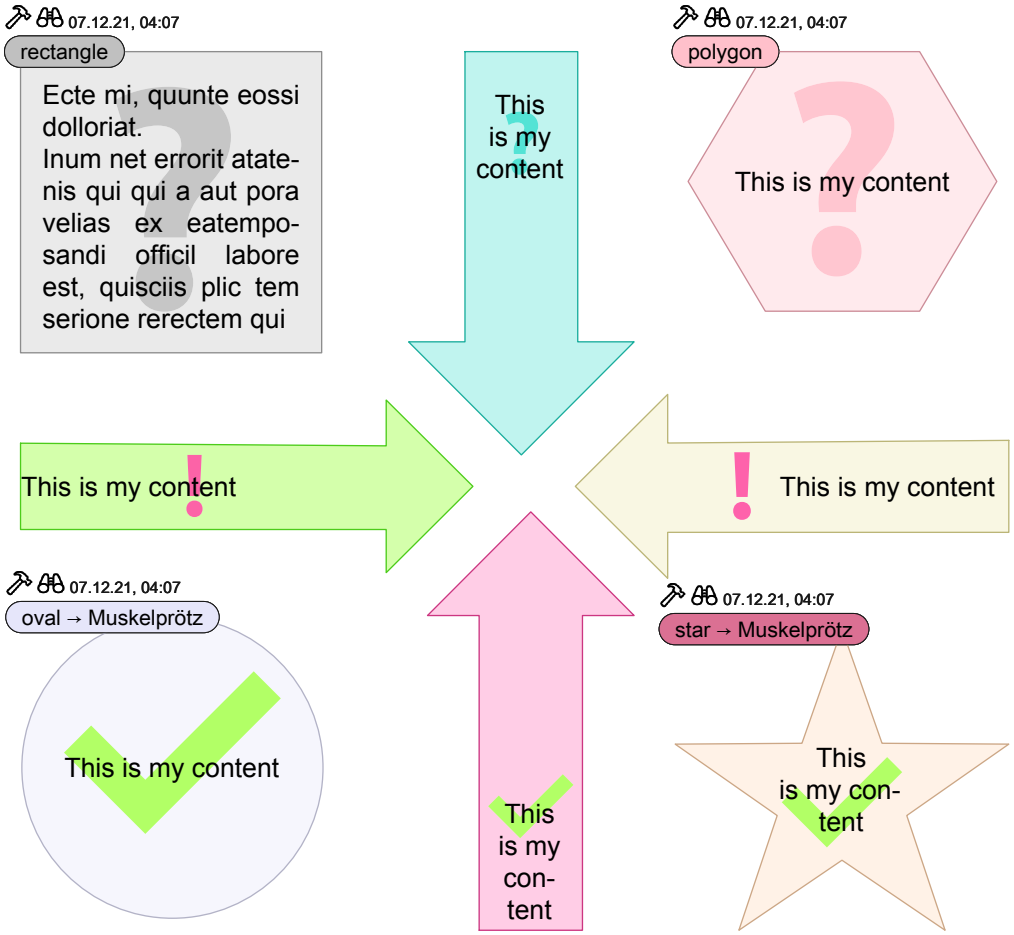
Comet Notes Shapes

The exchange of Comet Note between external systems and InDesign® is done via the so-called *notes.xml*. A complete description of the *notes.xml* can be found here.

This document describes how to **specify** the shape of a note. For geometric and layout definitions of the Comet Notes, please refer to the above documentation.

Built-In Shapes

There are eight built-in shapes. These shapes correspond to the shapes used in the *Preview* app from Apple. Here are screenshots of all build-in shapes:



To assign a default shape to a Come Note, the attribute

`notes.note.shape_type`

of the *notes.xml* is used. The following values are allowed. If the value is empty or undefined, a free-form note is created.

shape-type	Description
`rectangle`	Rectangle or square
`oval`	Circle or ellipse
`polygon`	Uniform hexagon
`star`	Uniform five pointed star
`arrow-left`	Arrow to the indicated direction
`arrow-right`	
`arrow-down`	
`arrow-up`	

If a built-in shape is used, the application (InDesign® or external tool) must be able to create the shape on its own. The definition of the **`area`** element of the note is ignored then and can be empty.

Here is the complete definition of the **`notes`** element of a oval Comet Note.

```
<note
    id="2"
    type="todo"
    shape_type="oval"
    customerID="0"
    customerID2="0"
    customerID3="0"
    customerStringID=""
    customerData1=""
    customerData2=""
    show_title="1"
    show_state="1"
    show_connect="1"
    visible="1"
>
```

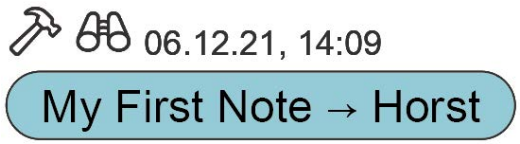
Generell Design

For the exact appearance of the notes, it is certainly useful to display the notes in InDesign®. This requires at least the priint:comet plug-ins **v4.2 R29555**.

Here are the default settings for new notes:

- Opacity 33%
- Solid stroke
  - Weight 0.5pt
  - Aligned centered
  - Opacity 100%
  - Rounded corners and mitered cap
- Inset 4pt

All notes except the arrows are provided with a label consisting of title, status and date (But of course you can turn on/off the title for every note individual).



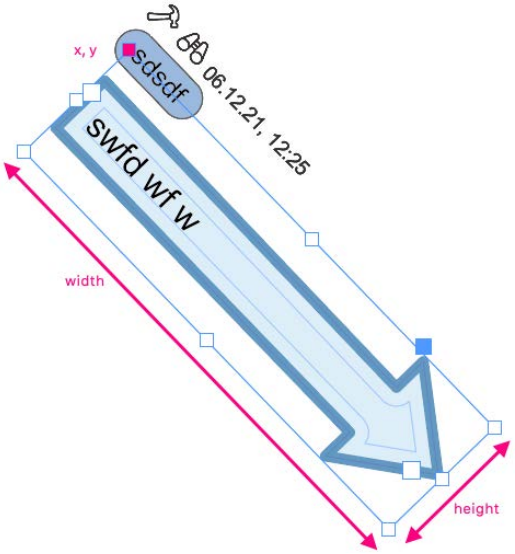
In rectangular notes every single comment is auto-prompted automatically by default. This means that each new comment in the note will automatically be prompted by an author/date label. To save space, the auto-prompt is disabled in all other note types (but can be turned on of course).

Position and Size

The **size** of notes is specified in points the attributes

```
note.reference.bbox.width
note.reference.bbox.height
```

Please note that the size of the **un-rotated frame including its stroke** is used here.



The built-in shapes are automatically adjusted to the specified size. So a rectangle note of size 100 x 100 automatically becomes a square and so on.

The **position** of notes is specified in points the attributes

```
note.reference.x
note.reference.y
```

The upper left corner of the (possibly rotated) frame is used as the position (see screenshot above).

The XY coordinates of this point can be calculated using the rotation matrix:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

rectangle

Depending on the size and scale of the note, a **rectangle** (or square) is created. The upper inset is set to 12 points by default. The text is left asigned.

There should be no difficulty in creating a rectangle shape using SVG.

oval

Depending on the size and scale of the note, an **oval** (or circle) is created. If nothing else is specified, the text frame has an inset of 4 points. The text is centered both vertically and horizontal.

There should be no difficulty in creating a circle shape using SVG.

polygon

Depending on the size and scale of the note, a **hexagon** with a horizontal line at the top is created. If nothing else is specified, the text frame has an inset of 4 points. The text is centered both vertically and horizontal.

There should be plenty of examples in the net of how to construct a hexagon using SVG.

star

Depending on the size and scale of the note, a **five pointed star** is created. If nothing else is specified, the text frame has an inset of 4 points. The text is centered both vertically and horizontal.

There should be plenty of examples in the net of how to construct a five pointed star using SVG.

arrow-~

Dependering on the size and scale of the note, an arrow with an arrow head to the given direction is created.

If nothing else is specified, the text frame has an inset of 4 points. The text is centered both vertically and horizontal.

The thickness of the arrow is half the frame size and the length of the arrow head depends of the arrows thickness.

Here is what we do to calculate the length **a** of the arrow head for InDesign® in C++:

```
PMReal              sx          (bounds.Left ()); // Abbreviations only
PMReal              sy          (bounds.Top ());
PMReal              ex          (bounds.Right());
PMReal              ey          (bounds.Bottom());

PMReal              W           (ex - sx);
PMReal              H           (ey - sy);

PMReal              a;          // Length of the arrowhead
PMReal              h           ((W > H) ? H : W);

if (h >= 60.0)      a           = h / 2.0;
else if (h > 20.0)  a           = h / (2.0 - ((60.0 - h) / 40.0));
else                a           = h;
if (a < 0.0)        a           *= -1.0; // Avoid problems with std::abs
```

For the arrows itself we implemented two shapes, one for horizotal arrows and one for vertical arrow. The direction of the arrows we change by a scaling of -1 in the respective direction.

Here is our C++ code for **horizontal arrows** with the definitions made above for `H`, `W`, `a`, ... :

```
PMReal          h4          (H / 4.0);

pathGeometry->AddPoint (0, PMPoint (sx,       sy + 1.0 * h4));
pathGeometry->AddPoint (0, PMPoint (sx,       sy + 3.0 * h4));
pathGeometry->AddPoint (0, PMPoint (ex - a,   sy + 3.0 * h4));
pathGeometry->AddPoint (0, PMPoint (ex - a,   sy + h));
pathGeometry->AddPoint (0, PMPoint (ex,       sy + 2.0 * h4));  // Arrow head
pathGeometry->AddPoint (0, PMPoint (ex - a,   sy));
pathGeometry->AddPoint (0, PMPoint (ex - a,   sy + 1.0 + h4));
pathGeometry->AddPoint (0, PMPoint (sx,       sy + 1.0 * h4));

if (IsCommandKeyPressed_ ())
{
    PMMatrix    m1, m2;

    m1.ScaleTo (-1.0, 1.0);
    m2.Translate (sx * 2.0 + W, 0.0);
    pathGeometry->TransformPath (m1 * m2);

    fShapeType = cstring ("arrow-left");
}
else
{
    fShapeType = cstring ("arrow-right");
}
```

Here is our C++ code for **vertical arrows** with the definitions made above for `H`, `W`, `a`, ... :

```
PMReal                  w4          (W / 4.0);

pathGeometry->AddPoint (0, PMPoint (sx + w4,        sy));
pathGeometry->AddPoint (0, PMPoint (sx + w4,        ey - a));
pathGeometry->AddPoint (0, PMPoint (sx,             ey - a));
pathGeometry->AddPoint (0, PMPoint (sx + w4 + w4,   ey));  // Arrow head
pathGeometry->AddPoint (0, PMPoint (ex,             ey - a));
pathGeometry->AddPoint (0, PMPoint (ex - w4,        ey - a));
pathGeometry->AddPoint (0, PMPoint (ex - w4,        sy));
pathGeometry->AddPoint (0, PMPoint (sx + w4,        sy));

if (IsCommandKeyPressed_ ())
{
    PMMatrix    m1, m2;

    m1.ScaleTo (1.0, -1.0);
    m2.Translate (0, sy * 2.0 + H);
    pathGeometry->TransformPath (m1 * m2);

    fShapeType = cstring ("arrow-up");
}
else
{
    fShapeType = cstring ("arrow-down");
}
```