DipTrace Component 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 Type="DipTrace-ComponentLibrary"> — component library root 4
<SubFolders> — library subfolder list 5
<Categories> — library classification tree 5
<Components> / <Component> — the components 6
<Part> — a component section 6
<Pins> / <Pin> — symbol pins 10
<Shapes> / <Shape> — symbol graphics and text 13
<Groups> / <Group> — pin/shape groups 15
<AddFields> / <AddField> — user fields 15
<InternalConnections> / <IntCon> — internally shorted pads 16

How to read this specification

This document describes the **DipTrace Component 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 component library is a single `<Library>` root; it **nests** the footprint (pattern) library, then the category tree, then the components:

```
<Library Type="DipTrace-ComponentLibrary" Name="..." Version="5.x" UID32="..."
Units="mm">
```

```
  <Library Type="DipTrace-PatternLibrary" ...>...</Library>   <!-- attached
footprints -->
  <SubFolders>...</SubFolders> <Categories>...</Categories>
  <Components> <Component> <Part ...>...</Part> </Component> ... </Components>
</Library>
```

A plug-in's data node is `/Library/Components/Component/Part`. Each nested `<Pattern>` in the footprint library carries a `PatternStyle` key that a component `<Part>` references to attach its footprint. The nested pattern library's internals are given in the Pattern Editor specification.

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 Type="DipTrace-ComponentLibrary"> — component library root

Root of a Component Editor library (`.eli` binary / `.elixml` text). It carries the library-wide attributes, then holds one **nested** pattern library (the attached footprints), an optional `<SubFolders>` list, a `<Categories>` tree, and the `<Components>` collection.

```
<Library Type="DipTrace-ComponentLibrary" Name="Audio Buzzers" Hint="Audio -
Buzzers"
         Version="5.3.0.0" UID32="699252860" Units="inch">
  <Library Type="DipTrace-PatternLibrary" Units="inch"> … </Library>
  <SubFolders> … </SubFolders>
  <Categories> … </Categories>
  <Components>
    <Component Id="0"> … </Component>
  </Components>
</Library>
```

Attribute	Type	Written	Description
`Type`	Text	always	Fixed marker `DipTrace-ComponentLibrary`.
`Name`	Text	full save	Library name.
`Hint`	Text	full save	Library caption / hint.
`Version`	Text	full save	DipTrace product version that wrote the file.
`UID32`	Int	full save	Random 32-bit library id, freshly regenerated on every save; if absent on import a new one is assigned.
`Units`	Text	always	File unit for all Reals: `inch`, `mm`, or `mil`.

Note. On a partial plug-in/exchange export only `Type` and `Units` are emitted; `Name`/`Hint`/`Version`/`UID32` are written only on a full library save.

<Library Type="DipTrace-PatternLibrary"> — nested footprint library

Immediately inside the component-library root sits a complete pattern library holding the footprints. When embedded this way only `Type` and `Units` are written on it; each footprint inside is

a <Pattern> carrying a PatternStyle key that a component <Part> references. Its full internals (pad types, pads, holes, shapes, 3D model) are the PattEdit dialect.

```
<Library Type="DipTrace-PatternLibrary" Units="inch">
  <PadTypes> … </PadTypes>
  <Categories> … </Categories>
  <Patterns>
    <Pattern PatternStyle="PatType0" Id="0"> … </Pattern>
  </Patterns>
</Library>
```

Attribute	Type	Written	Description
Type	Text	always	Fixed marker DipTrace-PatternLibrary.
Units	Text	always	Same unit convention as the enclosing library.

Note. The PatternStyle attribute (reader alias PatternType, since 4.2.0.1) on each nested <Pattern> is the unique key referenced by a part's <Pattern Style="…"/>. It is emitted only when the pattern library is embedded inside a component library.

<SubFolders> — library subfolder list

An ordered list of the library's subfolder names, written right after the root attributes when the library has any subfolders. A component's <Part SubFolderIndex="…"> indexes into this list.

```
<SubFolders>
  <Item>Analog</Item>
  <Item>Power</Item>
</SubFolders>
```

Element	Type	Written	Description
<Item>	Text	per subfolder	One subfolder name; position in the list is its index.

<Categories> — library classification tree

The library's three-level classification tree: each <Category> holds <Types>, each <Type> holds <SubTypes>. Every level carries an Index attribute and a <Name> text child. A component's per-part <Category> block references this tree by Index.

```
<Categories>
  <Category Index="1">
    <Name>Buzzers</Name>
    <Types>
      <Type Index="0">
        <Name>Magnetic</Name>
```

```
        <SubTypes>
          <SubType Index="0"><Name>SMD</Name></SubType>
        </SubTypes>
      </Type>
    </Types>
  </Category>
</Categories>
```

Attribute	Type	Written	Description
`Index`	Int	always	Identifier of the category / type / subtype at its level. A component references these by the same `Index` value.

Note. CompEdit uses `Index` for both the library category list and the per-component reference. (The PattEdit dialect uses `Number` for the same tree — the two dialects genuinely differ.)

<Components> / <Component> — the components

The `<Components>` collection holds one `<Component>` per library entry. A component is one or more `<Part>` sections (gates/units) plus an attached footprint linked from its first part.

```
<Component Id="0">
  <Part Id="0" …> … </Part>
  <Part Id="1" …> … </Part>
</Component>
```

Attribute	Type	Written	Description
`Id`	Int	always	Zero-based index of the component in the library.

<Part> — a component section

One symbol section of the component. Several fields are written on the **first part only** and inherited by the rest: `RefDes`, `<Name>`, `<Datasheet>`, `<Manufacturer>`, `<Category>`, `<Suppliers>`, `<AddFields>`, `<LibPath>`, `<Pattern>`, and `<InternalConnections>`. Per-part fields (`<PartName>`, `<Value>`, `<Origin>`, `<SpiceModel>`, `<Pins>`, `<Shapes>`, `<Groups>`) are written on every section.

```
<Part Id="0" RefDes="LS" PartType="Normal" Type="Free" ShowNumbers="Common"
      Int1="0" Int2="0" Width="0.2" Height="0.3" LockTypeChange="Y">
  <Pattern Style="PatType0"/>
  <Name>AI-1223-TWT-12V-R</Name>
  <PartName>Part 1</PartName>
  <Value>47k</Value>
  <LibPath><Path>C:\Lib\Buzzers.eli</Path><Var>$(LIB)</Var></LibPath>
  <Origin X="0" Y="0"/>
  <Datasheet>http://…</Datasheet>
  <SpiceModel Type="Diode"><Model>…</Model><Details><Item>…</Item></Details></
```

```
SpiceModel>
  <Manufacturer>ROHM Semiconductor</Manufacturer>
  <Category Index="1">…</Category>
  <Suppliers>…</Suppliers>
  <Pins>…</Pins>
  <Shapes>…</Shapes>
  <Groups><Group Id="0" X="0" Y="0"/></Groups>
  <AddFields>…</AddFields>
  <InternalConnections><IntCon X="1" Y="8"/></InternalConnections>
</Part>
```

Attribute	Type	Written	Description
`Id`	Int	always	Zero-based section index within the component.
`RefDes`	Text	first part only	Reference-designator prefix (e.g. `LS`).
`PartType`	enum	always	Section electrical role (table below).
`ShowNumbers`	enum	if > 0	Pin-number display mode; omitted when `Common`.
`Type`	enum	always	Symbol template (table below).
`Int1`	Int	always	Template parameter 1 (meaning depends on `Type`).
`Int2`	Int	always	Template parameter 2 (meaning depends on `Type`).
`Width`	Real	always	Symbol body width.
`Height`	Real	always	Symbol body height.
`LockTypeChange`	Bool	always	`Y`/`N` — lock the symbol template against change.
`SubFolderIndex`	Int	—	Index into the library `<SubFolders>` list, or `-1`. Accepted on import but currently **not written** by the serializer.

Value	Name (PartType)
`0`	Normal
`1`	Power
`2`	Net Port
`3`	Hierarchy Connector
`4`	Hierarchy Block

Value	Name (Type / template)
`0`	Free
`1`	2 Sides
`2`	IC-2 Sides
`3`	IC-4 Sides

Value	Name (ShowNumbers)
`0`	Common
`1`	Show
`2`	Hide

> **Note.** `SubFolderIndex` is defined and read but its emit is disabled in the current serializer, so a saved file omits it. The value, when supplied, is the subfolder index (or `-1`) — not the always-zero placeholder older documentation described.

<Part> value elements — <Name> / <PartName> / <Value> / <Datasheet> / <Manufacturer>
Text children of a part. `<PartName>` and `<Value>` are per-section; `<Name>`, `<Datasheet>` and `<Manufacturer>` are written on the first part only. `<Origin>` gives the offset from the symbol center to the origin.

```
<Origin X="0" Y="0"/>
```

Element / Attr	Type	Written	Description
`<Name>`	Text	first part only	Component base name.
`<PartName>`	Text	always	Section label (e.g. `Part 1`).
`<Value>`	Text	always	Component value.
`<Datasheet>`	Text	first part only	Datasheet URL / reference.
`<Manufacturer>`	Text	first part only	Manufacturer name.
`Origin X`	Real	always	Origin offset X from symbol center.
`Origin Y`	Real	always	Origin offset Y (sign-flipped on write/read).

<LibPath> — source-library path
Records where the part's source library lives, as a full path plus an optional environment-variable form. Written on the first part only, and only when either value is non-empty (so ordinary library files omit it).

```
<LibPath>
  <Path>C:\Lib\Buzzers.eli</Path>
  <Var>$(LIB)\Buzzers.eli</Var>
</LibPath>
```

Element	Type	Written	Description
`<Path>`	Text	if present	Full source-library path.
`<Var>`	Text	if present	Environment-variable-relative form of the path.

<Pattern Style="..."/> — attached footprint link

Binds the symbol to its footprint. `Style` matches a `PatternStyle` in the nested pattern library. Written on the first part only.

```
<Pattern Style="PatType0"/>
```

Attribute	Type	Written	Description
`Style`	Text	first part only	Key of the footprint in the nested pattern library.

Note. The reader also accepts `PatternType` as a 4.2.0.1 alias for `Style`; the writer emits `Style`.

<SpiceModel> — SPICE simulation model

Per-section SPICE model descriptor. `<Details>` holds an ordered list of `<Item>` lines.

```
<SpiceModel Type="Diode">
  <Model>D1N4148</Model>
  <Details><Item>.model D1N4148 D(…)</Item></Details>
</SpiceModel>
```

Attribute / Element	Type	Written	Description
`Type`	enum	always	SPICE primitive: `SubCkt`, `Resistor`, `Capacitor`, `Inductor`, `Diode`, `BJT`, `JFET`, `MOSFET`, and further source/line/switch types.
`<Model>`	Text	if present	Model name / reference string.
`<Item>`	Text	per line	One model-detail line inside `<Details>`.

<Category> (per component) — classification reference

First-part-only reference into the library `<Categories>` tree. Carries the category `Index`, a `<Name>`, and a `<CategoryTypes>` list of chosen type/subtype pairs.

```
<Category Index="1">
  <Name>Buzzers</Name>
  <CategoryTypes>
    <CategoryType>
      <Type Index="0">Magnetic</Type>
      <SubType Index="0">SMD</SubType>
    </CategoryType>
  </CategoryTypes>
</Category>
```

Attribute / Element	Type	Written	Description
`Index`	Int	if > −1	Category index into the library tree.
`<Name>`	Text	if present	Category name.
`<Type Index>`	Int	per type	Chosen type index; element text is its name.
`<SubType Index>`	Int	if > −1	Chosen subtype index; element text is its name.

<Suppliers> — supplier record

First-part-only supplier / part-number information.

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

Element	Type	Written	Description
`<PartNumber>`	Text	if present	Octopart / catalog part number.
`<Manufacturer>`	Text	if present	Manufacturer name.
`<Pattern>`	Text	if present	Supplier pattern / package name.
`<Supplier>`	Text	if present	Supplier name.
`<Currency>`	Text	if present	Pricing currency.

<Pins> / <Pin> — symbol pins

Per-section list of symbol pins. Each pin's position is offset from the symbol center; the pin-to-pad bind is by `PadId` plus the matching `<PadNumber>` name.

```
<Pin Id="0" X="0.088" Y="0.1" Locked="N" Type="Default" ElectricType="Undefined"
     Orientation="90" PadId="1" Length="0.15" ShowName="N"
     NumXShift="0" NumYShift="0" NameXShift="0" NameYShift="0"
```

```
     SignalDelay="0" NumOrientation="0" NameOrientation="0">
  <SpiceSignal>NE</SpiceSignal>
  <Name>PLUS</Name>
  <PadNumber>1</PadNumber>
  <NameFont Size="5" FontSizeFloat="5" Width="-2" Scale="1" FontMono="N"/>
</Pin>
```

Attribute	Type	Written	Description
`Id`	Int	always	Zero-based pin index within the section.
`X`	Real	always	Pin X offset from symbol center.
`Y`	Real	always	Pin Y offset (sign-flipped on write/read).
`Locked`	Bool	always	`Y`/`N` — pin locked against edit.
`Type`	enum	always	Pin visual type (table below).
`ElectricType`	enum	always	ERC electrical type: `Undefined`, `Passive`, `Input`, `Output`, `Bidirectional`, `Open High`, `Open Low`, `Passive High`, `Passive Low`, `3 State`, `Power`.
`Orientation`	enum	always	Pin direction: `0` / `90` / `180` / `270`.
`PadId`	Int	always	Numeric index into the footprint pad array (the pin↔pad bind).
`Length`	Real	always	Pin length.
`ShowName`	Bool	always	`Y`/`N` — show the pin name.
`NumXShift`	Real	always	Pad-number label X offset.
`NumYShift`	Real	always	Pad-number label Y offset (sign-flipped).
`NameXShift`	Real	always	Pin-name label X offset.
`NameYShift`	Real	always	Pin-name label Y offset (sign-flipped).
`SignalDelay`	Real	always	Signal delay value.
`NumOrientation`	Real	always	Pad-number label rotation.
`NameOrientation`	Real	always	Pin-name label rotation.
`Group`	Int	if > −1	Owning group id (see `<Groups>`); omitted when ungrouped.

Value	Name (Pin Type)
`0`	Default
`1`	Dot

Value	Name (Pin Type)
2	Polarity In
3	Polarity Out
4	Non Logic
5	Open
6	Open High
7	Open Low
8	3 State
9	Hysteresis
10	Amplifier
11	Postponed
12	Shift
13	Clock
14	Generator

Child element	Type	Written	Description
<SpiceSignal>	Text	if present	SPICE node/signal name for the pin.
<Name>	Text	if present	Pin name / hint.
<PadNumber>	Text	if present	The footprint pad's **name** this pin binds to.

> **Note.** The reader accepts PadIndex as an alias for PadId (since 4.2.0.1); the writer emits PadId. The symbol-pin ↔ footprint-pad link is made by the numeric PadId together with a matching name: <PadNumber> must equal the linked pattern pad's Number.

> **Note.** Enabled on a pin is import-only — honored on the plugin/exchange import path (a delete flag) but never written by a normal save (disabled pins are pruned).

<NameFont> — pin-name font

Font styling for the pin name, always emitted with the pin.

Attribute	Type	Written	Description
Size	Int	always	Rounded integer font size.
FontSizeFloat	Real	always	Exact fractional font size.

Attribute	Type	Written	Description
`Width`	Real	always	Stroke width factor.
`Scale`	Real	always	Font scale factor.
`FontMono`	Bool	always	`Y`/`N` — monospaced vs proportional.

<Shapes> / <Shape> — symbol graphics and text

Per-section list of drawing primitives and text objects. Geometry attributes and children depend on `Type`; text attributes are written only for `Text` shapes. Point coordinates are offsets from the symbol center.

```
<Shape Id="0" Type="Text" Locked="N" FontVector="N" FontMono="N"
       FontSize="6" FontSizeFloat="6.3" FontColor="8388608" TextShow="Any Text"
       FontName="" FontWidth="0" FontScale="0" Angle="1.5708"
       HorzAlign="Left" VertAlign="Top" TextAlign="Left" LineSpacing="1.2"
       TextWidth="8.15" TextHeight="6.53">
  <TextLines><TextLine>Segm</TextLine></TextLines>
  <Points><Point X="0.1" Y="0.15"/></Points>
</Shape>
```

Attribute	Type	Written	Description
`Id`	Int	always	Zero-based shape index within the section.
`Type`	enum	always	Shape kind (table below).
`LineWidth`	Real	if not Text	Stroke width; omitted for `Text`.
`Locked`	Bool	if locked	`Y` emitted only when locked (absent means not locked).
`FontVector`	Bool	Text only	`Y`/`N` — vector stroke font vs TrueType.
`FontMono`	Bool	Text + vector	`Y`/`N`; written only for a vector-font text object.
`FontSize`	Int	Text only	Rounded integer font size.
`FontSizeFloat`	Real	Text only	Exact fractional font size.
`FontColor`	Int	Text only	Integer text color.
`TextShow`	enum	Text only	What the text object displays (table below).
`FontName`	Text	Text, if set	TrueType font name.
`FontWidth`	Real	Text only	Stroke width factor.
`FontScale`	Real	Text only	Font scale factor.

Attribute	Type	Written	Description
`Angle`	Real	Text, if ≠ 0	Text rotation in radians CCW; omitted when zero.
`HorzAlign`	enum	Text only	Horizontal anchor: `Center` / `Right` / `Left`.
`VertAlign`	enum	Text only	Vertical anchor: `Center` / `Bottom` / `Top`.
`TextAlign`	enum	Text only	Multi-line justification: `Center` / `Right` / `Left`.
`LineSpacing`	Real	Text only	Line-spacing factor.
`TextWidth`	Real	Text, computed	Computed text-bounds width (read-only).
`TextHeight`	Real	Text, computed	Computed text-bounds height (read-only).
`Group`	Int	if > −1	Owning group id; omitted when ungrouped.

Value	Name (Shape Type)
`0`	Undefined
`1`	Line
`2`	Arc
`3`	Arrow
`4`	Rectangle
`5`	FillRect
`6`	Obround
`7`	FillObround
`8`	Polyline
`9`	Polygon
`10`	Text

Value	Name (TextShow)
`0`	Any Text
`1`	Name
`2`	RefDes

Value	Name (TextShow)
3	Value
4	Part Name
5	Manufacturer
6	Datasheet

Child element	Type	Written	Description
<TextLines>/<TextLine>	Text	Text shapes only	One <TextLine> per line of literal text.
<Points>/<Point X Y>	Real	if present	Geometry vertices; Y is sign-flipped on write/read.

> **Note.** Type index 0 (Undefined) is a sentinel, not a drawable shape. Enabled on a shape is import-only (a delete flag on the plugin/exchange path) and never written by a normal save.

<Groups> / <Group> — pin/shape groups

Per-section list of named groups that pins and shapes reference by Id. Written by the Component Editor.

```
<Groups>
  <Group Id="0" X="0.1" Y="0.2"/>
</Groups>
```

Attribute	Type	Written	Description
Id	Int	always	Group number referenced by pin/shape Group.
X	Real	always	Group anchor X.
Y	Real	always	Group anchor Y (sign-flipped).

<AddFields> / <AddField> — user fields

First-part-only list of user-defined fields, each a name/value pair with a type.

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

Attribute / Element	Type	Written	Description
Type	enum	always	Field kind: Text or Link.
<Name>	Text	always	Field name.

Attribute / Element	Type	Written	Description
`<Text>`	Text	always	Field value.

<InternalConnections> / <IntCon> — internally shorted pads

First-part-only list of pad pairs shorted inside the component. `X` and `Y` are **pad `Id` references** — the footprint pad's serialized `Id`, not a coordinate and not the printed pad name/number. Because they are references, **before you delete a pad you must remove or repoint every `<IntCon>` that names it**: a dangling pad `Id` here is not validated on import and surfaces later as a crash, not a clean error.

```
<InternalConnections>
  <IntCon X="1" Y="8"/>
</InternalConnections>
```

Attribute	Type	Written	Description
`X`	Int	always	First shorted pad — the footprint pad's `Id`.
`Y`	Int	always	Second shorted pad — the footprint pad's `Id`.