)
integer DirectIn4or5TI3 @ TI3axis + 64
--      (0 or 1)
```

Use with GetSrI, ModSrI, and PutSrI intrinsics

- 1) Access integer variables in 'TI3axis'
- 2) Substitute for the 'ibuf' parameter

integer TI3axis(32)

Database Format

```
-----
-- 'TI' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Pocket Toolpath integer info.
-----
integer TpathPocketType      @ TI2Pockt + 2 -- (1)
integer CutTypeTI2t1         @ TI2Pockt + 4
--      (0=spiral,
--      1=Zigzag,
--      2=Profile Every,
--      3=Profile Last,
--      20=Spiral + Profile Every,
--      21=Zigzag + Profile Every,
--      30=Spiral + Profile Every,
--      31=Zigzag + Profile Last)
integer CutDirectTI2t1       @ TI2Pockt + 6
--      (0=CLW, 1=CCLW)
integer ToolTypeTI2t1        @ TI2Pockt + 8
--      (0=ball mill, 1=flat mill)
integer CoolantFlgTI2t1      @ TI2Pockt + 10
--      (0=off, 1=on)
integer SpiralDirTI2t1       @ TI2Pockt + 12
--      (0=in, 1=out)
integer RecutTI2t1           @ TI2Pockt + 14
--      (0=off, 1=recut, 2=Lrecut, 3=Crecut)
integer ProfileApprTI2t1     @ TI2Pockt + 16
--      (0=Spiral, 1=Perpendicular)
integer ToolChangeTI2t1      @ TI2Pockt + 18
--      (0=no, 1=Yes)
integer RoughExceptTI2t1     @ TI2Pockt + 20
--      (0=off, 1=rough, 2=rough/except)
integer ThickTI2t1           @ TI2Pockt + 22
--      (0=no, 1=yes)
integer StepOverTI2t1        @ TI2Pockt + 24
--      (0=40% of tool diam. 1=step, 2=scallop)
integer CutterCompTI2t1      @ TI2Pockt + 26
--      (0=off, 1=right, 2=off)
integer CurrentCPLTI2t1      @ TI2Pockt + 28
integer MagPosTI2t1          @ TI2Pockt + 34
```

Database Format

```
-----  
-- Use with GetSrI, ModSrI, and PutSrI intrinsics  
-- 1) Access integer variables in 'TI2Pockt'  
-- 2) Substitute for the 'ibuf' parameter  
-----
```

```
integer TI2Pockt(32)  
-----
```

```
-----  
-- 'TI' Sub-record definition - (2 1/2 Axis) ---  
-----  
-- 2 1/2 Axis Mcycle Toolpath integer info.  
-----
```

```
integer TPathMcycleType      @ TI2Mcycl + 2 --(2)  
integer PropertyTI2t2        @ TI2Mcycl + 4  
-- (0=off, 1=on)  
integer ToolTypeTI2t2        @ TI2Mcycl + 8  
-- (3=drill, 4=tap)  
integer CoolantFlgTI2t2      @ TI2Mcycl + 10  
-- (0=off, 1=on)  
integer StartPointTI2t2      @ TI2Mcycl + 16  
-- (0=no, 1=Yes)  
integer ToolChangeTI2t2      @ TI2Mcycl + 18  
-- (0=no, 1=Yes)  
integer GCodeTI2t2           @ TI2Mcycl + 20  
-- (0=off, 1=rough, 2=rough/except)  
integer ZTypeTI2t2           @ TI2Mcycl + 22  
-- (0=ZDepth, 1=ZAbs, 3=ZIncr, 3=ZDiam)  
integer DwellTimeTI2t2       @ TI2Mcycl + 24  
integer CurrentCPLTI2t2      @ TI2Mcycl + 28  
integer MagPosTI2t2          @ TI2Mcycl + 34  
-----
```

```
-- Use with GetSrI, ModSrI, and PutSrI intrinsics  
-- 1) Access integer variables in 'TI2Mcycl'  
-- 2) Substitute for the 'ibuf' parameter  
-----
```

```
integer TI2Mcycl(32)  
-----
```

Database Format

```
-----
-- 'TI' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Profile Contour integer info.
-----
integer TPathPContType      @ TI2PCont + 2  --(3)
integer ClosedValTI2t3      @ TI2PCont + 4
--      (0=Between, 1=Complete)
integer CutDirectTI2t3      @ TI2PCont + 6
--      (0=CLW, 1=CCLW)
integer ToolTypeTI2t3       @ TI2PCont + 8
--      (0=ball mill, 1=flat mill)
integer CoolantFlgTI2t3     @ TI2PCont + 10
--      (0=off, 1=on)
integer ProfileApprTI2t3    @ TI2PCont + 16
--      (0=spiral, 1=perpendicular)
integer ToolChangeTI2t3     @ TI2PCont + 18
--      (0=no, 1=yes)
integer AvoidFlgTI2t3       @ TI2PCont + 20
--      (0=no, 1=yes)
integer ThickFlgTI2t3       @ TI2PCont + 22
--      (0=no, 1=Yes)
integer CutterCompTI2t3     @ TI2PCont + 26
--      (0=off, 1=right, 2=left)
integer CurrentCPLTI2t3     @ TI2PCont + 28
integer MagPostTI2t3        @ TI2PCont + 34
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access integer variables in 'TI2PCont'
-- 2) Substitute for the 'ibuf' parameter
-----
integer TI2PCont(32)
-----
```

Database Format

```
-----
-- 'TI' Sub-record definition - (2 1/2 Axis) ---
-----
-- 2 1/2 Axis Point to Point integer info.
-----
integer TPathPnt2PntType    @ TI2PTP + 2 --(4)
--
integer ToolTypeTI2t4        @ TI2PTP + 8
--      (0=ball mill, 1=flat mill)
integer CoolantFlgTI2t4      @ TI2PTP + 10
--      (0=off, 1=on)
--
integer CutterCompTI2t4      @ TI2PTP + 26
--      (0=off, 1=right, 2=left)
integer CurrentCPLTI2t4      @ TI2PTP + 28
integer MagPostTI2t4         @ TI2PTP + 34
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
--   1) Access integer variables in 'TI2PTP'
--   2) Substitute for the 'ibuf' parameter
-----
integer TI2PTP(32)
-----
```

Database Format

```
-----
-- 'TI' Sub-record definition - (2 1/2 Axis) ---
-----
-- 2 1/2 Axis Thread Toolpath integer info.
-----
integer TPathThreadType      @ TI2Thred + 2 -- (7)
integer ThreadTypeTI2t7      @ TI2Thred + 4
--      (0=OD, 1=ID)
integer ApprTypeTI2t7        @ TI2Thred + 6
--      (0=Straight, 1=XFirst, 2=YFirst)
integer ToolTypeTI2t7        @ TI2Thred + 8
--      (11=threading)
integer CoolantFlgTI2t7      @ TI2Thred + 10
--      (0=off, 1=on)
integer CycleTI2t7 @ TI2Thred + 12
--      (0=G33, 1=G76)
integer NumberCutsTI2t7      @ TI2Thred + 14
integer NumStrPassTI2t7      @ TI2Thred + 16
integer ToolChangeTI2t7      @ TI2Thred + 18
--      (0=none, 1=tool change,
--      11=tool change & intermediate)
integer NumOptPassTI2t7      @ TI2Thred + 20
integer CutSideTI2t7         @ TI2Thred + 22
--      (0=above center, 1=below center)
--
integer CurrentCPLTI2t7      @ TI2Thred + 28
--
integer MagPosTI2t7          @ TI2Thred + 34
--
integer UnitsTI2t7           @ TI2Thred + 36
--      (0=inches, 1=metric)
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access integer variables in ITI2Thred'
-- 2) Substitute for the 'ibuf' parameter
-----
integer TI2Thred(32)
-----
```


Database Format

```
-----
-- 'TI' Sub-record definition - (2 1/2 Axis) ---
-----
-- 2 1/2 Axis Lathe Toolpath integer info.
-----
integer TPathLatheType      @ TI2Lathe + 2 -- (8)
integer LatheTypeTI2t8      @ TI2Lathe + 4
--      (0=OD, 1=ID, 2=Face, 3=Profile Last,
--      10=OD groove, 11=ID groove, 12=face groove,
--      30=OD Undercut, 31=ID Undercut)
integer ApprTypeTI2t8       @ TI2Lathe + 6
--      (0=straight, 1=XFirst, 2=YFirst)
integer ToolTypeTI2t8       @ TI2Lathe + 8
--      (5=turn triangle, 6=face triangle
--      7=diamond, 8=square,
--      9=square w/radius, 10=button)
integer CoolantFlgTI2t8     @ TI2Lathe + 10
--      (0=off, 1=on)
integer OffsetTI2t8        @ TI2Lathe + 12
--      (0=radius, 1=Theoretical sharp corner)
integer ProfileTI2t8        @ TI2Lathe + 14
--      (0=off, 1=on, 2=only)
integer CSSTI2t8           @ TI2Lathe + 16
--      (0=off, 1=on)
integer ToolChangeTI2t8     @ TI2Lathe + 18
--      (0=none, 1=toolchange,
--      2=toolchange & intermediate)
integer CutDirectionTI2t8   @ TI2Lathe + 20
--      (1=CRight, 2=CLeft, 3=CUpp, 4=CDown)
integer CutSideTI2t8        @ TI2Lathe + 22
--      (0=above center, 1=below center)
integer CurrentCPLTI2t8     @ TI2Lathe + 28
integer MagPostTI2t8        @ TI2Lathe + 34
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access integer variables in 'TI2Lathe'
-- 2) Substitute for the 'ibuf' parameter
-----
integer TI2Lathe(32)
-----
```

Database Format

```
-----
-- 'TI' Sub-record definition - (2 1/2 Axis) ---
-----
-- 2 1/2 Axis Lcycle Toolpath integer info.
-----
integer TPathLcycleType      @ TI2Lcycl + 2  --(9)
integer ToolTypeTI2t9        @ TI2Lcycl + 8
--      (12=drill, 13=tap)
integer CoolantFlgTI2t9      @ TI2Lcycl + 10
--      (0=off, 1=on)
integer ToolChangeTI2t9      @ TI2Lcycl + 18
--      (0=no, 1=Yes)
integer GCodeTI2t9           @ TI2Lcycl + 20
--      (0=off, 1=rough, 2=rough/except)
integer ZTypeTI2t9           @ TI2Lcycl + 22
--      (0=ZDepth, 1=ZAbs, 3=ZIncr, 3=ZDiam)
integer DwellTimeTI2t9       @ TI2Lcycl + 24
integer CurrentCPLTI2t9      @ TI2Lcycl + 28
integer MagPosTI2t9          @ TI2Lcycl + 34
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access integer variables in 'TI2Lcycl'
-- 2) Substitute for the 'ibuf' parameter
-----
integer TI2Lcycl(32)
-----
```

Database Format

```
-----
-- 'TI' Sub-record definition - (2 1/2 Axis) ---
-----
-- 2 1/2 Axis Lathe Point to Point Toolpath
-- integer info.
-----
integer TPathLPTPType      @ TI2LPTP + 2 --(10)
integer ToolTypeTI2t10     @ TI2LPTP + 8
--      (5=turn triangle, 6=face triangle
--      7=diamond, 8=square, 9=square w/radius,
--      10=button)
integer CoolantFlgTI2t10   @ TI2LPTP + 10
--      (0=off, 1=on)
integer OffsetTI2t10       @ TI2LPTP + 12
--      (0=Radius, 1=Theoritcal sharp corner)
integer CurrentCPLTI2t10   @ TI2LPTP   + 28
integer MagPostTI2t10      @ TI2LPTP   + 34
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
-- 1) Access integer variables in 'TI2LPTP'
-- 2) Substitute for the 'ibuf' parameter
-----
integer TI2LPTP(32)
-----
```

Database Format

```
-----
-- 'TR' Sub-record definition - (3 Axis) ----
-----
-- 3 Axis Toolpath real information.
-----
coord VMinTR3          @ TR3axis + 2
coord VMaxTR3          @ TR3axis + 14
real  ToolDiameterTR3   @ TR3axis + 26
real  CornerRadiusTR3  @ TR3axis + 30
real  SpindleTR3        @ TR3axis + 34
real  FeedrateRapidTR3  @ TR3axis + 38 --(5000.0)
real  FeedrateWorkTR3   @ TR3axis + 42
real  FeedratePlunTR3   @ TR3axis + 46
real  ToleranceTR3      @ TR3axis + 50
real  CollisionTolTR3   @ TR3axis + 54 --(0.0)
real  StepOverTR3       @ TR3axis + 58
real  ScallopHghtTR3    @ TR3axis + 62
real  StockTR3          @ TR3axis + 66
real  CuttingAngTR3     @ TR3axis + 70
real  PrevDiamTR3       @ TR3axis + 74
real  PrevCornerRadTR3   @ TR3axis + 78
real  MaximumDepthTR3   @ TR3axis + 82
real  RecutAngleTR3     @ TR3axis + 86 --(MinAng)
real  CrossCutAngleTR3  @ TR3axis + 90 --(45.0)
real  AbsZRetrctValTR3  @ TR3axis + 94
real  IncZRetrctValTR3  @ TR3axis + 98
real  RetractRatioTR3   @ TR3axis + 102
real  MinSetOverTR3     @ TR3axis + 106 --(VarMin)
real  ZoomValueMapTR3   @ TR3axis + 110 --(0<ZV<1)
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR3axis'
-- 2) Substitute for the Irbuf' parameter
-----
real TR3axis(28)
-----
```

Database Format

```
-----
-- 'TR' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Pocket Toolpath real information.
-----

coord  CPLXaxisVctTR2t1 @ TR2Pockt + 2
coord  CPLYaxisVctTR2t1 @ TR2Pockt + 14
real   ToolDiamTR2t1    @ TR2Pockt + 26
--
real   SpeedTR2t1       @ TR2Pockt + 34
--
real   FeedTR2t1        @ TR2Pockt + 42
--
real   ToleranceTR2t1   @ TR2Pockt + 50
real   RoughValueTR2t1  @ TR2Pockt + 54
real   StepOverTR2t1    @ TR2Pockt + 58
real   ScallopHgtTR2t1  @ TR2Pockt + 62
real   StockTR2t1       @ TR2Pockt + 66
real   ZiZagAngleTR2t1  @ TR2Pockt + 70
real   PocketLevelTR2t1 @ TR2Pockt + 74
real   ApproachTR2t1    @ TR2Pockt + 78
real   CutDepthTR2t1    @ TR2Pockt + 82
real   ZwinTR2t1        @ TR2Pockt + 86 --(MinAng)
real   ZLevelTR2t1      @ TR2Pockt + 90 --(45.0)
real   ThicknessTR2t1   @ TR2Pockt + 94
real   PlungeAngleTR2t1 @ TR2Pockt + 98
real   MaxZDTR2t1       @ TR2Pockt + 102
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR2Pockt'
-- 2) Substitute for the 'rbuf' parameter
-----
real   TR2Pockt(28)
-----
```

Database Format

```
-----
-- 'TR' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Mcycle Toolpath real information.
-----
coord CPLXaxisVctTR2t1 @ TR2mcycl + 2
coord CPLYaxisVctTR2t1 @ TR2Mcycl + 14
real ToolDiamTR2t1 @ TR2Mcycl + 26
real ZDepthTR2t1 @ TR2Mcycl + 30
real SpeedTR2t1 @ TR2Mcycl + 34
--
real FeedTR2t1 @ TR2Mcycl + 42
real StartXTR2t1 @ TR2Mcycl + 46
real StartYTR2t1 @ TR2Mcycl + 50
real StartZTR2t1 @ TR2Mcycl + 54
--
real AvoidDistTR2t1 @ TR2Mcycl + 62
real CutInDistTR2t1 @ TR2Mcycl + 66
real ZDiameterTR2t1 @ TR2Mcycl + 70
real XDepthTR2t1 @ TR2Mcycl + 74
real RetrApprTR2t1 @ TR2Mcycl + 78
real DMinTR2t1 @ TR2Mcycl + 82
real DmaxTR2t1 @ TR2Mcycl + 86
real ZLevelTR2t1 @ TR2Mcycl + 90
real ToolTipAngTR2t1 @ TR2Mcycl + 94
real HoleDepthTR2t1 @ TR2Mcycl + 98
real ToolChangeXTR2t1 @ TR2Mcycl + 102
real ToolChangeYTR2t1 @ TR2Mcycl + 106
real ToolChangeZTR2t1 @ TR2Mcycl + 110
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR2Pockt'
-- 2) Substitute for the Irbuf' parameter
-----
real TR2Mcycl(28)
-----
```

Database Format

```
-----
-- 'TR' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Profile Contour Toolpath real info.
-----
coord CPLXaxisVctTR2t3 @ TR2PCont + 2
coord CPLYaxisVctTR2t3 @ TR2PCont + 14
real ToolIDiamTR2t3 @ TR2Pcont + 26
--
real SpeedTR2t3 @ TR2Pcont + 34
--
real FeedTR2t3 @ TR2Pcont + 42
--
real ToleranceTR2t3 @ TR2Pcont + 50
--
real ZAvoidTR2t3 @ TR2Pcont + 62
real StockTR2t3 @ TR2Pcont + 66
--
real ProfZLevelTR2t3 @ TR2Pcont + 74
real RetrApprTR2t3 @ TR2Pcont + 78
real CutDepthTR2t3 @ TR2Pcont + 82
real ZwinTR2t3 @ TR2Pcont + 86 --(MinAng)
real ZLevelTR2t3 @ TR2Pcont + 90 --(45.0)
real ThicknessTR2t3 @ TR2Pcont + 94
real MaxDTR2t3 @ TR2Pcont + 102
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR2PCont'
-- 2) Substitute for the 'rbuf' parameter
-----
real TR2PCont(28)
-----
```

Database Format

```
-----
-- 'TR' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Profile Point to Point Toolpath
-- real information.
-----
coord CPLXaxisVctTR2t4 @ TR2PPTP + 2
coord CPLYaxisVctTR2t4 @ TR2PPTP + 14
real ToolDiamTR2t4 @ TR2PPTP + 26
--
real SpeedTR2t4 @ TR2PPTP + 34
--
real FeedTR2t4 @ TR2PPTP + 42
--
real ToleranceTR2t4 @ TR2PPTP + 50
--
real ProfZlevelTR2t4 @ TR2PPTP + 74
real RetrApprTR2t4 @ TR2PPTP + 78
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR2PPTP'
-- 2) Substitute for the 'rbuf' parameter
-----
real TR2PPTP(28)
-----
```


Database Format

```
-----
-- 'TR' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Thread Toolpath real information.
-----
coord CPLXaxisVctTR2t7 @ TR2Thred + 2
coord CPLYaxisVctTR2t7 @ TR2Thred + 14
real ToolWidthTR2t7 @ TR2Thred + 26
real SpeedTR2t7 @ TR2Thred + 34
real FeedTR2t7 @ TR2Thred + 42
real ToleranceTR2t7 @ TR2Thred + 50
real PitchTR2t7 @ TR2Thred + 54
real MaxCutDepthTR2t7 @ TR2Thred + 58
real ThreadDepthTR2t7 @ TR2Thred + 62
real TaperTR2t7 @ TR2Thred + 66
real RetractTR2t7 @ TR2Thred + 78
real ApprAngleTR2t7 @ TR2Thred + 82
real RetrAngleTR2t7 @ TR2Thred + 86
real ApprOffsetTR2t7 @ TR2Thred + 90
real RetrOffsetTR2t7 @ TR2Thred + 94
real XGaugeTR2t7 @ TR2Thred + 102
real YGaugeTR2t7 @ TR2Thred + 106
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR2Thred'
-- 2) Substitute for the 'rbuf' parameter
-----
real TR2Thred(28)
-----
```

Database Format

```
-----
-- 'TR' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Lathe Toolpath real information.
-----
coord CPLXaxisVctTR2t8      @ TR2Lathe + 2
coord CPLYaxisVctTR2t8      @ TR2Lathe + 14
real  ToolWidthTR2t8         @ TR2Lathe + 26
--      (Tool Width if tool type is square
--      otherwise, Tool Nose Radius)
real  ToolNoseRadTR2t8       @ TR2Lathe + 30
--      (If tool type is is square w/radius this
--      holds the ToolNoseRadius)
real  SpeedTR2t8             @ TR2Lathe + 34
--
real  FeedTR2t8              @ TR2Lathe + 42
--
real  ToleranceTR2t8         @ TR2Lathe + 50
--
real  MaxTDepthTR2t8         @ TR2Lathe + 58
real  MaxCutDepthTR2t8       @ TR2Lathe + 62
real  StockTR2t8             @ TR2Lathe + 66
real  BackAngleTR2t8         @ TR2Lathe + 70
real  StartRadCCSTR2t8       @ TR2Lathe + 74
--
real  XGaugeTR2t8            @ TR2Lathe + 102
real  YGaugeTR2t8            @ TR2Lathe + 106
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR2Lathe'
-- 2) Substitute for the 'rbuf' parameter
-----
real  TR2Lathe(28)
-----
```

Database Format

```
-----
-- 'TR' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Lcycle Toolpath real information.
-----
coord CPLXaxisVctTR2t9      @ TR2Lcycl + 2
coord CPLYaxisVctTR2t9      @ TR2Lcycl + 14
real  ToolDiamTR2t9          @ TR2Lcycl + 26
real  DepthTR2t9             @ TR2Lcycl + 30
-- (ABS or INC value)
real  SpeedTR2t9             @ TR2Lcycl + 34
--
real  FeedTR2t9              @ TR2Lcycl + 42
--
real  CutInDistTR2t9         @ TR2Lcycl + 66
real  ZDiameterTR2t9         @ TR2Lcycl + 70
real  XDepthTR2t9            @ TR2Lcycl + 74
real  RetrApprTR2t9          @ TR2Lcycl + 78
--
real  TTipAngleTR2t9         @ TR2Lcycl + 94
real  HoleDepthTR2t9         @ TR2Lcycl + 98
real  XGaugeTR2t9            @ TR2Lcycl + 102
real  YGaugeTR2t9            @ TR2Lcycl + 106
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR2Lcycl'
-- 2) Substitute for the 'rbuf' parameter
-----
real  TR2Lcycl(28)
-----
```

Database Format

```
-----
-- 'TR' Sub-record definition - (2 1/2 Axis) ---
-----
-- 2 1/2 Axis Lathe Point to Point
-- Toolpath real information.
-----
coord CPLXaxVctTR2t10 @ TR2LPTP + 2
coord CPLYaxVctTR2t10 @ TR2LPTP + 14
real ToolWidthTR2t10 @ TR2LPTP + 26
-- (Tool Width if tool type is square
-- otherwise, Tool Nose Radius)
real ToolNoseRaTR2t10 @ TR2LPTP + 30
-- (If tool type is is square w/radius this
-- holds the Tool NoseRadius)
real SpeedTR2t10 @ TR2LPTP + 34
--
real FeedTR2t10 @ TR2LPTP + 42
--
real ToleranceTR2t10 @ TR2LPTP + 50
--
real MaxTDepthTR2t10 @ TR2LPTP + 58
--
real XGaugeTR2t10 @ TR2LPTP + 102
real YGaugeTR2t10 @ TR2LPTP + 106
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
-- 1) Access real variables in 'TR2LPTP'
-- 2) Substitute for the 'rbuf' parameter
-----
real TR2LPTP(28)
-----
```

Database Format

```
-----
-- 'TS' Sub-record definition - (2 1/2 Axis) ----
-----
-- 2 1/2 Axis Toolpath Tool Spin information.
-----
-- Holds Tool Spin Direction and Feed Rate.
-- Tool Spin Direction:
--   (-1, 0, 1)
-- Tool Feed Rate:
--   (0=feed, 1=rapid, etc)
--
-- TS subrecord appears in a triple of TS, XN, AC
-- to represent a toolpath. The XN subrecord may
-- hold up to 1350 vertices. There may be more
-- one triple of these subrecords if the toolpath
-- has more than 1350 vertices.
-----
-- Use with GetSrI, ModSrI, and PutSrI intrinsics
--   1) Access integer variables in 'TSsubrec'
--   2) Substitute for the 'ibuf, parameter
-----
integer TSsubrec(2)

integer ToolSpinDirTS @ TSsubrec + 2
integer ToolFeedRateTS @ TSsubrec + 4
-----

-----
-- WD Sub-record definition -----
-----
-- Use with GetSrC, ModSrC, and PutSrC intrinsics
--   1) Assign desired coordinates to 'WDsubrec'
--   2) Substitute for 'WDsubrec' the 'Cbuf'
--       parameter.
-- Note: WD sub-record has a variable length.
-- You may have to adjust size of WDsubrec array
-- for your needs.
-----
coord WDsubrec(100)
-----
```

Database Format

```
-----
-- XD Sub-record definition -----
-----
-- Interpolation data points.
-----
-- If an Nspline or Nsurface was created using
-- interpolation data points, those points are
-- saved in XD subrecords.
-- Nsplines: the 'I' flag in the Nspline header
--           subrecord NC must be 1. The XD
--           subrecord will contain Im' points.
--           (see NC subrecord for more info)
-- Nsurfaces: the 'I' flag in the Nsurface header
--             subrecord NS must be 1. There will
--             'pv' XD subrecords containing 'pu'
--             data points, where:
--             pv = nv - 2 * d
--             pu = nu - 2 * d
--             (see NS subrecord for more info)
-----
-- Use with GetSrc, ModSrc, and PutSrc intrinsics
-- 1) Access data points in array 'XDsubrec'.
-- 2) Substitute 'XDsubrec' for 'cbuf'param.
-- This is a variable length subrecord. You may
-- need to change the dimension of 'XDsubrec'
-- for your needs.
-----
coord XDsubrec(100)
-----
```

Database Format

```
-----
-- XP Sub-record definition -----
-----
-- Cross Hatch Parameters.
-----
-- Holds angle, spacing, offset, pattern, and
-- number of cross hatch boundaries
-- Pattern: -1 = solid fill; 0 = standard;
--           >0 = special
-----
-- Use with GetSrR, ModSrR, and PutSrR intrinsics
--   1) Access variables named below.
--   2) Substitute 'XPsubrec' for 'rbuf'param.
-----
real XPsubrec(4)
real    AngleXP      @ XPsubrec + 2
real    SpacingXP    @ XPsubrec + 6
real    OffsetXP     @ XPsubrec + 10
integer PatternXP    @ XPsubrec + 14
integer NumBoundXP   @ XPsubrec + 16
-----

-----
-- XT Sub-record definition -----
-----
-- 3-D Extents for Nsplines and Nsurfaces.
-----
Holds coordinates of opposite corners of a 3-D
box containing the of a Nspline or Nsurface
and their controlling polygons.
-----
Use with GetSrC, ModSrC, and PutSrC intrinsics
   1) Access variables named below.
   2) Substitute 'XTsubrec' for 'cbuf'param.
-----
coord XTsubrec(4)
coord PolygonMinXT      @ XTsubrec + 2
coord PolygoriMaxXT     @ XTsubrec + 14
coord CurveSurfaceMinXT @ XTsubrec + 26
coord CurveSurfaceMaxXT @ XTsubrec + 38
-----
```

Database Format

Direct Database Access Intrinsic Procedures

The MIB subrecord may be accessed using the intrinsics **ReadEnt**, **WriteEnt**, and **AddEnt**. Use **ReadEnt** to read the MIB record of an entity. A combination of **ReadEnt** and **WriteEnt** should be used to modify an entity's MIB portion. **AddEnt** should be used when adding an entity to the database.

The following is a list of special procedures to read, modify, and write, the Part Data File (PDF) subrecord specified by the last two letters of the name:

RSubRecAC	MSubRecAC	WSubRecAC
RSubRecAP	MSubRecAP	WSubRecAP
RSubRecDS	MSubRecDS	WSubRecDS
RSubRecEX	MSubRecEX	WSubRecEX
RSubRecIL	MSubRecIL	WSubRecIL
RSubRecL1	MSubRecL1	WSubRecL1
RSubRecL2	MSubRecL2	WSubRecL2
RSubRecPA	MSubRecPA	WSubRecPA
RSubRecPX	MSubRecPX	WSubRecPX
RSubRecTD	MSubRecTD	WSubRecTD
RSubRecTF	MSubRecTF	WSubRecTF
RSubRecTX	MSubRecTX	WSubRecTX
RSubRecVN	MSubRecVN	WSubRecVN
RSubRecXH	MSubRecXH	WSubRecXH
RSubRecXN	MSubRecXN	WSubRecXN
RSubRecXZ	MSubRecXZ	WSubRecXZ

The next group of procedures provides access to the remaining PDF subrecords. Most of the subrecords not supported by the above routines are documented above. These routines may be used to access them. The data types supported are Integer, Real, Coord, and String. For each data type, there is a routine to Get (read), Modify, and Put (write) a subrecord. These routines are:

GetSr	ModSr	Putsr
GetSrC	ModSrC	Putsrc
GetSrI	ModSrI	PutsrI
GetSrR	ModSrR	PutsrR
GetSrS	ModSrS	PutsrS

Database Format

AddEnt

Type

Intrinsic Procedure

Database Access

Purpose

Adds the MIB portion a new entity to the end of the database.

Syntax

AddEnt(etype, data(1), mib, ierr)

Parameters

etype: Integer expression (input)
Specifies the entity type number for the new entity. Values are:

1	Line	14	Ellipse
2	String	15	Construction line
3	Arc	16	Curve (cpole)
4	Text	17	Surface (spole)
5	Point	18	Plane
6	Linear dimension	30	Nspline
7	Label, point dimension	31	Nsurface
8	Radial dimension	35	3-axis toolpath
9	Angular dimension	36	2-1/2-axis toolpath
10	Cross-hatching	145	Display image
11	Figure instance	146	View
12	Diameter dimension	147	Figure image list
13	MView (multiple view)	148	Extents

data: Integer array of 5 elements (input)
Gives the attributes for the new entity. The array elements are:

- 1 Layer number-, if negative one, use active layer.
- 2 View of visibility.
- 3 Group number.
- 4 Line font number.
- 5 Color number, if 0 use current color.

mib: Integer4 variable (input/output)
Returns the Master Index Block (MIB) number of the new entity.

ierr: Integer variable (input/output)
Returns an error number after operation; zero indicates a successful operation. Other values indicate an error.

Database Format

GetSr

Type

Intrinsic Procedure

Database Access

Purpose

Allows direct reading of database subrecords that consist of only integer data. No data type conversion or variable basing is necessary.

WARNING: This procedure is for advanced users only.

Syntax

GetSr(*mib*, *srtype*, *occur* *nbytesget*, *nbytesgot*, *buf*(1), *error*)

Parameters

<i>mib</i> :	Integer4 expression (input) Specifies the MIB number of the entity whose subrecord is read from.
<i>srtype</i> :	Strino expression of 2 characters (input) Specifies the type of subrecord to retrieve. If the parameter is an empty string, the procedure retrieves any type of subrecord.
<i>occur</i> :	Integer expression (input) Specifies which occurrence of the <i>srtype</i> subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one <i>srtype</i> subrecord to read. If this is not the case, set the <i>occur</i> parameter to one.
<i>nbytesget</i> :	Integer expression (input) Specifies the number of bytes to read from the subrecord. If <i>nbytesget</i> equals negative one, all the bytes in the subrecord will be read. Note that there are two bytes of data per integer.
<i>nbytesgot</i> :	Integer variable (input/output) Returns the number of bytes actually retrieved from the subrecord.
<i>buf</i> :	String variable, output Data retrieved from the database subrecord is placed in this variable. It may contain several different data types. To extract this information, variables of the appropriate types should be based to the <i>buf</i> string variable.

Database Format

error: Integer variable (input/output)
Returns the error condition:

- 0 No errors were found.
- 1 An IO error was found.
- 2 There are not enough bytes to read
(*nbytesget* is too big).
- 3 The subrecord was not found.
- 4 An invalid MIB number was given.

Example

```
Get_Sr(Mib, "PL", 1, -1, NBytesGot, Buf(1), Error)
```

Database Format

GetSrC

Type

Intrinsic Procedure

Database Access

Purpose

Allows direct reading of database subrecords that consist of only coordinate data. No data type conversion or variable basing is necessary.

WARNING: This procedure is for advanced users only.

Syntax

GetSrC(*mib*, *srtype*, *occur* *nbytesget*, *nbytesgot*, *cbuff*(1), *error*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that the subrecord is read from.
- srtype*: String expression of 2 characters (input)
Specifies the type of subrecord to retrieve. If the parameter is an empty string, the procedure retrieves any type of subrecord.
- occur*: Integer expression (input)
Speeifies which occurrence of the *srtype* subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one *srtype* subrecord to read. If the PDF portion of the entity record does not have more than one *srtype* subrecord to read, set the *occur* parameter to one.
- nbytesget*: Integer expression (input)
Specifies the number of bytes to read from the subrecord. If *nbytesget* equals negative one, all the bytes in the subrecord will be read. Note that there are 12 bytes of data per coordinate.
- nbytesgot*: Integer variable (input/output)
This returns the number of bytes actually retrieved from the subrecord.

Database Format

cbuf: Coordinate array of *nbytes*/12 elements (input/output)
This is the buffer that returns the subrecord data. You must declare *cbuf* to have enough array elements for the largest subrecord that will be retrieved. The maximum size is 1.000 elements.

error: Integer variable (input/output)

Returns the error condition.

- 0 No errors were found.
- 1 An IO error was found.
- 2 There are not enough bytes read
(*nbytesget* is too big).
- 3 The subrecord was not found.
- 4 An invalid MIB number was given.

Examples

```
Get_Sr_C(StrMib,"XN", 1, -1 ,NbytesGot, Verts (1), Error)  
Get_Sr_C(LinMib,"XZ", 1, 24, NbytesGot, Verts (1), Error)
```

Database Format

GetSrl

Type

Intrinsic Procedure

Database Access

Purpose

Allows direct reading of database subrecords that consist of only integer data. No data type conversion or variable basing is necessary.

WARNING: This procedure is for advanced users only.

Syntax

GetSrl(*mib*, *srtype*, *occur*, *nbytesget*, *nbytesgot*, *ibuff*(1), *error*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that the subrecord is read from.
- srtype*: String expression of 2 characters (input)
Specifies the type of subrecord to retrieve. If the parameter is an empty string, the procedure retrieves any type of subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the *srtype* subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one *srtype* subrecord to read. If the PDF portion of the entity record does not have more than one *srtype* subrecord to read, set the *occur* Parameter to one.
- nbytesget*: Integer expression (input)
Specifies the number of bytes to read from the subrecord. If *nbytesget* equals negative one, all the bytes in the subrecord will be read. Note that there are two bytes of data per integer.
- nbytesgot*: Integer variable (input/output)
Returns the number of bytes actually retrieved from the subrecord.

Database Format

ibuf: Integer array of *nbytesget*/2 elements (input/output)
This is the buffer that returns the subrecord data. You must declare *ibuf* to have enough array elements for the largest subrecord to be retrieved. The maximum size is 6.000 elements.

error: Integer variable (input/output)
Returns the error condition:

- 0 No errors were found.
- 1 An IO error was found.
- 2 There are not enough bytes to read
(*nbytesget* is too big).
- 3 The subrecord was not found.
- 4 An invalid MIB number was given.

Example

```
Get_Sr_I(Mib, PL, 1, -1, NBytesGot, IBuf(1), Error)
```

Database Format

GetSrR

Type

Intrinsic Procedure

Database Access

Purpose

Allows direct reading of database subrecords that consist of only real data. No data type conversion or variable basing is necessary.

WARNING: This procedure is for advanced users only.

Syntax

GetSrR(*mib*, *srtype*, *occur*, *nbytesget*, *nbytesgot*, *rbuf*(1), *error*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that the subrecord is read from.
- srtype*: String expression of 2 characters (input)
Specifies the type of subrecord to retrieve. If the parameter is an empty string, the procedure retrieves any type of subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the *srtype* subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one *srtype* subrecord to read. If the PDF portion of the entity record does not have more than one *srtype* subrecord to read, set the *occur* Parameter to one.
- nbytesget*: Integer expression (input)
Specifies the number of bytes to read from the subrecord. If *nbytesget* equals a negative one, all the bytes in the subrecord will be read. Note: there are four bytes of data per real number.
- nbytesgot*: Integer variable (input/output)
This returns the number of bytes retrieved from the subrecord.
- rbuf*: Real array of *nbytesget*/4 elements (input/output)
This is the buffer that returns the subrecord data. You must declare *rbuf* to have enough array elements for the largest subrecord to be retrieved. The maximum size is 3,000 elements.

Database Format

error: Integer variable (input/output)

Returns the error condition:

- 0 No errors were found.
- 1 An IO error was found.
- 2 There are not enough bytes to read
(*nbytesget* is too big).
- 3 The subrecord was not found.
- 4 An invalid MIB number was given.

Example

```
Get_Sr_R(Mib, "EP", 1, -1, NBytesGot, AVT(1), Error)
```

Database Format

GetSrS

Type

Intrinsic Procedure

Database Access

Purpose

Allows direct reading of database subrecords that consist only of string data. No data type conversion or variable basing is necessary. Data from the subrecord is stored starting in the string data field. You must update the current length field of the character string in your program.

WARNING: This procedure is for advanced users only.

Syntax

GetSrS(*mib*, *srtype*, *occur* *nbytesget*, *nbytesgot*, *sbuf*, *error*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that the subrecord is read from.
- srtype*: String expression of 2 characters (input)
Specifies the type of subrecord to retrieve. If the parameter is an empty string, the procedure retrieves any type of subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the *srtype* subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one *srtype* subrecord to read. If the PDF portion of the entity record does not have more than one *srtype* subrecord to read, set the *occur* parameter to one.
- nbytesget*: Integer expression (input)
Specifies the number of bytes to read from the subrecord. If *nbytesget* equals a negative one, all the bytes in the subrecord have been read.
- nbytesgot*: Integer variable (input/output)
This returns the number of bytes actually retrieved from the subrecord. You should update the length attribute of *sbuf* with this value. See the example below.

Database Format

sbuf: String variable of *nbytesget* characters (input/output)
This is the buffer that returns the subrecord data. You must declare *sbuf* to have enough characters for the largest subrecord to be retrieved. The maximum size is 12.000 bytes.

error: Integer variable (input/output)
Returns the error condition:

- 0 No errors were found.
- 1 An IO error was found.
- 2 There are not enough bytes to read (*nbytesget* is too big).
- 3 The subrecord was not found.
- 4 An invalid MIB number was given.

Example

```
Get_Sr_S(PropMib, 'D2', 2, - 1, NBytesGot, PropStr, Error)  
PropStr.length = NBytesGot
```

Database Format

ModSr

Type

Intrinsic Procedure

Database Access

Purpose

Modifies a database subrecord that consists of integer data only. Use this procedure when you want to change integer information about an entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database

Syntax

ModSr(mib, srtype, nn, nbyte, buf(1), error)

Parameters

- mib:* Integer4 expression (input) Specifies the MIB number of the entity whose subrecord is to be modified.
- srtype:* String expression of 2 characters (input)
Specifies the two-character subrecord type.
- nn:* Integer expression (input)
Get the nnth occurrence of subrecord of the type *srtype*.
- nbyte:* Integer expression (input)
Specifies the number of bytes of valid data in *buf*.
- buf:* String expression (input)
Specifies new data to replace old data received from the subrecord.
- error:* Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

ModSrC

Type

Intrinsic Procedure

Database Access

Purpose

Modifies a database subrecord that consists of coordinate data only. Use this procedure when you want to change coordinate information about an entity in the PDF portion of the part database.

ModSrC is similar to the **ModSr** procedure, except that no variable basing or data type conversion is necessary.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database.

Syntax

ModSrC(*mib*, *srtype*, *occur*, *nbytesmod*, *cbuf*(1), *error*)

Parameters

- mib*: Integer4 expression (input). Specifies the MIB number of the entity whose subrecord is to be modified.
- srtype*: String expression of 2 characters (input)
Specifies the two-character subrecord type.
- occur*: Integer expression (input). Specifies which occurrence of the *srtype* subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one *srtype* subrecord to read. If the PDF portion of the entity record does not have more than one *srtype* subrecord to read, set *occur* to one.
- nbytesmod*: Integer expression (input) Specifies the number of bytes of valid data in *cbuf*.
- cbuf*: Coordinate array of *nbytesmod* /12 elements (input/output)
Specifies the buffer holding new subrecord data. The maximum size is 1.000 elements.
- error*: Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

ModSrl

Type

Intrinsic Procedure

Database Access

Purpose

Modifies a database subrecord that consists of integer data only. Use this procedure when you want to change integer information about an entity in the PDF portion of the part database.

ModSrl is similar to the **ModSr** procedure except that no variable basing or data type conversion is necessary.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database

Syntax

ModSrl(*mib*, *srtype*, *occur*, *nbytesmod*, *ibuf*(1), *error*)

Parameters

<i>mib</i> :	Integer4 expression (input) Specifies the MIB number of the entity whose subrecord is to be modified.
<i>srtype</i> :	String expression of 2 characters (input) Specifies the two-character subrecord type.
<i>occur</i> :	Integer expression (input) Specifies which occurrence of the <i>srtype</i> subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one <i>srtype</i> subrecord to read. If the PDF portion of the entity record does not have more than one <i>srtype</i> subrecord to read, set <i>occur</i> to one.
<i>nbytesmod</i> :	Integer expression (input) Specifies the number of bytes of valid data in <i>ibuf</i> .
<i>ibuf</i> :	Integer array of <i>nbytesmod</i> / 2 elements (input/output) Specifies the buffer holding new subrecord data. The maximum size is 6.000 elements.
<i>error</i> :	Integer variable (input/output) Returns the error condition: <div><div>0</div>no errors were found. <div>1</div>an IO error was found. <div>3</div>the subrecord was not found. <div>4</div>an invalid MIB number was given.</div>

Database Format

ModSrR

Type

Intrinsic Procedure

Database Access

Purpose

Modifies a database subrecord that consists of real data only. Use this procedure when you want to change real number information about an entity in the PDF portion of the part database.

ModSrR is similar to the **ModSr** procedure except that no variable basing or data type conversion is necessary.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database.

Syntax

ModSrR(*mib*, *srtype*, *occur*, *nbytesmod*, *rbuf*(1), *error*)

Parameters

<i>mib</i> :	Integer4 expression (input) Specifies the MIB number of the entity whose subrecord is to be modified.								
<i>srtype</i> :	String expression of 2 characters (input) Specifies the two-character subrecord type.								
<i>occur</i> :	Integer expression (input) Specifies which occurrence of the <i>srtype</i> subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one <i>srtype</i> subrecord to read. If the PDF portion of the entity record does not have more than one <i>srtype</i> subrecord to read, set <i>occur</i> to one.								
<i>nbytesmod</i> :	Integer expression (input) Specifies the number of bytes of valid data in <i>rbuf</i> . The maximum size is 3.000 elements.								
<i>rbuf</i> :	Coordinate array of <i>nbytesmod</i> /4 elements (input/output) Specifies the buffer holding new subrecord data.								
<i>error</i> :	Integer variable (input/output) Returns the error condition: <table><tr><td>0</td><td>no errors were found.</td></tr><tr><td>1</td><td>an IO error was found.</td></tr><tr><td>3</td><td>the subrecord was not found.</td></tr><tr><td>4</td><td>an invalid MIB number was given.</td></tr></table>	0	no errors were found.	1	an IO error was found.	3	the subrecord was not found.	4	an invalid MIB number was given.
0	no errors were found.								
1	an IO error was found.								
3	the subrecord was not found.								
4	an invalid MIB number was given.								

Database Format

ModSrS

Type

Intrinsic Procedure

Database Access

Purpose

Modifies a database subrecord that consists of character string data only. Only the actual string data is written to the subrecord, the string variable attributes LEN and .LENGTH are not written to the subrecord. If the attributes are needed, you must explicitly write them to the database by basing another string variable to the *sbuf* parameter.

Use this procedure when you want to change character string information about an entity in the PDF portion of the part database.

ModSrS is similar to the **ModSr** procedure except that no variable basing or data type conversion is necessary.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database

Syntax

ModSrS(*mib*, *srtype*, *occur*, *nbytesmod*, *sbuf*, *error*)

Parameters

<i>mib</i> :	Integer4 expression (input) Specifies the MIB number of the entity whose subrecord is to be modified.
<i>srtype</i> :	String expression of 2 characters (input) Specifies the two-character subrecord type.
<i>occur</i> :	Integer expression (input) Specifies which occurrence of the <i>srtype</i> subrecord to get. It is used if the Part Data File (PDF) portion of the entity record has more than one <i>srtype</i> subrecord to read. If the PDF portion of the entity record does not have more than one <i>srtype</i> subrecord to read, set <i>occur</i> to one.
<i>nbytesmod</i> :	Integer expression (input) Specifies the number of bytes of valid data in <i>sbuf</i> . Note that it is this parameter, and not the <i>sbuf</i> .LENGTH attribute, that defines how many bytes are modified.

Database Format

- sbuf:* String variable of *nbytesmod* characters (input/output)
Specifies the buffer holding new subrecord data. The maximum length is 12.000 bytes.
- error:* Integer variable (input/output) . Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

MSubrecAC

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an AC type subrecord. The AC subrecord holds data about an arc entity. Use this procedure when you want to change the information about an arc entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecAC(*mib*, *occur*, *error*, *transform(1)*, *radius*, *abeg*, *aend*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity that contains the AC subrecord.

occur: Integer expression (input)
Specifies which occurrence of the AC subrecord to modify. If there is one or more AC subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next AC subrecord. An error three returns if there are no more AC subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

- error.:* Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- transform.:* Real array of 12 elements (input/output)
Specifies the arc's view transform. This transformation matrix defines the plane which the arc lines in. Usually, this is the view the arc was created in. Elements 10, 11, and 12 represent the origin of the arc.
- radius.:* Real expression (input)
Specifies the radius of the arc in database units.
- abeg.:* Real expression (input)
Specifies the beginning angle of the arc given in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.
- aend.:* Real expression (input)
Specifies the ending angle of the arc given in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.

Database Format

MSubrecAP

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an AP type subrecord. The AP subrecord holds data about partial arcs used in angular dimensions. Use this procedure when you want to change information about an angular dimension entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecAP(*mib, occur, error, org, radius1, abeg1, aend1, radius2, abeg2, aend2*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the AP subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the AP subrecord to modify. If there is one or more AP subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next AP subrecord. An error three returns if there are no more AP subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

- error:* Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- org:* Coordinate expression (input)
Specifies the origin of the arc(s) in model coordinates.
- radius1:* Real expression (input)
Specifies the radius of the first arc in database units.
- abeg1:* Real expression (input)
Specifies the beginning angle of the first arc in radians. Angle zero begins at the X-axis (defined by the arc's view transform) and increases counterclockwise.
- aend1:* Real expression (input)
Specifies the ending angle of the first arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.
- radius2:* Real expression (input)
Specifies the radius of the second arc in database units.
- abeg2:* Real expression (input)
Specifies the beginning angle of the second arc in radians. Angle zero begins at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.
- aend2:* Real expression (input)
Specifies the ending angle of the second arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.

Database Format

MSubrecDS

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of a DS type subrecord. The DS subrecord holds data about the display image that was saved in the Personal Designer command SAVE IMAGE. Use this procedure when you want to change information about an image entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecDS(*mib*, *occur*, *error*, *extents(1)*, *scrscr*, *viewno*, *dispno*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity containing the DS subrecord.

occur: Integer expression (input)
Specifies which occurrence of the DS subrecord to modify. If there is one or more DS subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next DS subrecord. An error three will be returned if there are no more DS subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

- error:* Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 2 the subrecord was not found.
 - 4 an invalid MIB number was given.
- extents:* Real array of 4 elements (input/output)
Specifies the maximum and minimum values used in a display image. They are given as X minimum and maximum followed by Y minimum and maximum.
- scrscl:* Real expression (input)
Specifies the display image's screen scale.
- viewno:* Integer expression (input)
Specifies the view number associated with the display image.
- dispno:* Integer expression (input)
Specifies the display image number.

Database Format

MSubrecEX

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an EX type subrecord. The EX subrecord holds data about the extents of a part or figure in 3D space. Use this procedure when you want to change information about extents, figure instance, or figure image entities, in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecEX(*mib*, *occur*, *error*, *extents(1)*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity that contains the EX subrecord.

occur: Integer expression (input)
Specifies which occurrence of the EX subrecord to modify. If there is one or more EX subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next EX subrecord. An error three will be returned if there are no more EX subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 The subrecord was not found.
- 4 an invalid MIB number was given.

extents: Real array of 24 elements (input/output)
Specifies the range of X, Y, and Z values used in a drawing or figure. These values are always model space coordinates.
This parameter can be used in two ways.

1. The first holds the eight X, Y, and Z values that define the corners of an imaginary cube which surrounds all part or figure geometry.
2. The second method, used by surfacing in Personal Designer, holds the six minimum and maximum values used by any entity in the database. These are specified in the following order: minimum X, maximum X, minimum Y, maximum Y, minimum Z, maximum Z. The same imaginary cube can be created by generating planes normal to X, Y, and Z axes and passing through the six points.

Database Format

MSubrecIL

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an IL type subrecord. The IL subrecord holds data about a figure image that is to be inserted in the current part. Use this procedure when you want to change information about a figure image entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecIL(*mib*, *occur*, *error*, *figname*, *figmib*, *entcount*, *figdate*, *figtime*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity that contains the IL subrecord.

occur: Integer expression (input)
Specifies which occurrence of the IL subrecord to modify. If there is one or more IL subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next IL subrecord. An error three will be returned if there are no more IL subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

<i>error:</i>	Integer variable (input/output) This parameter returns the error condition: 0 no errors were found. 1 an IO error was found. 3 the subrecord was not found. 4 an invalid MIB number was given.
<i>figname:</i>	String expression of 64 characters (input) Specifies the figure file name including the path name.
<i>figmib:</i>	Integer4 expression (input) Specifies the MIB number of the first entity in the figure image list.
<i>entcount:</i>	Integer expression (input) Specifies the number of entities in the figure image list starting with the <i>mib</i> parameter.
<i>figdate:</i>	Integer expression (input) Specifies the file date of the figure part file. This is the date the file was last modified. See Appendix E, Internal Data Storage Format, for more information about the date format.
<i>figtime:</i>	Integer expression (input) Specifies the file time of the figure part file. This is the time the file was last modified. See Appendix E, Internal Data Storage Format, for more information about the time format.

Database Format

MSubrecL1

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an L 1 type subrecord. The L1 subrecord holds data about dimension extension line one. Use this procedure when you want to change extension line information about a dimension entity in the PDF portion of the part database.

WARNING; This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MsubrecL1(*mib, occur, error pnt1, pnt2*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity that contains the L1 subrecord.

occur: Integer expression (input)
Specifies which occurrence of the L1 subrecord to modify. If there is one or more L1 subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next L1 subrecord. An error three will be returned if there are no more L1 subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

- error:* Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- pnt1:* Coordinate expression (input)
Specifies the first endpoint of the extension line.
- pnt2:* Coordinate expression (input)
Specifies the second endpoint of the extension line.

Database Format

MSubrecL2

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an L2 type subrecord. The L2 subrecord holds data about dimension extension line two. Use this procedure when you want to change extension line information about a dimension entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecL2(*mib, occur, error pnt1, pnt2*)

Parameters

- mib:* Integer4 expression (input) Specifies the MIB number of the entity that contains the L2 subrecord.
- occur:* Integer expression (input) Specifies which occurrence of the L2 subrecord to modify. If there is one or more L2 subrecord for a given entity, use *occur* to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter. Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next L2 subrecord. An error three will be returned if there are no more L2 subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.
- error:* Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

- pnt1*: Coordinate expression (input)
 Specifies the first endpoint of the extension line.
- pnt2*: Coordinate expression (input)
 Specifies the second endpoint of the extension line.

Database Format

MSubrecPA

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of a PA type subrecord. The PA subrecord holds data about entity properties. Use this procedure when you want to change property information about any entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecPA(*mib, occur, error, pname, ptype, pval*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity that contains the PA subrecord.

occur: Integer expression (input)
Specifies which occurrence of the PA subrecord to modify. If there is one or more PA subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next PA subrecord. An error three will be returned if there are no more PA subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

pname: String expression of 8 characters (input)
Specifies the property name.

ptype: String expression of 7 characters (input)
Specifies the property type.

pval: String expression of 100 characters (input)
Specifies the property value.

Database Format

MSubrecPX

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of a PX type subrecord. The PX subrecord holds data about the point entity. Use this procedure when you want to change information about a point entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecPX(*mib*, *occur*, *error*, *pnt*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity that contains the PX subrecord.

occur: Integer expression (input)
Speeifies which occurrence of the PX subrecord to modify. If there is one or more PX subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next PX subrecord. An error three will be returned if there are no more PX subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

pnt: Coordinate expression(input)
Specifies the X, Y, Z values of a point in model coordinates.

Database Format

MSubrecTD

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of a TD type subrecord. The TD subrecord holds data about text format and orientation. Note that it does not hold the actual text itself, which is usually in a TX subrecord. Use this procedure when you want to change information about text strings, dimensions, labels, and MView entities in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubreeTD(*mib*, *occur*, *error*, *transform(1)*, *height*, *width*, *linesp*, *just(1)*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the TD subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the TD subrecord to modify. If there is one or more TD subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next TD subrecord. An error three will be returned if there are no more TD subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

transform: Real array of 12 elements (input/output)
Specifies the text's view transform. This transformation matrix defines the plane that the text lies in. Usually, this is the view the text was created in. Elements 10, 11, and 12 are the origin of the arc. For TD subrecords, only elements 1 through 6 and 10 through 12 of the view transform are used. Elements 1 through 6 are the X-axis and Y-axis cosines. Elements 10 through 12 are the origin of the text.

height: Real expression (input)
Specifies the text height in database units.

width: Real expression (input)
Specifies the text width in database units.

linesp: Real expression (input)
Specifies the line spacing, in database units.

just: Integer array of 6 elements (input/output)
This array specifies text justification.
Valid values for *just(1)* are:

- 1 left justification.
- 2 right justification.
- 3 center justification.

just(2) is an associativity flag for dimension text. Values are:

- 1 has D4 subrecord.
- 0 has no associativity

just(3) is MView number

just(4) is text font number

just(5) is text slant angle

just(6) is reserved; do not use.

Database Format

MSubrecTF

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of a TF type subrecord. The TF subrecord holds transformation data about a figure instance which is inserted into a part. Use this procedure when you want to change information about a figure instance entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecTF(*mib*, *occur*, *error*, *transform(1)*, *figuremib*)

Parameters

- mib*: Integer4 expression (input) Specifies the MIB number of the entity that contains the TF subrecord.
- occur*: Integer expression (input) Specifies which occurrence of the TF subrecord to modify. If there is one or more TF subrecord for a given entity, use *occur* to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter. Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next TF subrecord. An error three will be returned if there are no more TF subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.
- error*: Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

- transform:* Real array of 15 elements (input/output)
The first nine elements of the transformation matrix specify the figure viewing matrix. Elements 10, 11, and 12 are the figure origin. Elements 13, 14, and 15 are the figure's X, Y, and Z scale factors.
- figuremib:* Integer4 expression (input)
Specifies the MIB number of the figure image list entity.

Database Format

MSubrecTX

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of a TX type subrecord. The TX subrecord holds the actual text of a text string. Use this procedure when you want to change information about text, dimension, label, MView, curve, and surface entities in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecTX(*mib*, *occur*, *error*, *textstring*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that contains the TX subrecord.

occur: Integer expression (input)
Specifies which occurrence of the TX subrecord to modify. Use this when there is one or more TX subrecord for a given entity. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, automatically access the next TX subrecord. An *error* three is returned if there are no more TX subrecords. This programming tip is faster than incrementing, the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

textstring: String expression (input) Specifies the text string to be modified.

Database Format

MSubrecVN

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of a VN type subrecord. The VN subrecord holds data about a defined view. Note that there is no entity record and therefore no VN subrecord for Personal Designer's six predefined views. Use this procedure when you want to change information about a view entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecVN(*mib, occur, error, transform(1)*)

Parameters

- mib:* Integer4 expression (input)
Specifies the MIB number of the entity that contains the VN subrecord.
- occur:* Integer expression (input)
Specifies which occurrence of the VN subrecord to modify. If there is one or more VN subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next TX subrecord. An error three will be returned if there are no more TX subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

transform: Real array of 15 elements (input/output)
Specifies the view transformation matrix. Only the first nine elements of transform are used. The offset and scaling factors are not modified.

Database Format

MSubrecXH

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an XH type subrecord. The XH subrecord holds data about cross-hatch lines. Use this procedure when you want to change information about cross-hatch line entities in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecXH(*mib, occur error, endpoints, npnts*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the XH subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the XH subrecord to modify. If there is one or more XH subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next XH subrecord. An error three will be returned if there are no more XH subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

endpoints: Coordinate array of *npnts* elements (input/output)
Specifies the endpoints of the cross-hatch lines. This parameter contains pairs of endpoints rather than the cross-hatch boundaries.

npnts: Integer expression (input)
Specifies the number of endpoints in the *endpoints* parameter. The maximum number of lines allowed is 500; and therefore 1.000 endpoints for each cross-hatch area. This should be an even number since the points are pairs.

Database Format

MSubrecXN

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an XN type subrecord. The XN subrecord holds the vertices of a string. A string consists of two or more line segments. Use this procedure when you want to change information about a string, label, arrow, dimension, cross-hatch, MView, curve, or surface entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecXN(*mib, occur, error, vertices(1), nvert*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity that contains the XN subrecord.

occur: Integer expression (input)
Specifies which occurrence of the XN subrecord to modify. If there is one or more XN subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next XN subrecord. An error three will be returned if there are no more XN subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

<i>error:</i>	Integer variable (input/output) This parameter returns the error condition: 0 no errors were found. 1 an IO error was found. 3 the subrecord was not found. 4 an invalid MIB number was given.
<i>vertices:</i>	Coordinate array of <i>nvert</i> elements (input/output) Specifies string vertices in model coordinates. The order of the vertices in this array is the order in which the string was created.
<i>nvert:</i>	Integer expression (input) This is the number of vertices in the <i>vertices</i> parameter.

Database Format

MSubrecXZ

Type

Intrinsic Procedure

Database Access

Purpose

Modifies the contents of an XZ type subrecord. 'Me XZ subrecord holds the endpoints of a line. Use this procedure when you want to change information about a line entity in the PDF portion of the part database.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

MSubrecXZ(*mib*, *occur*, *error*, *pnt1*, *pnt2*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the XZ subrecord.
- occur*: Integer expression (input) Specifies which occurrence of the XZ subrecord to modify. If there is one or more XZ subrecord for a given entity, use *occur* to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter. Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next XN subrecord. An error three will be returned if there are no more XN subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.
- error*: Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

- pnt1*: Coordinate expression (input)
 Specifies the first endpoint of the line in model coordinates.
- pnt2*: Coordinate expression (input)
 Specifies the second endpoint of the line in model coordinates.

Database Format

PutSr

Type

Intrinsic Procedure

Database Access

Purpose

This procedure puts, or adds, a database subrecord that consists of only integer data to the PDF portion of the database.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database.

Syntax

PutSr(*mib*, *snype*, *nbytes*, *buf*(1), *error*)

Parameters

<i>mib</i> :	Integer4 expression (input) Specifies the MIB number of the entity to put the new subrecord on.
<i>srtype</i> :	String expression of 2 characters (input) Specifies the two-character subrecord type.
<i>nbyte</i> :	Integer expression (input) Specifies the number of bytes of valid data in <i>buf</i> for the new subrecord.
<i>buf</i> :	String expression (input) This buffer specifies the new subrecord data.
<i>error</i> :	Integer variable (input/output) Returns the error condition: <div><div>0</div>no errors were found. <div>1</div>an IO error was found. <div>4</div>an invalid MIB number was given.</div>

Database Format

PutSrC

Type

Intrinsic Procedure

Database Access

Purpose

Puts, or adds, a database subrecord that consists of only coordinate data to the PDF portion of the database. This procedure is similar to the **PutSr** procedure, except that no variable basing or data type conversion is necessary.

There are two ways to use the **PutSrC** procedure:

1. The first is for adding a subrecord to an existing entity.
2. The second is for adding a whole entity to the database. In this latter case, you must first make a call to the **AddEnt** intrinsic procedure to set up the MIB portion of the database. Then you make calls to "put" or write database subrecords to the end of the PDF portion of the database.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database.

Syntax

PutSrC(*mib*, *srtype*, *nbytes*, *cbuff*(1), *error*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity to put the new subrecord on.

srtype: String expression of 2 characters (input) Specifies the two-character subrecord type.

nbytes: Integer expression (input) Specifies the number of bytes of valid data in *cbuff* for the new subrecord.

Database Format

PutSrl

Type

Intrinsic Procedure

Database Access

Purpose

This procedure puts, or adds, a database subrecord that consists of only integer data to the PDF portion of the database. This procedure is similar to the **PutSr** procedure except that no variable basing or data type conversion is necessary.

There are two ways to use the **PutSrl** procedure:

1. The first way is for adding a subrecord to an existing entity.
2. The second is for adding a whole entity to the database. In this case, you must first make a call to the **AddEnt** intrinsic procedure to set up the MIB portion of the database. Then you make calls to "put" or write database subrecords to the end of the PDF portion of the database.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database.

Syntax

PutSrl(*mib*, *srtype*, *nbytes*, *ibuf*(1), *error*)

Parameters

<i>mib</i> :	Integer4 expression (input) Specifies the MIB number of the entity to put the new subrecord on.
<i>srtype</i> :	String expression of 2 characters (input) Specifies the two-character subrecord type.
<i>nbytes</i> :	Integer expression (input) Specifies the number of bytes of valid data in <i>ibuf</i> for the new subrecord.
<i>ibuf</i> :	Integer array of <i>nbytes</i> / 2 elements (input/output) This buffer specifies the new subrecord data. The maximum buffer size is 6.000 elements.

Database Format

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

Database Format

PutSrR

Type

Intrinsic Procedure

Database Access

Purpose

Puts, or adds, a database subrecord that consists of only real data to the PDF portion of the database. This procedure is similar to the **PutSr** procedure except that no variable basing or data type conversion is necessary.

There are two ways to use the **PutSrR** procedure:

1. The first way is for adding a subrecord to an existing entity.
2. The second is for adding a whole entity to the database. In this case, you must first make a call to the **AddEnt** intrinsic procedure to set up the MIB portion of the database. Then you make calls to "put" or write database subrecords to the end of the PDF portion of the database.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database.

Syntax

PutSrR(*mib*, *srtype*, *nbytes*, *rbuf*(1), *error*)

Parameters

mib: Integer4 expression (input)
Specifies the MIB number of the entity to put the new subrecord on.

srtype: String expression of 2 characters (input)
Specifies the two-character subrecord type.

nbytes: Integer expression (input)
Specifies the number of bytes of valid data in *rbuf* for the new subrecord.

Database Format

PutSrS

Type

Intrinsic Procedure

Database Access

Purpose

Puts, or adds, a database subrecord that consists of only string data to the PDF portion of the database. This procedure is similar to the **PutSr** procedure except that no variable basing or data type conversion is necessary.

There are two ways to use the **PutSrS** procedure:

1. The first way is for adding a subrecord to an existing entity.
2. The second is for adding a whole entity to the database. In the latter case, you must first make a call to the **AddEnt** intrinsic procedure to set up the MIB portion of the database. Then you make calls to "put" or write database subrecords to the end of the PDF portion of the database.

WARNING: This procedure is for advanced users only. Improper use could damage or destroy your part database.

Syntax

PutSrS(*mib*, *srtype*, *nbytes*, *sbuf*, *error*)

Parameters

<i>mib</i> :	Integer4 expression (input) Specifies the MIB number of the entity to put the new subrecord on.
<i>srtype</i> :	String expression of 2 characters (input) Specifies the two-character subrecord type.
<i>nbytes</i> :	Integer expression (input) Specifies the number of bytes of valid data in <i>sbuf</i> for the new subrecord. It is this parameter and not the <i>sbuf</i> .LENGTH attribute that defines how many bytes are put in the subrecord.
<i>sbuf</i> :	String variable of <i>nbytes</i> characters (input/output) This buffer specifies the new subrecord data. The maximum buffer size is 12.000 characters.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

Database Format

ReadEnt

Type

Intrinsic Procedure

Database Access

Purpose

Returns MIB information for an entity.

Syntax

ReadEnt (*mib*, *mdata*(1))

Parameters

- mib*: Integer4 expression (input)
Specifies the entity number to query.
- mdata*: Integer array of 8 elements (input/output)
This array returns entity index data:
 - mdata*(1) entity type; if less than 0 this is an erased entity.
 - mdata*(2) low word of PDF pointer.
 - mdata*(3) high word of PDF pointer.
 - mdata*(4) layer entity is on.
 - mdata*(5) view of visibility.
 - mdata*(6) group number.
 - mdata*(7) font number
 - mdata*(8) color number.

Database Format

RSubrecAC

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an AC type subrecord. The AC subrecord holds data about an arc entity.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecAC(*mib, occur, error, transform(1), radius, abeg, aend*)

Parameters

- mib:* Integer4 expression (input)
Specifies the MIB number of the entity that contains the AC subrecord. A string consists of one or more line segments.
- occur:* Integer expression (input)
Specifies which occurrence of the AC subrecord to modify. If there is one or more AC subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next AC subrecord. An error three will be returned if there are no more AC subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.
- error:* Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

- transform:* Real array of 12 elements (input/output) Returns the arc's view transform. This transformation matrix defines the plane which the arc lies in. Usually, this is the view the arc was created in. Elements 10, 11, and 12 are the origin of the arc.
- radius:* Real variable (input/output) This returns the radius of the arc in database units.
- abeg:* Real variable (input/output)
Returns the beginning angle of the arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.
- aend:* Real variable (input/output)
This returns the ending angle of the arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.

Database Format

RSubrecAP

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an AP type subrecord. The AP subrecord holds data about partial arcs used in angular dimensions.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecAP(*mib*, *occur*, *error*, *org*, *radius1*, *abeg1*, *aend1*,
radius2, *abeg2*, *aend2*)

Parameters

- mib*: Integer4 expression (input) Specifies the MIB number of the entity that contains the AP subrecord.
- occur*: Integer expression (input) Specifies which occurrence of the AP subrecord to modify. If there is one or more AP subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* Parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next AP subrecord. An error three will be returned if there are no more AP subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each subrecord occurrence.
- error*: Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

<i>org:</i>	Coordinate variable (input/output) This returns the origin of the arc(s) in model coordinates.
<i>radius1:</i>	Real variable (input/output) This returns the radius of the first arc.
<i>abeg1:</i>	Real variable (input/output) This returns the beginning angle of the first arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.
<i>aend1:</i>	Real variable (input/output) This returns the ending angle of the first arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.
<i>radius2:</i>	Real variable (input/output) Returns the radius of the second arc in database units. The default is inches.
<i>abeg2:</i>	Real variable (input/output) Returns the beginning angle of the second arc.
<i>aend2:</i>	Real variable (input/output) Returns the ending angle of the second arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.

Database Format

RSubrecDS

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of a DS type subrecord. The DS subrecord holds data about the display image that was saved in the Personal Designer SAVE IMAGE command.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecDS(*mib, occur, error, extents(1), scrscl, viewno, dispno*)

Parameters

- mib:* Integer4 expression (input)
Specifies the MIB number of the entity that contains the DS subrecord.
- occur:* Integer expression (input)
Specifies which occurrence of the DS subrecord to modify. If there is one or more DS subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next DS subrecord. An error three will be returned if there are no more DS subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

- error:* Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- extents:* Real array of 4 elements (input/output)
Returns the maximum and minimum values used in a display image. They are given as X-minimum and maximum followed by Y-minimum and maximum.
- scrscl:* Real variable (input/output)
Returns the display screen scale.
- viewno:* Integer variable (input/output)
Returns the view number associated with the display image.
- dispno:* Integer variable (input/output)
Returns the display image number.

Database Format

RSubrecEX

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an EX type subrecord. The EX subrecord holds data about the extents of a part or figure in 3D space.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecEX(*mib*, *occur*, *error*, *extents* (1))

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the EX subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the EX subrecord to modify. If there is one or more EX subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, and error code of three is returned in the *error* parameter.
Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one subrecord. An error three will be returned if there are no more EX subrecords. using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.
- error*: Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.

Database Format

- extents:* Real array of 24 elements (input/output)
The *extents* parameter returns the range of X, Y, and Z values used in a part or figure. This parameter can be used in two ways.
1. The first holds the eight X, Y, and Z values that define the corners of an imaginary cube which surrounds all part or figure geometry. This method is used primarily by Personal Designer.
 2. The second way, used by surfacing in Personal Designer, holds the six minimum and maximum values used by any entity in the database. These are specified in the following order: minimum x, maximum X, minimum Z, maximum Z. The same imaginary cube can be created by generating planes normal to XYZ areas and passing through the six points.

Database Format

RSubrecIL

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an IL type subrecord. The IL subrecord holds data about a figure image list to be inserted in the current part.

WARNING: This procedure is for advanced users only.

Syntax

RSubreeIL(*mib*, *occur*, *error*, *figname*, *mib*, *entcount*,
figdate, *figtime*)

Parameters

- mib*: Integer4 expression (input) Specifies the MIB number of the entity that contains the IL subrecord.
- occur*: Integer expression (input) Specifies which occurrence of the IL subrecord to modify. If there is one or more IL subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next IL subrecord. An error three will be returned if there are no more IL subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

<i>error:</i>	Integer variable (input/output) This parameter returns the error condition: 0 no errors were found. 1 an IO error was found. 3 the subrecord was not found. 4 an invalid MIB number was given.
<i>figname:</i>	String variable of 64 characters (input/output) Returns the figure file name including the full path name.
<i>mib:</i>	Integer4 variable (input/output) Returns the MIB number of the first entity in the figure image list.
<i>entcount:</i>	Integer variable (input/output) Returns the number of entities in the figure starting with the <i>mib</i> parameter.
<i>figdate:</i>	Integer variable (input/output) Returns the system file date of the figure part file.
<i>figtime:</i>	Integer variable (input/output) Returns the system file time of the figure part file.

Database Format

RsubrecL1

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an L 1 type subrecord. The L 1 subrecord holds data about dimension extension line one in a dimension entity.

WARNING: This procedure is for advanced users only.

Syntax

RsubrecL1(*mib, occur, error, pnt1, pnt2*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that contains the L1 subrecord.

occur: Integer expression (input) Specifies which occurrence of the L1 subrecord to modify. If there is one or more L1 subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next L1 subrecord. An error three will be returned if there are no more L1 subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

Database Format

- pnt1*: Coordinate variable (input/output) Returns the first endpoint of the extension line in model coordinates.
- pnt2*: Coordinate variable (input/output)
Returns the second endpoint of the extension line in model coordinates.

Database Format

RSubrecL2

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an L2 type subrecord. The L2 subrecord holds data about dimension extension line two in a dimension entity.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecL2(*mib*, *occur*, *error pnt1*, *pnt2*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the L2 subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the L2 subrecord to modify. If there is one or more L2 subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next L2 subrecord. An error three will be returned if there are no more L2 subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

pnt1: Coordinate variable (input/output)
Returns the first endpoint of the extension line in model coordinates.

pnt2: Coordinate variable (input/output)
Returns the second endpoint of the extension line in model coordinates.

Database Format

RSubrecPA

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of a PA type subrecord. The PA subrecord holds data about entity properties on an entity.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecPA(*mib*, *occur*, *error*, *pname*, *ptype*, *pval*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the PA subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the PA subrecord to modify. If there is one or more PA subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next PA subrecord. An error three will be returned if there are no more PA subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

- error:* Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- pname:* String variable of 8 characters (input/output)
Returns the property name.
- ptype:* String variable of 7 characters (input/output)
Returns the property type.
- pval:* String variable of 100 characters (input/output)
Returns the property value.

Database Format

RSubrecPX

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of a PX type subrecord. The PX subrecord holds data about a point.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecPX(*mib*, *occur*, *error*, *pnt*)

Parameters

- mib*: Integer4 expression (input) Specifies the MIB number of the entity that contains the PX subrecord.
- occur*: Integer expression (input) Specifies which occurrence of the PX subrecord to modify. If there is one or more PX subrecord for a given entity, use this parameter to specify the subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter. Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next PX subrecord. An error three will be returned if there are no more PX subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.
- error*: Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- pnt*: Coordinate variable (input/output) Returns the X, Y, Z locations of the point in model coordinates.

Database Format

RSubrecTD

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of a TD type subrecord. The TD subrecord holds data about text format and orientation used in text dimension labels and MView entities.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecTD(*mib*, *occur*, *error*, *transform*(1), *height*,
width, *linesp*, *just*(1))

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the TD subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the TD subrecord to modify. If there is one or more TD subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next TD subrecord. An error three will be returned if there are no more TD subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

transform: Real array of 12 elements (input/output)
Returns the text's view transform. This transformation matrix defines the plane the text lies in. Usually, this is the view the arc was created in. Elements 10, 11, and 12 are the origin of the arc. Only the first six elements of transform (X and Y), are used for the view transform.

height: Real variable (input/output)
Returns the text height.

width: Real variable (input/output)
Returns the text width.

linesp: Real variable (input/output)
Returns the line spacing, in database units.

just: Integer array of 6 elements (input/output)
This array returns text justification.
Values for *just(1)* are:

- 1 = left justification.
- 2 = right justification.
- 3 = center justification.

just(2 - 6) are reserved; do not use.

Database Format

RSubrecTF

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of a TF type subrecord. The TF subrecord holds data about a figure instance which is inserted into a part.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecTF(*mib, occur, error, transform(1), figuremib*)

Parameters

- mib:** Integer4 expression (input)
Specifies the MIB number of the entity that contains the TF subrecord.
- occur:** Integer expression (input)
Specifies which occurrence of the TF subrecord to modify. If there is one or more TF subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the error parameter.
Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the occur parameter, will automatically access the next TF subrecord. An error three will be returned if there are no more TF subrecords. Using this programming tip is faster than incrementing the occur parameter for each occurrence of the subrecord.

Database Format

- error:* Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- transform:* Real array of 15 elements (input/output)
The first nine elements of the transformation matrix return the figure viewing matrix. Elements 10, 11, and 12 are the figure origin. Elements 13, 14, and 15 return the figure's X, Y, and Z scale factors.
- figuremib:* Integer4 variable (input/output)
Returns the MIB number of the figure definition entity.

Database Format

RSubrecTX

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of a TX type subrecord. The TX subrecord holds the text of a text string used in a text, dimension, MView, curve, or surface entity.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecTX(*mib, occur, error, textstring*)

Parameters

- mib:* Integer4 expression (input) Specifies the MIB number of the entity that contains the TX subrecord.
- occur:* Integer expression (input) Specifies which occurrence of the TX subrecord to modify. If there is one or more TX subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next TX subrecord. An error three will be returned if there are no more TX subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.
- error:* Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- textstring:* String expression (input/output) Returns the text string.

Database Format

RSubrecVN

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of a VN type subrecord. The VN subrecord holds data about a defined view. Note that there is no entity records and therefore no VN subrecords for the six predefined views.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecVN(*mib, occur, error, transform(1)*)

Parameters

- mib*: Integer4 expression (input) Specifies the MIB number of the entity that contains the VN subrecord.
- occur*: Integer expression (input) Specifies which occurrence of the VN subrecord to modify when more than one is present for a given entity. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned. Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, automatically access the next VN subrecord. An error three is returned if there are no more VN subrecords. Using this tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.
- error*: Integer variable (input/output) Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- transform*: Real array of 15 elements (input/output) Returns the view transformation matrix. Only the first nine elements of *transform* are used. The offset and scaling factors are not modified.

Database Format

RSubrecXH

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an XH type subrecord. The XH subrecord holds data about cross-hatch lines used in cross-hatch entities.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecXH(*mib*, *occur*, *error*, *endpoints*, *npts*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the XH subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the XH subrecord to rmodify. If there is one or more XH subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next XH subrecord. An error three will be returned if there are no more XH subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

endpoints: Coordinate array of *nvert* elements (input/output)
Returns the endpoints of the cross-hatch lines. This parameter contains pairs of endpoints rather than the cross-hatch boundaries.

npnts: Integer expression (input/output)
This returns the number of endpoints in the *endpoints* parameter. The maximum number of lines returned is 500; and therefore 1.000 *endpoints* for each cross-hatch area.

Database Format

RSubrecXN

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an XN type subrecord. The XN subrecord holds the vertices of a string used in a string, label, arrow, dimension, cross-hatch, MView, curve, or surface entity. A string consists of two or more line segments.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecXN(*mib*, *occur*, *error*, *vertices(1)*, *nvert*)

Parameters

mib: Integer4 expression (input)

Specifies the MIB number of the entity that contains the XN subrecord.

occur: Integer expression (input)

Speeifies which occurrence of the XN subrecord to modify. If there is one or more XN subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.

Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative one for the *occur* parameter, will automatically access the next XN subrecord. An error three will be returned if there are no more XN subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 3 the subrecord was not found.
- 4 an invalid MIB number was given.

vertices: Coordinate array of *nvert* elements (input/output)
Returns string vertices.

nvert: Integer variable (input/output)
This returns the number of vertices in the *vertices* parameter.

Database Format

RSubrecXZ

Type

Intrinsic Procedure

Database Access

Purpose

Reads the contents of an XZ type subrecord in the PDF portion of the part database. The XZ subrecord holds data about the endpoints of a line.

WARNING: This procedure is for advanced users only.

Syntax

RSubrecXZ(*mib*, *occur*, *error*, *pnt1*, *pnt2*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that contains the XZ subrecord.
- occur*: Integer expression (input)
Specifies which occurrence of the XZ subrecord to modify. If there is one or more XZ subrecord for a given entity, use this parameter to specify the particular subrecord. The occurrences start at one and increase with each additional occurrence. If a specified occurrence of the subrecord does not exist, an error code of three is returned in the *error* parameter.
- Programming Tip: To call this procedure, give the MIB number for the entity and the first occurrence of the subrecord you want to access. Subsequent calls with the same MIB number, and a negative 1 for the *occur* parameter, will automatically access the next XZ subrecord. An error three will be returned if there are no more XZ subrecords. Using this programming tip is faster than incrementing the *occur* parameter for each occurrence of the subrecord.

Database Format

- error:* Integer variable (input/output)
Returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 3 the subrecord was not found.
 - 4 an invalid MIB number was given.
- pnt1:* Coordinate variable (input/output)
Returns the first endpoint of the line.
- pnt2:* Coordinate variable (input/output)
Returns the second endpoint of the line.

Database Format

WriteEnt

Type

Intrinsic Procedure

Database Access

Purpose

Modifies entity index (or MIB) information. Note that the PDF pointer cannot be modified. To add a new entity index, use **AddEnt**. See **ReadEnt** for more information.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WriteEnt(*mib*, *mdata*(1))

Parameters

mib: Integer4 expression (input)
Specifies the entity MIB number of the entity to modify.

mdata: Integer array of 8 elements (input/output)
Specifies entity index data:

- mdata*(1) entity type; if less than 0, this will erase the entity.
- mdata*(2) low word of PDF pointer; this value is not modified.
- mdata*(3) high word of PDF pointer; this value is not modified.
- mdata*(4) layer entity is on.
- mdata*(5) view of visibility.
- mdata*(6) group number.
- mdata*(7) font number.
- mdata*(8) color number.

Database Format

WSubrecAC

Type

Intrinsic Procedure

Database Access

Purpose

Writes an AC type subrecord to the PDF portion of the database. The AC subrecord holds data about an arc entity. Write an AC subrecord when you are adding a new entity to the database with the intrinsic procedure

AddEnt.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecAC(*mib*, *error*, *transform(1)*, *radius*, *abeg*, *aend*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that will contain the AC subrecord.

error: Integer variable (input/output)
This parameter returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

transform: Real array of 12 elements (input/output) Specifies the arc's view transform. This transformation matrix defines the plane which the arc lies in. Usually, this is the view the arc was created in. Elements 10, 11, and 12 are the origin of the arc.

radius: Real expression (input)
Specifies the radius of the arc in database units.

abeg: Real expression (input)
Specifies the beginning angle of the arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.

aend: Real expression (input) Specifies the ending angle of the arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.

Database Format

WSubrecAP

Type

Intrinsic Procedure

Database Access

Purpose

Writes an AP type subrecord to the PDF portion of the database. The AP subrecord holds data about partial arcs used in angular dimensions. Write an AP subrecord when you are adding a new angular dimension entity to the database with the intrinsic procedure *AddEnt*.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database

Syntax

WSubrecAP(*mib*, *error*, *org*, *radius1*, *abeg1*, *aend1*, *radius2*,
abeg2, *aend2*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that will contain the AP subrecord.
- error*: Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 4 an invalid MIB number was given.
- org*: Coordinate expression (input)
Specifies the origin of the arc(s) in model coordinates.
- radius1*: Real expression (input)
Specifies the radius of the first arc in database units.
- abeg1*: Real expression (input)
Specifies the beginning angle of the first arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.

Database Format

- aend1*: Real expression (input)
Specifies the ending angle of the first arc in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.
- radius2*: Real expression (input)
Specifies the radius of the second arc in database units.
- abeg2*: Real expression (input)
Specifies the beginning angle of the second are in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.
- aend2*: Real expression (input)
Specifies the ending angle of the second are in radians. Angle zero starts at the X-axis and increases counterclockwise. The X-axis is defined by the arc's view transform.

Database Format

WSubrecDS

Type

Intrinsic Procedure

Database Access

Purpose

Writes a DS type subrecord to the PDF portion of the database. The DS subrecord holds data about the display image that was saved in the Personal Designer command SAVE IMAGE. Write a DS subrecord when you are adding a new display image entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Ineorrect use could damage or destroy your part database.

Syntax

WSubrecDS(*mib*, *error*, *extents*(1), *scrscl*, *viewno*, *dispno*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that will contain the DS subrecord.
- error*: Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 4 an invalid MIB number was given.
- extents*: Real array of 4 elements (input/output)
Specifies the maximum and minimum values used in a display image. They are given as X minimum and maximum followed by Y minimum and maximum.
- scrscl*: Real expression (input)
Specifies the display image's screen scale.
- viewno*: Integer expression (input)
Specifies the view number associated with the display image.
- dispno*: Integer expression (input)
Specifies the display image number.

Database Format

WSubrecEX

Type

Intrinsic Procedure

Database Access

Purpose

Writes an EX type subrecord to the PDF portion of the database. The EX subrecord holds data about the extents of a part or figure in 3D space. Write an EX subrecord when you are adding new entities to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecEX(*mib*, *error*, *extents*(1))

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that will contain the EX subrecord.
- error*: Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 4 an invalid MIB number was given.
- extents*: Real array of 24 elements (input/output)
Specifies the range of X, Y, and Z values used in a drawing or figure. This parameter can be used in two ways.
- 1 . The first holds the eight X, Y, and Z values that define the corners of an imaginary cube which surrounds all part or figure geometry. This method is used primarily by Personal Designer.
 2. The second way, used with surfacing, specifies the six minimum and maximum values used by any entity in the database. These are specified in the following order: minimum X, maximum X, minimum Y, maximum Y, minimum Z, maximum Z. The same imaginary cube can be created by generating planes normal to X, Y, and Z axes and passing through the six points.

Database Format

WSubrecIL

Type

Intrinsic Procedure

Database Access

Purpose

Writes an IL type subrecord to the PDF portion of the database. The IL subrecord holds data about a figure image that is to be inserted in the current part. Write an IL subrecord when you are adding a new figure image entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecIL(*mib, error, figname, figmib, entcount, figdate, figtime*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that will contain the IL subrecord.
- error*: Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 4 an invalid MIB number was given.
- figname*: String expression of 64 characters (input)
Specifies the figure file name including the full path name.
- figmib*: Integer expression (input)
Specifies the MIB number of the first entity in the figure image list.
- entcount*: Integer expression (input)
Specifies the number of entities in the figure starting with the *mib* parameter.

Database Format

- figdate:* Integer expression (input)
Specifies the system file date of the figure part file. This is the date the file was last modified. See Appendix E, Internal Data Storage Format, for more information.
- figtime:* Integer expression (input)
Specifies the system file time of the figure part file. This is the time the file was last modified. See Appendix E, Internal Data Storage Format, for more information.

Database Format

WSubrecL1

Type

Intrinsic Procedure

Database Access

Purpose

Writes an L1 type subrecord to the PDF portion of the database. The L 1 subrecord holds data about dimension extension line one. Write an L1 subrecord when you are adding a new dimension entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WsubrecL1(*mib, error, pnt1, pnt2*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that will contain the L1 subrecord.
- error*: Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 4 an invalid MIB number was given.
- pnt1*: Coordinate expression (input)
Specifies the first endpoint of extension line one.
- pnt2*: Coordinate expression (input)
Specifies the second endpoint of extension line one.

Database Format

WSubrecL2

Type

Intrinsic Procedure

Database Access

Purpose

Writes an L2 type subrecord to the PDF portion of the database. The L2 subrecord holds data about dimension extension line two. Write an L2 subrecord when you are adding a new dimension entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecL2(*mib*, *error*, *pnt1*, *pnt2*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that will contain the L2 subrecord.

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

pnt1: Coordinate expression (input)
Specifies the first endpoint of extension line two.

pnt2: Coordinate expression (input)
Specifies the second endpoint of extension line two.

Database Format

WSubrecPA

Type

Intrinsic Procedure

Database Access

Purpose

Writes a PA type subrecord to the PDF portion of the database. The PA subrecord holds data about entity properties. Write a PA subrecord when you are adding a new entity to the database with the intrinsic procedure **AddEnt**. You can also use this procedure to add property subrecords to existing entities.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecPA(*mib*, *error*, *pname*, *ptype*, *pval*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that will contain the PA subrecord.

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

pname: String expression of 8 characters (input)
Specifies the property name.

ptype: String expression of 7 characters (input)
Specifies the property type.

pval: String expression of 100 characters (input)
Specifies the property value.

Database Format

WSubrecPX

Type

Intrinsic Procedure

Database Access

Purpose

Writes a PX type subrecord to the PDF portion of the database. The PX subrecord holds data about the point. Write a PX subrecord when you are adding a new point entity to the database with the intrinsic procedure

AddEnt.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecPX(*mib*, *error*, *pnt*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that will contain the PX subrecord.

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

pnt: Coordinate expression (input) Specifies the X, Y, and Z locations of the point in model coordinates.

Database Format

WSubrecTD

Type

Intrinsic Procedure

Database Access

Purpose

Writes a TD type subrecord to the PDF portion of the database. The TD subrecord holds data about text format and orientation. Note that it does not hold the actual text itself, which is usually in a TX subrecord. Write a TD subrecord when you are adding a new text, dimension, label, or auxiliary entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecTD(*mib, error, transform(1), height, width, linesp, just(1)*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that will contain the TD subrecord.

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

transform: Real array of 15 elements (input/output)
Specifies the text's view transform. This transformation matrix defines the plane which the text lies in. Usually, this is the view the arc was created in. In TD subrecords, only elements 1 through 6 and 10 through 12 of the view transform are used. Elements 1 through 6 are the X and Y axes cosine. Elements 10 through 12 are the origin of the text.

height: Real expression (input) Specifies text height in database units.

width: Real expression (input)
Specifies the text width in database units.

linesp: Real expression (input)
Specifies line spacing, in database units.

Database Format

just: Integer array of 6 elements (input/output) Specifies text justification. Values for *just(1)* are:

- 1 = left justification.
- 2 = right justification.
- 3 = center justification,

just(2) is an associativity flag for dimension text. Values are:

- 1 has D4 subrecord.
- 0 has no associativity

just(3) is MView number

just(4) is text font number

just(5) is text slant angle

just(6) is reserved; do not use.

Database Format

WSubrecTF

Type

Intrinsic Procedure

Database Access

Purpose

Writes a TF type subrecord to the PDF portion of the database. The TF subrecord holds data about a figure which is inserted into a part. Write a TF subrecord when you are adding a new figure instance entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecTF(*mib*, *error*, *transform(1)*, *figuremib*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that will contain the TF subrecord.

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

figuremib: Integer4 expression (input)
Specifies the MIB number of the figure image list entity.

transform: Real array of 15 elements (input/output). The first nine elements of the transformation matrix specify the figure viewing matrix. Elements 10, 11, and 12 are the figure origin. Elements 13, 14, and 15 are the figure's X, Y, and Z scale factors.

Database Format

WSubrecTX

Type

Intrinsic Procedure

Database Access

Purpose

Writes a TX type subrecord to the PDF portion of the database. The TX subrecord holds the actual text of a text string entity. Write a TX subrecord when you are adding a new text, dimension, MView, curve, or surface entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecTX(*mib*, *error*, *textstring*)

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that will contain the TX subrecord.

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

textstring: String expression (input) Specifies the text string to be modified.

Database Format

WSubrecVN

Type

Intrinsic Procedure

Database Access

Purpose

Writes a VN type subrecord to the PDF portion of the database. The VN subrecord holds data about a defined view. Write a VN subrecord when you are adding a new view entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecVN(*mib*, *error transform*(1))

Parameters

mib: Integer4 expression (input) Specifies the MIB number of the entity that will contain the VN subrecord.

error: Integer variable (input/output) Returns the error condition:

- 0 no errors were found.
- 1 an IO error was found.
- 4 an invalid MIB number was given.

transform: Real array of 15 elements (input/output)
Specifies the view transformation matrix. Only the first nine elements of transform are used. The offset and scaling factors are not written.

Database Format

WSubrecXH

Type

Intrinsic Procedure

Database Access

Purpose

Writes an XH type subrecord to the PDF portion of the database. The XH subrecord holds data about cross-hatch lines. Write an XH subrecord when you are adding a new cross-hatch entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecXH(*mib*, *error*, *endpoints*, *npnts*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that will contain the XH subrecord.
- error*: Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 4 an invalid MIB number was given.
- endpoints*: Coordinate array of *npnts* elements (input/output)
Specifies the endpoints of the cross-hatch lines. The contents of this parameter is pairs of endpoints rather than the cross-hatch boundaries.

Database Format

WSubrecXN

Type

Intrinsic Procedure

Database Access

Purpose

Writes an XN type subrecord to the PDF portion of the database. The XN subrecord holds the vertices of a string entity. A string consists of two or more line segments. Write an XN subrecord when you are adding a new string, label, arrow, dimension, cross-hatch, MView, curve or surface entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecXN(*mib*, *error*, *vertices*(1), *nvert*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that will contain the XN subrecord.
- error*: Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 4 an invalid MIB number was given.
- vertices*: Coordinate array of *nvert* elements (input/output)
Specifies string vertiees. The order of the vertices in the array is the order in which the string is created.
- nvert*: Integer expression (input)
Specifies the number of vertices in the *vertices* parameter.

Database Format

WSubrecXZ

Type

Intrinsic Procedure

Database Access

Purpose

Writes an XZ type subrecord to the PDF portion of the database. The XZ subrecord holds data about the endpoints of a line. Write an XZ subrecord when you are adding a new line entity to the database with the intrinsic procedure **AddEnt**.

WARNING: This procedure is for advanced users only. Incorrect use could damage or destroy your part database.

Syntax

WSubrecXZ(*mib, error, pnt1, pnt2*)

Parameters

- mib*: Integer4 expression (input)
Specifies the MIB number of the entity that will contain the XZ subrecord.
- error*: Integer variable (input/output)
This parameter returns the error condition:
- 0 no errors were found.
 - 1 an IO error was found.
 - 4 an invalid MIB number was given.
- pnt1*: Coordinate expression (input)
Specifies the first endpoint of the line.
- pnt2*: Coordinate expression (input)
This parameter specifies the second endpoint of the line.

Database Format

Examples for Direct Database Access

Example: CIRCLE.UPL

```
-----
-- CIRCLE.UPL -- create a circle.
-- This program demonstrates direct database
-- access using UPL intrinsics. Note that a
-- circle contains the AC subrecord which has
-- it own UPL intrinsics.
-----
Proc Main

    Const      Integer ArcType = 3
    Integer    ColorC, LayerC, VvisC, FontC, GroupC
    Integer4    Mib
    Integer     DataBuffer(5)

    Integer     Occur, Error, CPL
    Real        Transform(15), RadiusC, ABeg, AEnd

    Coord      Origin
    Integer     NDigs

-- start of code --

-- First, create the MIB sub-record. Use system
-- default values for layer, color, etc.
-- The call to AddEnt actually adds the subrecord.

    SysVarI(1, ColorC)
    SysVarI(2, FontC)
    SysVarI(3, LayerC)
    SysVarI(1474, VvisC)
    If VvisC <> 0 then
        SysVarI(4, VvisC)
    Endif
    SysVarI(1333, GroupC)
    GroupC = GroupC - 1
```

Database Format

```
DataBuffer(1)      = LayerC
DataBuffer(2)      = VvisC
DataBuffer(3)      = GroupC
DataBuffer(4)      = FontC
DataBuffer(5)      = ColorC

AddEnt( ArcType, DataBuffer(1), Mib, Error)

-- Now create the PDF subrecord and add it to
-- the database. Prompt user for radius and
-- origin. Use current CPL. Note that origin,
-- orientation of circle, and scaling factors (not
-- used here) are all stored in Transform array.
-- WSubrecAC actually adds PDF subrecord.

Accept RadiusC Prompt('Enter radius: ') NewLine

SysVarI( 12, CPL)
GetCPL( CPL, Transform(1))

Print 'Dig center of circle ',
GetDig( 1, 1, Ndigs, Origin)
Transform(10) = Origin.X
Transform(11) = Origin.Y
Transform(12) = Origin.Z

ABeg = 0
AEnd = TwoPio

WSubrecAC( Mib, Error, Transform(1), RadiusC, \
          ABeg, AEnd)

RpntEnt( Mib, 1, Error)

End Proc
```

Database Format

Example: CHCIRCLE.UPL

```
-----
-- CHCIRCLE.UPL -- change a circle.
-- This program demonstrates direct database
-- access using UPL intrinsics. Note that a
-- circle contains the AC subrecord which has
-- it own UPL intrinsics.
-----

Proc Main

    Const Integer ArcType = 3
    Integer ColorC, LayerC, VvisC, FontC, GroupC
    Integer4 Mib(1)
    Integer DataBuffer(8)

    Integer Occur, Error, CPL
    Real Transform(15), RadiusC, ABeg, AEnd

    Coord Origin
    Integer NDigs, Iend

    Integer4 NEnts

-- start of code --

-- First, identify the circle using GetEnt. Only
-- allow circlces to be digged.

    EntMask(0)
    EntMask(ArcType)
    Print 'Digitize circle: ',
    GetEnt( 1, NEnts, Mib(1), IEnd)
    Print

-- Next, read the MIB sub-record using ReadEnt,
-- modify the data, and then modify the MIB
-- subrecord using WriteEnt.
-- Note that DataBuffer has different offsets
-- from the one used with AddEnt. Also, you could
-- access data directly in DataBuffer and not use
-- ColorC and LayerC.
```

Database Format

```
ReadEnt( Mib(1), DataBuffer(1))

LayerC = DataBuffer(4)
ColorC = DataBuffer(8)
Print 'Circle color is ',ColorC, '.'
Print 'Circle layer is ',LayerC, '.'
Accept ColorC Prompt('New color: ') NewLine
Accept LayerC Prompt('New layer: ') NewLine
DataBuffer(4) = LayerC
DataBuffer(8) = ColorC

WriteEnt(Mib(1), DataBuffer(1))

-- Now read the PDF subrecord using RSubrecAC,
-- modify the data, and then modify the
-- subrecord with MSubrecAC.

Occur = 1 RSubrecAC( Mib(1), Occur, Error,
Transform(1), RadiusC, ABeg, AEnd)

Print 'Current radius is ',RadiusC, '.'
Accept RadiusC Prompt('New radius: ') NewLine

Print 'Dig new center of circle: ',
GetDig( 1, 1, Ndigs, Origin)
Transform(10) = Origin.X
Transform(11) = Origin.Y
Transform(12) = Origin.Z

MSubrecAC( Mib(1), Occur, Error, \
          Transform(1), RadiusC, ABeg, AEnd)

RpntEnt( Mib(1), 1, Error)

End Proc
```

Database Format

Example: ELLIP.UPL

```
-----
-- ELLIP.UPL -- create an ellipse.
-- This program demonstrates direct database
-- access using UPL intrinsics. Note that an
-- ellipse contains the EP subrecord which
-- DOES NOT it own UPL intrinsics. Instead,
-- use the code provided in this appendix
-- listed as 'EP subrecord definition'.
-- In this example it is read (via 'include'
-- statement) from a file named 'epsubrec.inc'.
-----

Proc Main

    $Include 'epsubrec.inc'

    Const Integer EllipseType = 14
    Integer ColorC, LayerC, VvisC, FontC, GroupC
    Integer4 Mib
    Integer DataBuffer(5)

    Integer Occur, Error, CPL

    Coord Origin
    Integer NDigs

-- start of code --

-- First, create the MIB sub-record. Use system
-- default values for layer, color, etc.
-- The call to AddEnt actually adds the subrecord.

    SysVarI(1, ColorC)
    SysVarI(2, FontC)
    SysVarI(3, LayerC)
    SysVarI(1474, VvisC)
    If VvisC <> 0 then
        SysVarI(4, VvisC)
    Endif
    SysVarI(1333, GroupC)
    GroupC = GroupC - 1
```

Database Format

```
DataBuffer(1) = LayerC
DataBuffer(2) = VvisC
DataBuffer(3) = GroupC
DataBuffer(4) = FontC
DataBuffer(5) = ColorC

AddEnt( EllipseType, DataBuffer(1), Mib, Error)

-- Now create the PDF subrecord and add it to
-- the database. Prompt user for radius and
-- origin. Use current CPL. Note that origin,
-- orientation of ellipse, and scaling factors
-- (not used here) are all stored in Transform
-- array. PutSrR actually adds PDF subrecord.

Accept MajorRadEP Prompt('Enter major axis: ')\
                               NewLine
Accept MinorRadEP Prompt('Enter minor axis: ')\
                               NewLine

SysVarI( 12, CPL)
GetCPL( CPL, EPsubrec(1))

Print 'Dig center of ellipse: ',
GetDig( 1, 1,      Ndigs, Origin)
TranslateEP      Origin

StartAngEP  0
EndAngEP = TwoPio

PutSrR(Mib, 'EP', 64, EPsubrec(1), Error)

RpntEnt( Mib, 1, Error)

End Proc
```


Database Format

Example: CHELLIP.UPL

```
-----  
-- CHELLIP.UPL -- change an ellipse.  
-- This program demonstrates direct database  
-- access using UPL intrinsics. Note that an  
-- ellipse contains the EP subrecord which  
-- DOES NOT it own UPL intrinsics. Instead,  
-- use the code provided in this appendix  
-- listed as 'EP subrecord definition'.  
-- In this example it is read (via 'include'  
-- statement) from a file named 'epsubrec.inc'.  
-----
```

Proc Main

```
    $Include 'epsubrec.inc'  
  
    Const Integer EllipseType = 14  
    Integer ColorC, LayerC, VvisC, FontC, GroupC  
    Integer4 Mib(1)  
    Integer DataBuffer(8)  
  
    Integer Occur, Error, CPL  
  
    Coord Origin  
    Integer NDigs, Iend, NBytes  
  
    Integer4 NEnts  
  
    String EPrec:2    'EP'  
  
-- start of code --  
  
-- First, identify the ellipse using GetEnt.  
-- Only allow ellipses to be digged.  
  
    EntMask(0)  
    EntMask(EllipseType)  
    Print 'Digitize ellipse: ',  
    GetEnt( 1, NEnts, Mib(1), IEnd)  
    Print
```

Database Format

```
-- Next, read the MIB sub-record using ReadEnt,
-- modify the data, and then modify the MIB
-- subrecord using WriteEnt.Note that DataBuffer
-- has different offsets from the one used with
-- AddEnt. Also, you could access data directly in
-- DataBuffer and not use ColorC and LayerC.

ReadEnt( Mib(1), DataBuffer(1))

LayerC = DataBuffer(4)
ColorC = DataBuffer(8)
Print 'Circle color is ', ColorC , '.'
Print 'Circle layer is ', LayerC , '.'
Accept ColorC Prompt('New color: ') NewLine
Accept LayerC Prompt('New layer: ') NewLine
DataBuffer(4) = LayerC
DataBuffer(8) = ColorC
WriteEnt(Mib(1), DataBuffer(1))

-- Now read the PDF subrecord using GetSrR, modify
-- the data,then modify the subrecord with ModSrR.

Occur = 1
GetSrR( Mib(1), EPrec, Occur, 64, NBytes, \
      ESubrec(1), Error)
Print 'Current major axis is ',\
      MajorRadEP, '.'
Accept MajorRadEP Prompt('New major axis:')\
      NewLine
Print 'Current minor axis is ',\
      MinorRadEP, '.'
Accept MinorRadEP Prompt('New minor axis:')\
      NewLine
Print 'Dig new center of ellipse: ',
GetDig( 1, 1, Ndigs, Origin)
TranslateEP = Origin
ModSrR( Mib(1), 'EP', Occur, 64, ESubrec(1), \
      Error)
RpntEnt( Mib(1), 1, Error)

End Proc
```