DipTrace Pattern Editor Library — XML Format Specification

Generated from the DipTrace serializer, repository revision 7276 — reflects current program behaviour.

Contents

How to read this specification 1
Document structure 1
Value types & formatting 2
Coordinate frames, units & angles 2
Identifiers, references & list ordering 2
Object state flags 3
Plug-in exchange & import merge 3
<Library> — pattern-library root 4
<PadStyle> — reusable pad stack 4
<Categories> / <Category> — library category tree 8
<Pattern> — a footprint 9
<Pad> — a footprint pad 13
<Shape> — silkscreen / assembly graphic or text 14
<Hole> — mounting hole 16
<Dimension> — a dimension object 16
<Model3D> — attached 3D model 18

How to read this specification

This document describes the **DipTrace Pattern Editor Library** flavour of the DipTrace XML format — the same format used both for `File ▸ Save/Open As ▸ DipTrace XML` and for the plug-in exchange file. It is generated directly from the DipTrace serializer and reflects what the program actually reads and writes.

Each element is given with a short description, a representative XML fragment, and a table of its attributes. The **Written** column states the condition under which the serializer emits the attribute (`always`, a guard such as `if >-1`, or a context such as `Text shapes only`); an attribute left at its default is usually omitted, so a missing attribute means *use the default*, not zero.

Document structure

A pattern (footprint) library is a single `<Library>` root holding the pad-style palette and the patterns:

```
<Library Type="DipTrace-PatternLibrary" Name="..." Version="5.x" UID32="..."
Units="mm">
  <PadStyles>...</PadStyles> <Categories>...</Categories>
```

```
  <Patterns> <Pattern Id="0" ...>...</Pattern> ... </Patterns>
</Library>
```

A plug-in's data node is `/Library/Patterns/Pattern`. Pad geometry is defined by a shared `<PadStyles>` palette that individual pads reference by name.

Value types & formatting

Type	Encoding
`Int`	A plain decimal integer.
`Real`	A decimal number expressed in the file's `Units`, written with a **dot** decimal separator. The reader is tolerant (it normalises both dot and comma), but the dot is canonical and is what other tools expect.
`Bool`	`Y` / `N`. (A few Schematic simulator fields use `+` / `-`.)
`Text`	A string attribute or element text.
`Color`	A 24-bit integer in Windows `0x00BBGGRR` order (e.g. `255` = red).
enum	A fixed set of text (or, in a few cases, integer) values, listed with the element.

Coordinate frames, units & angles

- The document root carries `Units="mm" | "inch" | "mil"`. **Every** `Real` coordinate, length and size in the file is in those units; there is no per-object override.
- Objects inside a library symbol or footprint (pins, pads, shapes) are stored in an **object-centre** frame — coordinates are relative to the part/footprint origin and are not affected by where an instance is placed.
- Angles are in **radians, counter-clockwise**, unless an attribute is explicitly a discrete orientation enum (e.g. a footprint `Orientation` of `0/90/180/270`).
- Coordinate scaling and sign are internal to the file; read and write values as they are — do not rescale or flip them.

Identifiers, references & list ordering

- Most list elements carry an `Id` (or `Index`/`Number`) that other objects reference. A reference value of `-1` means *none / not connected*.
- A few lists are referenced **positionally**, not by an `Id` attribute: a `ViaStyle` value indexes the in-order `<ViaStyles>` list and a `NetClass` value indexes the in-order `<NetClasses>` list. Reordering those lists silently reassigns the references.
- When a whole file is opened, top-level object lists are resolved **by position**: each array must be dense, ascending from `Id` 0, with position equal to `Id`. A file authored out of order or with gaps will bind references to the wrong objects.

Object state flags

Attribute	Type	Meaning
`Selected`	Bool	User selection state.
`Locked`	Bool	Edit lock.
`Group`	Int	Group `Id`, or `-1` for none; groups are listed in a `<Groups>` section.
`Enabled`	Bool	*Exists / removed.* A normal save prunes disabled objects, so a saved file rarely shows `Enabled="N"`; on the plug-in exchange / edit-import path, however, `Enabled="N"` is honoured as a **delete** flag on the editable objects. An absent flag means enabled.

Plug-in exchange & import merge

The same format is the **plug-in exchange** file: DipTrace exports the selected data, launches the plug-in with the file path as its argument, waits for it to exit, then re-imports the (possibly edited) file. How the result merges back depends on the import mode:

- **Whole-list (`ImpMode=All`)** — the incoming list *replaces* the project's list of that object kind; anything omitted is removed.
- **`Edit`** — each **top-level** object is matched to an existing one **by its `Id`** and overwritten in place; an object with a new or absent `Id` is added. Keep an object's `Id` to edit it, and **omit the `Id`** on objects you add — never invent one, because over a partial export a computed "free" Id can collide with an unexported object and overwrite it.

> **Nested lists without their own `Id` are REPLACED, not merged.** A net's `<Traces>`, a net's or bus's `<Wires>`, and point lists have no per-member `Id`. When such a container is present in your XML, DipTrace **discards the existing list and rebuilds it from exactly the children you supply** — so appending one trace under an existing net replaces that net's *entire* routing. To add one member, re-list every current member plus the new one; to leave the list untouched, omit the container element entirely.

> **The `Selected` filter skips — it does not add.** When a plug-in runs with a `Selected` filter under `Edit`, only objects flagged `Selected="Y"` are processed at all: a marked object with an existing `Id` is edited, a marked object with a new `Id` is added, and an **unflagged** object is **skipped entirely** (neither edited nor added). Put `Selected="Y"` on every object you add or edit under such a filter, or its data is silently dropped.

To remove an object, set `Enabled="N"` on it and leave it in the file (see *Object state flags*) — do not delete the XML node. Component and Pattern *library* files use coarser Library / Component / Part import modes rather than per-object `Edit`, but the `Enabled` delete flag and the `Id` rules still apply.

A note on completeness. Where a newer serializer field has no counterpart in an older published table (for example the real-valued/mono font fields, or per-object shield-group ids), it is documented here as a first-class attribute. Preserve attributes and elements you do not recognise when editing a file in place — they belong to other subsystems and to future format versions.

<Library> — pattern-library root

The root element of a footprint library (`.lib`). It carries the library identity and the measurement unit, then holds the reusable pad stacks (`<PadStyles>`), the category tree (`<Categories>`) and the footprints (`<Patterns>`). All `Real` coordinates elsewhere in the file are expressed in the unit named here.

```
<?xml version="1.0" encoding="UTF-8"?>
<Library Type="DipTrace-PatternLibrary" Name="BGA 0.35mm" Hint="…"
         Version="5.3.0.0" UID32="1906685151" Units="inch">
  <PadStyles> … </PadStyles>
  <Categories> … </Categories>
  <Patterns> … </Patterns>
</Library>
```

Attribute	Type	Written	Description
`Type`	Text	always	Library kind — the literal `DipTrace-PatternLibrary`.
`Name`	Text	full save only	Library name.
`Hint`	Text	full save only	Library caption / description.
`Version`	Text	full save only	Product version that wrote the file.
`UID32`	Int	full save only	32-bit library UID, regenerated on every save.
`Units`	enum	always	File unit for all Reals: `inch` (emitted for internal `in`), `mil`, or `mm`.

Note. `Name`, `Hint`, `Version` and `UID32` are emitted only for a full library save; when the node is exported as plug-in data (a single pattern or a nested section) they are omitted and only `Type` and `Units` are present.

<PadStyle> — reusable pad stack

A named, shared pad definition inside `<PadStyles>`. Pads reference it by name through `<PadStyle="…">`; there is no numeric id. It holds the through/surface flag, the hole geometry, the copper `<MainStack>`, optional secondary `<Terminals>`, and mask/paste settings.

```
<PadStyle Name="PadT3" Type="Through" HoleType="Round" Hole="0.0354">
  <MainStack Shape="Obround" Width="0.0591" Height="0.0591"/>
</PadStyle>
```

Attribute	Type	Written	Description
`Name`	Text	always	Style name; the value pads bind to via `Style`.
`Type`	enum	always	Pad kind — `Surface` or `Through`.
`HoleType`	enum	Through only	Hole shape — `Round` or `Obround`.
`Hole`	Real	Through / Fiducial	Round-hole diameter (or obround-hole width) for a Through pad; also the keepout diameter of a Fiducial pad (see note).
`HoleH`	Real	Obround hole only	Obround hole height. Present only when `HoleType="Obround"`.

Value	Name
`0`	Surface
`1`	Through

> **Note.** `Hole` serves two roles. For a Through pad it is the drilled-hole size (round diameter or obround width). For a **Fiducial** stack (`<MainStack Shape="Fiducial">`) it is emitted regardless of Surface/Through and means the **keepout diameter**. `HoleH` is strictly the obround hole height and never carries any Fiducial meaning. The `Side` attribute is intentionally absent — the writer does not emit it (the reader still tolerates it in old files).

<MainStack> — copper shape

The primary copper pad shape of the style. `XOff`/`YOff` offset the copper from the hole center and appear only on Through pads. `Corner` is a corner-rounding value emitted only for a Rectangle stack; a Polygon stack instead carries a `<Points>` list.

```
<MainStack Shape="Rectangle" Width="0.8" Height="0.4" XOff="0" YOff="0"
Corner="0"/>
```

Attribute	Type	Written	Description
`Shape`	enum	always	Copper outline — see enum below.
`Width`	Real	always	Copper width. For a Fiducial this doubles as the copper diameter.

Attribute	Type	Written	Description
`Height`	Real	if not Fiducial	Copper height. Not written when `Shape="Fiducial"` (Width alone defines the fiducial).
`XOff`	Real	Through only	Copper X offset from the hole center.
`YOff`	Real	Through only	Copper Y offset from the hole center.
`Corner`	Real	Rectangle only	Corner-rounding value (roundrect); emitted only for a Rectangle shape.

Value	Name
`0`	Ellipse
`1`	Obround
`2`	Rectangle
`3`	Polygon
`4`	D-shape
`20`	Fiducial

> **Note.** A `Polygon` stack emits a child `<Points>` containing center-relative `<Point X Y/>` vertices.

<Terminal> — secondary pad shape

Optional additional shapes (up to four) inside `<Terminals>`, positioned relative to the pad center. `Angle` is in radians CCW (not scaled by the file unit).

```
<Terminals>
  <Terminal Shape="Rectangle" X="0" Y="0" Angle="0" Width="0.8" Height="0.4"
Corner="0"/>
</Terminals>
```

Attribute	Type	Written	Description
`Shape`	enum	always	Terminal outline — see enum below.
`X`	Real	always	X offset from the pad center.
`Y`	Real	always	Y offset from the pad center.
`Angle`	Real	always	Rotation in radians CCW.
`Width`	Real	always	Terminal width.
`Height`	Real	always	Terminal height.

Attribute	Type	Written	Description
`Corner`	Real	Rectangle only	Corner-rounding value; emitted only for a Rectangle terminal.

Value	Name
`0`	Ellipse
`1`	Obround
`2`	Rectangle
`3`	Polygon
`4`	D-shape

> **Note.** Index `0` is `Ellipse`, not a "None" value. A `Polygon` terminal carries its own `<Points>` list of center-relative vertices.

<MaskPaste> — solder mask & paste

Per-pad solder-mask and paste behavior. It is written only when any mask/paste state differs from Common or a custom swell/shrink is set. Each mask/paste attribute is emitted only when its state is non-Common. Segment geometry (`Segment_*` plus `<TopSegments>`/`<BotSegments>`) applies when a paste side is in `Segments` mode.

```
<MaskPaste TopMask="Tented" BotPaste="Segments"
           Segment_Percent="70" Segment_EdgeGap="0.05" Segment_Gap="0.1"
Segment_Side="0.3">
  <BotSegments><Item X1="-0.4" Y1="-0.2" X2="0.4" Y2="0.2"/></BotSegments>
</MaskPaste>
```

Attribute	Type	Written	Description
`TopMask`	enum	if not Common	Top solder-mask mode.
`BotMask`	enum	if not Common	Bottom solder-mask mode.
`TopPaste`	enum	if not Common	Top paste mode.
`BotPaste`	enum	if not Common	Bottom paste mode.
`Segment_Percent`	Real	if Segments	Segmented-paste coverage percentage.
`Segment_EdgeGap`	Real	if Segments	Gap from the pad edge.
`Segment_Gap`	Real	if Segments	Gap between paste segments.
`Segment_Side`	Real	if Segments	Segment side size.
`CustomSwell`	Real	if set	Custom mask swell (omitted when unset / `-1000`).

Attribute	Type	Written	Description
`CustomShrink`	Real	if set	Custom paste shrink (omitted when unset / `-1000`).

Value	Name
`0`	Common (omitted)
`1`	Open (mask) / Solder (paste)
`2`	Tented (mask) / No Solder (paste)
`3`	By Paste (mask) / Segments (paste)

Note. Mask sides use `Common`/`Open`/`Tented`/`By Paste`; paste sides use `Common`/`Solder`/`No Solder`/`Segments`. `<TopSegments>`/`<BotSegments>` hold `<Item>` entries, each a diagonal rectangle given by its two corners `X1 Y1`–`X2 Y2`.

<Categories> / <Category> — library category tree

The library-level classification tree: categories, their types, and subtypes. Every level is identified by `Number` and named by a text `<Name>` child.

```
<Categories>
  <Category Number="0">
    <Name>Capacitors</Name>
    <Types>
      <Type Number="0"><Name>Ceramic</Name>
        <SubTypes><SubType Number="0"><Name>X7R</Name></SubType></SubTypes>
      </Type>
    </Types>
  </Category>
</Categories>
```

Attribute	Type	Written	Description
`Number`	Int	always	Index of the category / type / subtype within its list. Present on `<Category>`, `<Type>` and `<SubType>` alike.

Note. The category *list* here uses `Number`, but a pattern's *reference* into it (the per-pattern `<Category>`) uses `Index` — the two contexts deliberately use different attribute names in PattEdit.

<Pattern> — a footprint

One footprint inside `<Patterns>`: its identity, extent, creation template, metadata text, origin, and the object collections (`<Pads>`, `<Shapes>`, `<Holes>`, `<Dimensions>`, `<Model3D>`). `Id` is the pattern's position in the library and is written on every pattern.

```
<Pattern Id="0" RefDes="U" Mounting="SMD" Width="4" Height="4" Orientation="0"
         LockTypeChange="N" Type="IPC-7351" Float1="0" Float2="0" Float3="0"
Int1="0" Int2="0">
  <Name>BGA16_4X4</Name>
  <DefPad Style="PadT7"/>
  <Pads> … </Pads>
</Pattern>
```

Attribute	Type	Written	Description
`Id`	Int	always	0-based index of the pattern in the library; used on re-import to re-target an existing record.
`RefDes`	Text	if non-empty	Default reference-designator prefix.
`Mounting`	enum	always	Mounting kind — see enum (there is no None value).
`Width`	Real	always	Extent of the footprint objects (X).
`Height`	Real	always	Extent of the footprint objects (Y).
`Orientation`	enum	always	Metadata orientation `0`/`90`/`180`/`270` — does not rotate the stored geometry.
`LockTypeChange`	Bool (Y/N)	always	Whether changing the creation type is locked.
`Type`	enum	always	Creation-template type — see enum below.
`Float1`	Real	always	Creation-template parameter 1.
`Float2`	Real	always	Creation-template parameter 2.
`Float3`	Real	always	Creation-template parameter 3.
`Int1`	Int	always	Creation-template integer parameter 1.
`Int2`	Int	always	Creation-template integer parameter 2.

Value	Name
`0`	Through
`1`	SMD
`2`	Chassis

Value	Name
3	Mixed

Creation-template `Type` values:

Value	Name
-1	None (legacy)
0	Free
1	Right Angle (legacy)
2	Left Angle (legacy)
3	T (legacy)
4	Circle
5	Lines
6	Square
7	Matrix
8	Rectangle
9	Zig-Zag
10	IPC-7351

> **Note.** `Mounting` has no `None` value — an unrecognized string resolves to no value. `Orientation` is metadata only and does not transform the pad/shape coordinates. The `Type` values `None`, `Right Angle`, `Left Angle` and `T` are legacy and only accepted from old files.

<Pattern> metadata text elements

Text children carrying the footprint's descriptive fields. Each is written when non-empty.

```
<Name>BGA16_4X4</Name>
<Name_Description>16-ball BGA</Name_Description>
<Name_Unique>ROHM_BGA016W030</Name_Unique>
<Value/> <Manufacturer>ROHM</Manufacturer> <Datasheet>…</Datasheet>
<PossibleNames><PossibleName>CSBGA16</PossibleName></PossibleNames>
```

Element	Type	Written	Description
`Name`	Text	if non-empty	Original / primary pattern name.
`Name_Description`	Text	if non-empty	Descriptive name.
`Name_Unique`	Text	if non-empty	Unique name.
`Value`	Text	if non-empty	Default value.

Element	Type	Written	Description
`Manufacturer`	Text	if non-empty	Manufacturer.
`Datasheet`	Text	if non-empty	Datasheet reference (emitted in PattEdit/PCB).
`PossibleNames`	list	if any	Holds one `<PossibleName>` text child per alternate name.

<Category> — per-pattern category reference

The pattern's reference into the library category tree. Unlike the library-level list, this reference uses `Index` (and `Index` on its nested type/subtype entries).

```
<Category Index="0">
  <Name>Capacitors</Name>
  <CategoryTypes><CategoryType><Type Index="0">Ceramic</Type></CategoryType></
CategoryTypes>
</Category>
```

Attribute	Type	Written	Description
`Index`	Int	if >−1	Referenced category index (on `<Category>`, and on the nested `<Type>`/`<SubType>`).

> **Note.** The reference carries a `<Name>` text child and an optional `<CategoryTypes>` subtree; the subtype entry is emitted only when its subtype index is `>-1`.

<Origin> — pattern origin marker

The origin offset from the pattern center plus its on-screen marker flags. It has no angle attribute.

```
<Origin X="-1.95" Y="0.35" Cross="Y" Circle="Y" Common="Hide" Courtyard="Show"/>
```

Attribute	Type	Written	Description
`X`	Real	always	Origin X offset from pattern center.
`Y`	Real	always	Origin Y offset from pattern center.
`Cross`	Bool (Y/N)	always	Show the cross marker.
`Circle`	Bool (Y/N)	always	Show the circle marker.
`Common`	enum	always	Common-marker visibility — `Show`/`Hide`/`Show if not center`.
`Courtyard`	enum	always	Courtyard visibility — `Show`/`Hide`.

<RecoveryCode> — IPC generator string

The IPC-7351 generator recovery string; written when non-empty (empty for hand-drawn footprints). Its text content is the code.

```
<RecoveryCode Generator="Y" Model="Y">…code…</RecoveryCode>
```

Attribute	Type	Written	Description
`Generator`	Bool (Y/N)	always	Generator flag.
`Model`	Bool (Y/N)	always	Model flag.

<Groups> — object groups

Object-grouping definitions for the pattern. Each `<Group>` gives the group's number (as `Id`) and its position; pads, shapes, holes and dimensions reference a group by its number via their `Group` attribute.

```
<Groups><Group Id="0" X="1.2" Y="0.5"/></Groups>
```

Attribute	Type	Written	Description
`Id`	Int	always	Group number (the value objects reference).
`X`	Real	always	Group anchor X.
`Y`	Real	always	Group anchor Y.

<AddFields> — user fields

Additional user-defined fields. Each `<AddField>` has a type plus `<Name>` and `<Text>` text children.

```
<AddFields>
  <AddField Type="Text"><Name>Tolerance</Name><Text>5%</Text></AddField>
</AddFields>
```

Attribute	Type	Written	Description
`Type`	enum	always	Field kind — `Text` or `Link`.

<Suppliers> — supplier record

Supplier / part-sourcing data for the pattern, written as text children (all always emitted inside the element).

```
<Suppliers>
  <PartNumber>…</PartNumber> <Manufacturer>…</Manufacturer> <Pattern>…</Pattern>
  <Supplier>…</Supplier> <Currency>USD</Currency>
</Suppliers>
```

Element	Type	Written	Description
`PartNumber`	Text	always	Octopart / part number.
`Manufacturer`	Text	always	Manufacturer name.
`Pattern`	Text	always	Supplier pattern name.
`Supplier`	Text	always	Supplier name.
`Currency`	Text	always	Currency code.

<DefPad> — default pad style

The default pad style used when new pads are added to this pattern.

```
<DefPad Style="PadT7"/>
```

Attribute	Type	Written	Description
`Style`	Text	always	Name of a `<PadStyle>`.

<Pad> — a footprint pad

A pad instance inside `<Pads>`. It references a shared pad style by name and places it at an offset from the pattern center. The pad's printed number is the text child `<Number>`.

```
<Pad Id="1" Style="PadT7" X="-0.7849" Y="5.2299" Angle="0" Locked="N" Side="Top">
  <Number>A1</Number>
</Pad>
```

Attribute	Type	Written	Description
`Id`	Int	always	Pad sequence number within the pattern.
`Style`	Text	always	Referenced `<PadStyle>` name.
`X`	Real	always	X offset from the pattern center.
`Y`	Real	always	Y offset from the pattern center.
`Angle`	Real	always	Pad rotation in radians CCW.
`Locked`	Bool (Y/N)	always	Whether the pad is locked.
`Side`	enum	always	Pad side — `Top` or `Bottom`.
`Group`	Int	if grouped	Group number; present only when the pad belongs to a group.

Note. The pad name is the text child `<Number>` (what a symbol pin binds to); an optional `<Note>` child carries a hint. `Enabled` is import-only — the writer never emits it (disabled pads are simply skipped).

<Shape> — silkscreen / assembly graphic or text

A graphic or text object inside `<Shapes>`. Geometry is a `<Points>` list of center-relative vertices; a `Text` shape adds font/alignment attributes and a `<TextLines>` child. A shape attached to a pad carries a `PadId`.

```
<Shape Id="1" Type="Line" Locked="N" Layer="Top Silk" LineWidth="0.15"
AllLayers="N">
  <Points><Point X="-1.35" Y="-1.5"/><Point X="1.65" Y="1.5"/></Points>
</Shape>
```

Attribute	Type	Written	Description
`Id`	Int	always	Shape sequence number within the pattern.
`Type`	enum	always	Shape kind — see enum below.
`Locked`	Bool (Y/N)	always	Whether the shape is locked.
`Layer`	enum	always	Target layer — single-literal token (see note).
`LineWidth`	Real	non-Text, if custom	Custom stroke width; present only when the object overrides the layer's common width (else `-1`).
`FontVector`	Bool (Y/N)	Text only	Vector font vs. TrueType.
`FontMono`	Bool (Y/N)	Text, vector only	Monospaced vector font; present only when `FontVector="Y"`.
`FontName`	Text	Text only	Font name.
`FontSize`	Int	Text only	Rounded font size.
`FontSizeFloat`	Real	Text only	Exact font size.
`FontScale`	Real	Text only	Font aspect scale.
`FontWidth`	Real	Text only	Stroke-width factor.
`TextShow`	enum	Text only	Which field the text renders (see enum).
`HorzAlign`	enum	Text only	Horizontal alignment — `Center`/`Right`/`Left`.

Attribute	Type	Written	Description
VertAlign	enum	Text only	Vertical alignment — Center/Bottom/Top.
TextAlign	enum	Text only	Multi-line justification — Center/Right/Left.
LineSpacing	Real	Text only	Line-spacing factor.
TextWidth	Real	Text, if lines	Computed text bounding width.
TextHeight	Real	Text, if lines	Computed text bounding height.
Angle	Real	Text only	Text rotation in radians CCW.
AllLayers	Bool (Y/N)	always	Place a signal shape on all signal layers.
PadId	Int	if attached	Sequence number of the <Pad> this shape/marking belongs to; absent when unlinked.
Group	Int	if grouped	Group number; present only when the shape is grouped.

Value	Name
0	None
1	Line
2	Rectangle
3	Obround
4	FillRect
5	FillObround
6	Arc
7	Text
8	Polyline
9	Polygon

Note. The fill-rectangle type is spelled FillRect, not FillRectangle. Layer values are single literal tokens: Top Silk, Top Assy, Top Mask, Top Paste, Bottom Paste, Bottom Mask, Bottom Assy, Bottom Silk, Top, Top Keepout, Bottom Keepout, Bottom, Board Cutout, Top Dimension, Bottom Dimension, Non-Signal, Top Courtyard, Bottom Courtyard, Top Outline, Bottom Outline, Top Terminals, Bottom Terminals. There is no combined Top/Bottom X form; an unrecognized string resolves to no layer. TextShow values: Any Text,

`Name`, `RefDes`, `Value`, `Manufacturer`, `Unique Name`, `Datasheet`. A `Text` shape holds its lines in a `<TextLines>`/`<TextLine>` child.

<Hole> — mounting hole

A mounting hole inside `<Holes>`, positioned relative to the pattern center.

```
<Hole Id="1" Locked="Y" X="-7.1982" Y="1.3482" Diam="2" HoleDiam="1"/>
```

Attribute	Type	Written	Description
`Id`	Int	always	Hole sequence number within the pattern.
`Locked`	Bool (Y/N)	always	Whether the hole is locked.
`X`	Real	always	X offset from the pattern center.
`Y`	Real	always	Y offset from the pattern center.
`Diam`	Real	always	Keepout diameter.
`HoleDiam`	Real	always	Drilled-hole diameter.
`Group`	Int	if grouped	Group number; present only when the hole is grouped.

Note. `Enabled` is import-only — the writer never emits it (disabled holes are pruned).

<Dimension> — a dimension object

A dimension inside `<Dimensions>`. It stores its type, two measured points and a dimension-line origin, its font, and the objects it is connected to (footprint pad/shape/hole/terminal). The label text is the child `<PointerText>`.

```
<Dimension Locked="N" Type="Horizontal" Connected1="Pad" Object1="1"
SubObject1="0" Point1="0"
          Connected2="Pad" Object2="2" SubObject2="0" Point2="0" Layer="Top
Assy"
          X1="0" Y1="0" X2="4" Y2="0" XD="0" YD="-1" ArrowSize="0.3" Units="mm"
          FontVector="Y" FontName="Tahoma" FontSize="8" FontSizeFloat="8"
FontScale="1"
          FontWidth="0.1" ShowUnits="Y" Angle="0" ExternalRadius="0"
PointerMode="0">
  <PointerText>4.0</PointerText>
</Dimension>
```

Attribute	Type	Written	Description
`Locked`	Bool (Y/N)	always	Whether the dimension is locked.

Attribute	Type	Written	Description
`Type`	enum	always	Dimension kind — `Horizontal`/`Vertical`/`Free`/`Radius`/`Pointer`.
`Connected1`	enum	always	First endpoint object kind — `Pad`/`Shape`/`Hole`/`Terminal`.
`Object1`	Int	always	First connected object.
`SubObject1`	Int	always	First connected sub-object.
`Point1`	Int	always	First connected point.
`Connected2`	enum	always	Second endpoint object kind.
`Object2`	Int	always	Second connected object.
`SubObject2`	Int	always	Second connected sub-object.
`Point2`	Int	always	Second connected point.
`Layer`	enum	always	Target layer (same layer token list as `<Shape>`).
`X1`, `Y1`	Real	always	First point.
`X2`, `Y2`	Real	always	Second point.
`XD`, `YD`	Real	always	Dimension-line origin.
`ArrowSize`	Real	always	Arrow size.
`Units`	enum	always	Displayed units — `Common`/`inch`/`mil`/`mm`.
`FontVector`	Bool (Y/N)	always	Vector font vs. TrueType.
`FontMono`	Bool (Y/N)	vector only	Monospaced vector font; present only when `FontVector="Y"`.
`FontName`	Text	if non-empty	Font name.
`FontSize`	Int	always	Rounded font size.
`FontSizeFloat`	Real	always	Exact font size.
`FontScale`	Real	always	Font aspect scale.
`TextWidth`	Real	if text	Computed label width.
`TextHeight`	Real	if text	Computed label height.
`FontWidth`	Real	always	Stroke-width factor.
`ShowUnits`	Bool (Y/N)	always	Append the unit suffix to the label.
`Angle`	Real	always	Rotation in radians CCW.

Attribute	Type	Written	Description
ExternalRadius	Real	always	External-radius factor.
PointerMode	Int	always	Pointer mode.
Group	Int	if grouped	Group number; present only when grouped.

Note. The label text is the child <PointerText>, written when non-empty. Enabled is import-only — the writer never emits it (disabled dimensions are pruned).

<Model3D> — attached 3D model

The 3D model attached to the footprint, always emitted per pattern. It carries the model file (as an environment-string <Filename> with <Path>/<Var>) plus <Rotate>, <Offset> and <Zoom> transforms, each with X/Y/Z.

```
<Model3D Mirror="N" NoSearch="N" Units="mm" IPC_XOff="0" IPC_YOff="0"
         AutoHeight="0.9" AutoColor="0" Type="File" KeepPins="N">
  <Filename><Path>models\bga16.step</Path><Var>$(MODELS)\bga16.step</Var></
Filename>
  <Rotate X="0" Y="0" Z="0"/> <Offset X="0" Y="0" Z="0"/> <Zoom X="1" Y="1"
Z="1"/>
</Model3D>
```

Attribute	Type	Written	Description
Mirror	Bool (Y/N)	always	Mirror the model.
NoSearch	Bool (Y/N)	always	Suppress auto file search.
Units	enum	always	Model file units — mm/mil/inch/Wings.
IPC_XOff	Real	always	IPC X offset.
IPC_YOff	Real	always	IPC Y offset.
AutoHeight	Real	always	Auto-generated model height.
AutoColor	Real	always	Auto-generated model color.
Type	enum	always	Model source — File/IPC-7351/Outline.
KeepPins	Bool (Y/N)	always	Keep generated pins.

Child transform elements — <Rotate> (degrees), <Offset>, <Zoom> — each carry:

Attribute	Type	Written	Description
`X`	Real	always	X component.
`Y`	Real	always	Y component.
`Z`	Real	always	Z component.

Note. The `Units` value `mm` means **millimeters**. `<Filename>` is emitted only when a path or variable is present; `<Rotate>` values are in degrees.