XY-pic Reference Manual

Kristoffer H. Rose
⟨krisrose@brics.dk⟩[×]

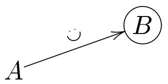
Ross Moore
⟨ross@mpce.mq.edu.au⟩[†]

Version 3.3 ⟨1996/12/19⟩

Abstract

This document summarises the capabilities of the XY-pic package for typesetting graphs and diagrams in TEX. For a general introduction as well as availability information and conditions refer to the User's Guide [14].

A characteristic of XY-pic is that it is built around a *kernel drawing language* which is a concise notation for general graphics, *e.g.*,

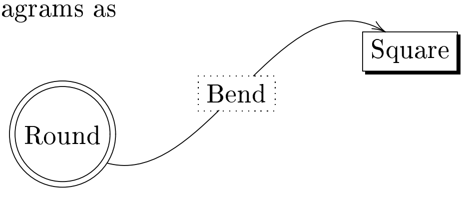


was drawn by the XY-pic kernel code

```
\xy (3,0)*{A} ; (20,6)*+{B}*\cir{} **\dir{-}
    ? *_!/3pt/\dir{)} *_!/7pt/\dir{:}
    ?>* \dir{>} \endxy
```

It is an object-oriented graphic language in the most literal sense: 'objects' in the picture have 'methods' describing how they typeset, stretch, etc. However, the syntax is rather terse.

Particular applications make use of *extensions* that enhance the graphic capabilities of the kernel to handle such diagrams as



which was typeset by

```
\xy *[o]=<40pt>\hbox{Round}="o"*\frm{oo}
    +<5em,-5em>@+,
    (46,11)*+\hbox{Square}="s"  *\frm{-,}
    -<5em,-5em>@+,
 "o";"s" **i\crvs{},
   ?*+\hbox{Bend}="b"*\frm{.} ?>*\dir{>},
 "o";"s"."b" **\crvs{-},
 "o"."b";"s" **\crvs{-}
\endxy
```

using the 'curve' and 'frame' extensions.

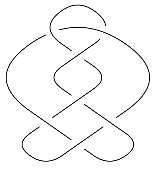
All this is made accessible through the use of *features* that provide convenient notation such that users can enter special classes of diagrams in an intuitive form, *e.g.*, the diagram

$$\begin{tikzcd}
U \arrow[dr] \arrow[ddr, bend right, "y"'] \arrow[drr, bend left, "x"] & & \\
 & X \times_Z Y \arrow[r, "p"'] \arrow[d, "q"] & X \arrow[d, "f"'] \\
 & Y \arrow[r, "g"] & Z
\end{tikzcd}$$

was typeset using the 'matrix' features by the XY-pic input lines

```
\xymatrix{
 U \ar@/_/[ddr]_y \ar[dr] \ar@/^/[drr]^x \\
  & X \times_Z Y \ar[d]^q \ar[r]_p
                & X \ar[d]_f             \\
  & Y \ar[r]^g   & Z                     }
```

Features exist for many kinds of input; here is a knot typeset using the 'knots and links' feature:



The current implementation is programmed entirely within "standard TEX and METAFONT", *i.e.*, using TEX macros (no `\specials`) and with fonts designed using METAFONT. Optionally special 'drivers' make it possible to produce DVI files with 'specials' for extra graphics capabilities, *e.g.*, using POSTSCRIPT.[1]

[×] BRICS, Computer Science, Aarhus University (bld. 540), Ny Munkegade, DK–8000 Aarhus C, Denmark.

[†] MPCE (Mathematics dept.), Macquarie University, North Ryde, Sydney, Australia NSW 2109.

[1] POSTSCRIPT is a registered Trademark of Adobe, Inc. [1].

Contents

List of Figures

Preface

This reference manual gives concise descriptions of the modules of Xy-pic, written by the individual authors. Please direct any TeXnical question or suggestion for improvement directly to the author of the component in question, preferably by electronic mail using the indicated address. Complete documents and printed technical documentation or software is most useful.

The first part documents the Xy-pic kernel which is always loaded. The remaining parts describe the three kinds of options: *extensions* in part II extend the kernel graphic capabilities, *features* in part III provide special input syntax for particular diagram types, and *drivers* in part IV make it possible to exploit the printing capabilities supported by DVI driver programs. For each option it is indicated how it should be loaded. The appendices contain answers to all the exercises, a summary of the compatibility with version 2, and list some reasons why Xy-pic might sometimes halt with a cryptic TeX error.

License. Xy-pic is free software in the sense that it is available under the following license conditions:

In practice this means that you are free to use Xy-pic for your documents but if you distribute any part of Xy-pic

Kristoffer Rose

Ross Moore

(including modified versions) to someone then you are obliged to ensure that the full source text of XY-pic is available to them (the full text of the license in the file `COPYING` explains this in somewhat more detail ☺).

Notational conventions. We give descriptions of the *syntax* of pictures as BNF[2] rules; in explanations we will use upper case letters like X and Y for ⟨dimen⟩sions and lower case like x and y for ⟨factor⟩s.

Part I

The Kernel

Vers. 3.3 by Kristoffer H. Rose ⟨krisrose@brics.dk⟩

After giving an overview of the XY-pic environment in §1, this part document the basic concepts of XY-picture construction in §2, including the maintained 'graphic state'. The following sections give the precise syntax rules of the main XY-pic constructions: the position language in §3, the object constructions in §4, and the picture 'decorations' in §5. §6 presents the kernel repertoire of objects for use in pictures; §7 documents the interface to XY-pic options like the standard 'feature' and 'extension' options.

Details of the implementation are not discussed here but in the complete TEXnical documentation [15].

1 The XY-pic implementation

This section briefly discusses the various aspects of the present XY-pic kernel implementation of which the user should be aware.

1.1 Loading XY-pic

XY-pic is careful to set up its own environment in order to function with a large variety of formats. For most formats a single line with the command

```
\input xy
```

in the preamble of a document file should load the kernel (see 'integration with standard formats' below for variations possible with certain formats, in particular LATEX [9]).

The rest of this section describes things you must consider if you need to use XY-pic together with other macro packages, style options, or formats. The less your environment deviates from plain TEX the easier it should be. Consult the TEXnical documentation [15] for the exact requirements for other definitions to coexist with XY-pic.

Privacy: XY-pic will warn about control sequences it redefines—thus you can be sure that there are no conflicts between XY-pic-defined control sequences, those of your format, and other macros, provided you load XY-pic last and get no warning messages like

```
Xy-pic Warning: '...' redefined.
```

In general the XY-pic kernel will check all control sequences it redefines *except* that (1) generic temporaries like `\next` are not checked, (2) predefined font identifiers (see §1.3) are assumed intentionally preloaded, and (3) some of the more exotic control sequence names used internally (like `@{-}`) are only checked to be different from `\relax`.

Category codes: Unfortunately the situation is complicated by the flexibility of TEX's input format. The culprit is the 'category code' concept of TEX (*cf.* [6, p.37]): when loaded XY-pic requires the characters `␣\{}%` (the first is a space) to have their standard meaning and all other printable characters to have the *same category as when XY-pic will be used*—in particular this means that (1) you should surround the loading of XY-pic with `\makeatother` ... `\makeatletter` when loading it from within a LATEX package, and that (2) XY-pic should be loaded after files that change category codes like the `german.sty` that makes `"` active. Some styles require that you reset the catcodes for every diagram, *e.g.*, with `french.sty` you should use the command `\english` before every `\xymatrix`.
However, it is possible to 'repair' the problem in case any of the characters `#$&'+-.<=>`` ` `` change category code:

```
\xyresetcatcodes
```

will load the file `xyrecat.tex` (version 3.3) to do it.

Integration with standard formats This is handled by the `xyidioms.tex` file and the integration as a LATEX [9] package by `xy.sty`.

xyidioms.tex: This included file provides some common idioms whose definition depends on the used format such that XY-pic can use predefined dimension registers etc. and yet still be independent of the format

---

[2] BNF is the notation for "meta-linguistic formulae" first used in [10] to describe the syntax of the Algol programming language. We use it with the conventions of the TEXbook [6]: '$\longrightarrow$' is read "is defined to be", ' | ' is read "or", and '⟨empty⟩' denotes "nothing"; furthermore, '⟨id⟩' denotes anything that expands into a sequence of TEX character tokens, '⟨dimen⟩' and '⟨factor⟩' denote decimal numbers with, respective without, a dimension unit (like `pt` and `mm`), ⟨number⟩ denotes possibly signed integers, and ⟨text⟩ denotes TEX text to be typeset in the appropriate mode. We have chosen to annotate the syntax with brief explanations of the 'action' associated with each rule; here '$\leftarrow$' should be read 'is copied from'.

under which it is used. The current version (3.3) handles plain TeX (version 2 and 3 [6]), AMS-TeX (version 2.0 and 2.1 [16]), LaTeX (version 2.09 [8] and 2ε [9]), AMS-LaTeX (version 1.0, 1.1 [2], and 1.2), and eplain (version 2.6 [3])[3].

xy.sty: If you use LaTeX then this file makes it possible to load Xy-pic as a 'package' using the LaTeX 2ε [9] `\usepackage` command:

`\usepackage [`⟨option⟩`,...] {xy}`

---

where the ⟨option⟩s will be interpreted as if passed to `\xyoption` (*cf.* §7).

The only exception to this is the options named the same as those driver package options of part IV that appear in *cf.* [4, table 11.2, p.317] or the LaTeX 2ε `graphics` bundle: these will automatically invoke any backend extension required to best emulate the LaTeX 2ε behaviour (this means that, *e.g.*, `[dvips]` and `[textures]` can be used as options to the `\documentclass` command with the normal effect).

The file also works as a LaTeX 2.09 [8] 'style option' although you will then have to load options with the `\xyoption` mechanism described in §7.

1.2 Logo, version, and messages

Loading Xy-pic prints a banner containing the version and author of the kernel; small progress messages are printed when each major division of the kernel has been loaded. Any options loaded will announce themself in a similar fashion.

If you refer to Xy-pic in your written text (please do ☺) then you can use the command `\Xy-pic` to typeset the "Xy-pic" logo. The version of the kernel is typeset by `\xyversion` and the release date by `\xydate` (as found in the banner). By the way, the Xy-pic *name*[4] originates from the fact that the first version was little more than support for (x, y) coordinates in a configurable coordinate system where the main idea was that *all* operations could be specified in a manner independent of the orientation of the coordinates. This property has been maintained except that now the package allows explicit absolute orientation as well.

Messages that start with "`Xy-pic Warning`" are indications that something needs your attention; an "`Xy-pic Error`" will stop TeX because Xy-pic does not know how to proceed.

1.3 Fonts

The Xy-pic kernel implementation makes its drawings using five specially designed fonts:

Font	Characters	Default
`\xydashfont`	dashes	`xydash10`
`\xyatipfont`	arrow tips, upper half	`xyatip10`
`\xybtipfont`	arrow tips, lower half	`xybtip10`
`\xybsqlfont`	quarter circles for hooks and squiggles	`xybsql10`
`\xycircfont`	1/8 circle segments	`xycirc10`

The first four contain variations of characters in a large number of directions, the last contains 1/8 circle segments.

Note: The default fonts are not part of the Xy-pic kernel *specification*: they just set a standard for what drawing capabilities should at least be required by an Xy-pic implementation. Implementations exploiting capabilitites of particular output devices are in use. Hence the fonts are only loaded by Xy-pic if the control sequence names are undefined—this is used to preload them at different sizes or prevent them from being loaded at all.

1.4 Allocations

One final thing that you must be aware of is that Xy-pic allocates a significant number of dimension registers and some counters, token registers, and box registers, in order to represent the state and do computations. The current kernel allocates 4 counters, 28 dimensions, 2 box registers,4 token registers, 1 read channel, and 1 write channel (when running under LaTeX; some other formats use slightly more because standard generic temporaries are used). Options may allocate further registers (currently loading *everything* loads 6 dimen-, 3 toks-, 1 box-, and 9 count-registers in addition to the kernel ones).

2 Picture basics

The basic concepts involved when constructing Xy-pictures are positions and objects, and how they combine to form the state used by the graphic engine.

The general structure of an Xy-picture is as follows:

`\xy` ⟨pos⟩ ⟨decor⟩ `\endxy`

---

builds a box with an Xy-picture (LaTeX users may substitute `\begin{xy}` ... `\end{xy}` if they prefer).

⟨pos⟩ and ⟨decor⟩ are components of the special 'graphic language' which Xy-pictures are specified in. We explain the language components in general terms in this § and in more depth in the following §§.

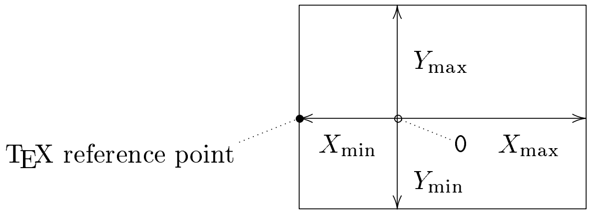
---

[3] The 'v2' feature introduces some name conflicts for compatibility.

[4] No description of a TeX program is complete without an explanation of its name.

2.1 Positions

All *positions* may be written `<`X`,`Y`>` where X is the TeX dimension distance *right* and Y the distance *up* from the *zero position* `0` of the Xy-picture (`0` has coordinates `<0mm,0mm>`, of course). The zero position of the Xy-picture determines the box produced by the `\xy`...`\endxy` command together with the four parameters $X_{\min}$, $X_{\max}$, $Y_{\min}$, and $Y_{\max}$ set such that all the objects in the picture are 'contained' in the following rectangle:

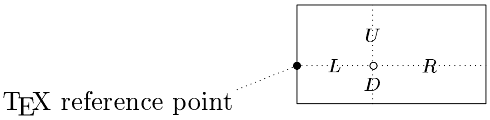


where the distances follow the "up and right > 0" principle, *e.g.*, the indicated TeX reference point has coordinates `<`$X_{\min}$`,0pt>` within the Xy-picture. The zero position does not have to be contained in the picture, but $X_{\min} \le X_{\max} \wedge Y_{\min} \le Y_{\max}$ always holds. The possible positions are described in detail in §3.

When an Xy-picture is entered in *math mode* then the reference point becomes the "vcenter" instead, *i.e.*, we use the point `<`$X_{\min}$`,-\the\fontdimen22>` as reference point.

2.2 Objects

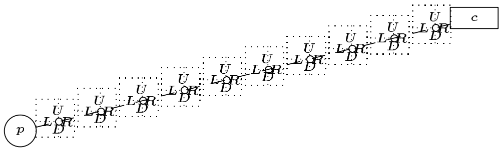
The simplest form of putting things into the picture is to 'drop' an *object* at a position. An object is like a TeX box except that it has a general *Edge* around its reference point—in particular this has the *extents* (*i.e.*, it is always contained within) the dimensions L, R, U, and D away from the reference point in each of the four directions left, right, up, and down. Objects are encoded in TeX boxes using the convention that the TeX reference point of an object is at its left edge, thus shifted `<`$-L$`,0pt>` from the center—so a TeX box may be said to be a rectangular object with $L =$ `0pt`. Here is an example:



The object shown has a rectangle edge but others are available even though the kernel only supports rectangle and circle edges. It is also possible to use entire Xy-pictures as objects with a rectangle edge, `0` as the reference point, $L = -X_{\min}$, $R = X_{\max}$, $D = -Y_{\min}$, and $U = Y_{\max}$. The commands for objects are described in §4.

2.3 Connections

Besides having the ability to be dropped at a position in a picture, all objects may be used to *connect* the two current objects of the state, *i.e.*, p and c. For most objects this is done by 'filling' the straight line between the centers with as many copies as will fit between the objects:



The ways the various objects connect are described along with the objects.

2.4 Decorations

When the `\xy` command reaches something that can not be interpreted as a continuation of the position being read, then it is expected to be a *decoration*, *i.e.*, in a restricted set of TeX commands which add to pictures. Most such commands are provided by the various *user options* (*cf.* §7)—only a few are provided within the kernel to facilitate programming of such options (and user macros) as described in §5.

2.5 The Xy-pic state

Finally we summarise the user-accessible parts of the Xy-picture state of two positions together with the last object associated with each: the *previous*, p, is the position `<`X_p`,` Y_p`>` with the object L_p, R_p, D_p, U_p, $Edge_p$, and the *current*, c, is the position `<`X_c`,` Y_c`>` with the object L_c, R_c, D_c, U_c, $Edge_c$.

Furthermore, Xy-pic has a configurable *cartesian coordinate system* described by an *origin* position `<`X_{origin}`,`Y_{origin}`>` and two *base vectors* `<`X_{xbase}`,`Y_{xbase}`>` and `<`X_{ybase}`,`Y_{ybase}`>` accessed by the usual notation using parentheses:

$$\begin{array}{rcll} (x,y) & = & < & X_{origin} + x \times X_{xbase} + y \times X_{ybase} \quad , \\ & & & Y_{origin} + x \times Y_{xbase} + y \times Y_{ybase} \quad > \end{array}$$

This is explained in full when we show how to set the base in note 3d of §3.

Finally typesetting a connection will setup a "placement state" for referring to positions on the connection that is accessed through a special `?` position construction; this is also discussed in detail in §3.

The Xy-pic *state* consists of all these parameters together. They are initialised to zero except for $X_{xbase} = Y_{ybase} =$ `1mm`.

The edges are not directly available but points on the edges may be found using the different ⟨corner⟩ forms described in §3.

It is possible to insert an 'initial' piece of ⟨pos⟩ ⟨decor⟩ at the start of every Xy-picture with the declaration

---

`\everyxy={` ⟨text⟩ `}`

This will act as if the ⟨text⟩ was typed literally right after each `\xy` command, parsing the actual contents as if it follows this – thus it is recommended that ⟨text⟩ has the form ⟨pos⟩, such that users can continue with ⟨pos⟩ ⟨decor⟩.

3 Positions

A ⟨pos⟩ition is a way of specifying locations as well as dropping objects at them and decorating them—in fact any aspect of the Xy-pic state can be changed by a ⟨pos⟩ but most will just change the coordinates and/or shape of c.

All possible positions are shown in figure 1 with explanatory notes below.

Exercise 1: Which of the positions `0`, `<0pt,0pt>`, `<0pt>`, `(0,0)`, and `/0pt/` is different from the others?

Notes

3a. When doing arithmetic with `+` and `-` then the resulting current object inherits the size of the ⟨coord⟩, *i.e.*, the right argument—this will be zero if the ⟨coord⟩ is a ⟨vector⟩.

Exercise 2: How do you set c to an object the same size as the saved object `"ob"` but moved `<`X`,`Y`>`?

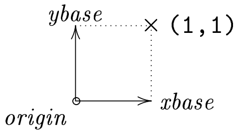
3b. *Skewing* using `!` just means that the reference point of c is moved with as little change to the shape of the object as possible, *i.e.*, the edge of c will remain in the same location except that it will grow larger to avoid moving the reference point outside c.

Exercise 3: What does the ⟨pos⟩ …`!R-L` do?
Bug: The result of `!` is always a rectangle currently.

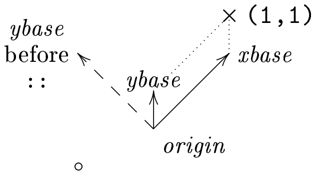
3c. A ⟨pos⟩ *covers* another if it is a rectangle with size sufficiently large that the other is "underneath". The `.` operation "extends" a ⟨pos⟩ to cover an additional one—the reference point of c is not moved but the shape is changed to a rectangle such that the entire p object is covered.

Bug: non-rectangular objects are first "translated" into a rectangle by using a diagonal through the object as the diagonal of the rectangle.

3d. The operations `:` and `::` set the *base* used for ⟨coord⟩inates having the form `(`x`,`y`)`. The `:` operation will set `<`X_{origin}`,` Y_{origin}`>` to p, `<`X_{xbase}`,` Y_{xbase}`>` to $c - origin$, and `<`X_{ybase}`,` Y_{ybase}`>` to `<`$-Y_{xbase}$`,` X_{xbase}`>` (this ensures that it is a usual square coordinate system). The `::` operation may then be used afterwards to make nonsqare bases by just setting *ybase* to $c - origin$. Here are two examples: firstly `0;<1cm,0cm>:` sets the coordinate system



while `<1cm,.5cm>;<2cm,1.5cm>:<1cm,1cm>::` defines



where in each case the ∘ is at `0`, the base vectors have been drawn and the × is at `(1,1)`.

When working with cartesian coordinates these three special ⟨factor⟩s are particularly useful:

`\halfroottwo`	$0.70710678 \approx \frac{1}{2}\sqrt{2}$
`\partroottwo`	$0.29289322 \approx 1 - \frac{1}{2}\sqrt{2}$
`\halfrootthree`	$0.86602540 \approx \frac{1}{2}\sqrt{3}$

More can be defined using `\def` (or `\newcommand` in LaTeX).

3e. An *angle* α in Xy-pic is the same as the coordinate pair `(`$\cos\alpha, \sin\alpha$`)` where α must be an integer interpreted as a number of degrees. Thus the ⟨vector⟩ `a(0)` is the same as `(1,0)` and `a(90)` as `(0,1)`, etc.

3f. To *drop* an ⟨object⟩ at c with `*` means to actually physically typeset it in the picture with reference position at c—how this is done depends on the ⟨object⟩ in question and is described in detail in §4. The intuition with a drop is that it typesets something at `<`X_c`,`Y_c`>` and sets the edge of c accordingly.

3g. The *connect* operation `**` will first compute a number of internal parameters describing the direction from p to c and then typesets a connection filled with copies of the ⟨object⟩ as illustrated in §2.3. The exact details of the connection depend on the actual ⟨object⟩ and are described in general in §4. The intuition with a connection is that it typesets something connecting p and c and sets the `?` ⟨pos⟩ operator up accordingly.

Syntax			Action
⟨pos⟩	⟶	⟨coord⟩	$c \leftarrow$ ⟨coord⟩
	|	⟨pos⟩ `+` ⟨coord⟩	$c \leftarrow$ ⟨pos⟩ $+$ ⟨coord⟩[3a]
	|	⟨pos⟩ `-` ⟨coord⟩	$c \leftarrow$ ⟨pos⟩ $-$ ⟨coord⟩[3a]
	|	⟨pos⟩ `!` ⟨coord⟩	$c \leftarrow$ ⟨pos⟩ then skew[3b] c by ⟨coord⟩
	|	⟨pos⟩ `.` ⟨coord⟩	$c \leftarrow$ ⟨pos⟩ but also covering[3c] ⟨coord⟩
	|	⟨pos⟩ `,` ⟨coord⟩	$c \leftarrow$ ⟨pos⟩ then $c \leftarrow$ ⟨coord⟩
	|	⟨pos⟩ `;` ⟨coord⟩	$c \leftarrow$ ⟨pos⟩, swap p and c, $c \leftarrow$ ⟨coord⟩
	|	⟨pos⟩ `:` ⟨coord⟩	$c \leftarrow$ ⟨pos⟩, set base[3d], $c \leftarrow$ ⟨coord⟩
	|	⟨pos⟩ `::` ⟨coord⟩	$c \leftarrow$ ⟨pos⟩, $\mathit{ybase} \leftarrow c - \mathit{origin}$, $c \leftarrow$ ⟨coord⟩
	|	⟨pos⟩ `*` ⟨object⟩	$c \leftarrow$ ⟨pos⟩, drop[3f] ⟨object⟩
	|	⟨pos⟩ `**` ⟨object⟩	$c \leftarrow$ ⟨pos⟩, connect[3g] using ⟨object⟩
	|	⟨pos⟩ `?` ⟨place⟩	$c \leftarrow$ ⟨pos⟩, $c \leftarrow$ ⟨place⟩[3h]
	|	⟨pos⟩ `@` ⟨stacking⟩	$c \leftarrow$ ⟨pos⟩, do ⟨stacking⟩[3o]
	|	⟨pos⟩ `=` ⟨saving⟩	$c \leftarrow$ ⟨pos⟩, do ⟨saving⟩[3p]
⟨coord⟩	⟶	⟨vector⟩	⟨pos⟩ is ⟨vector⟩ with zero size
	|	⟨empty⟩ | `c`	reuse last c (do nothing)
	|	`p`	p
	|	`x` | `y`	axis intersection[3k] with $\overline{pc}$
	|	`s`⟨digit⟩ | `s{`⟨number⟩`}`	stack[3o] position ⟨digit⟩ or ⟨number⟩ below the top
	|	`"`⟨id⟩`"`	restore what was saved[3p] as ⟨id⟩ earlier
	|	`{` ⟨pos⟩ ⟨decor⟩ `}`	the c resulting from interpreting the group[3l]
⟨vector⟩	⟶	`0`	zero
	|	`<` ⟨dimen⟩ `,` ⟨dimen⟩ `>`	absolute
	|	`<` ⟨dimen⟩ `>`	absolute with equal dimensions
	|	`(` ⟨factor⟩ `,` ⟨factor⟩ `)`	in current base[3d]
	|	`a (` ⟨number⟩ `)`	angle in current base[3e]
	|	⟨corner⟩	from reference point to ⟨corner⟩ of c
	|	⟨corner⟩ `(` ⟨factor⟩ `)`	The ⟨corner⟩ multiplied with ⟨factor⟩
	|	`/` ⟨direction⟩ ⟨dimen⟩ `/`	vector ⟨dimen⟩ in ⟨direction⟩[3m]
⟨corner⟩	⟶	`L` | `R` | `D` | `U`	offset[3n] to left, right, down, up side
	|	`CL` | `CR` | `CD` | `CU` | `C`	offset[3n] to center of side, true center
	|	`LD` | `RD` | `LU` | `RU`	offset[3n] to actual left/down, … corner
	|	`E` | `P`	offset[3n] to nearest/proportional edge point to p
⟨place⟩	⟶	`<` ⟨place⟩ | `>` ⟨place⟩	shave[3h] `(0)`/`(1)` to edge of p/c, $f \leftarrow$ `0`/`1`
	|	`(` ⟨factor⟩ `)` ⟨place⟩	$f \leftarrow$ ⟨factor⟩
	|	⟨slide⟩	pick place[3h] and apply ⟨slide⟩
	|	`! {`⟨pos⟩`}` ⟨slide⟩	intercept[3j] with line setup by ⟨pos⟩ and apply ⟨slide⟩
⟨slide⟩	⟶	`/` ⟨dimen⟩ `/`	slide[3i] ⟨dimen⟩ further along connection
	|	⟨empty⟩	no slide

Figure 1: ⟨pos⟩itions.

3h. Using `?` will "pick a place" along the most recent connection typeset with `**`. What exactly this means is determined by the object that was used for the connection and by the modifiers described in general terms here.

The "shave" modifiers in a ⟨place⟩, `<` and `>`, change the default ⟨factor⟩, f, and how it is used, by 'moving' the positions that correspond to `(0)` and `(1)` (respectively): These are initially set equal to p and c, but shaving will move them to the point on the edge of p and c where the connection "leaves/enters" them, and change the default f as indicated. When one end has already been shaved thus then subsequent shaves will correspond to sliding the appropriate position(s) a TEX `\jot` (usually equal to `3pt`) further towards the other end of the connection (and past it). Finally the *pick* action will pick the position located the fraction f of the way from `(0)` to `(1)` where $f =$ `0.5` if it was not set (by `<`, `>`, or explicitly).

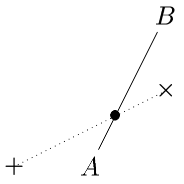
All this is probably best illustrated with some examples: each ⊗ in figure 2 is typeset by a sequence of the form p`;` c `**@{.}` `?`⟨place⟩ `*{\oplus}` where we indicate the `?`⟨place⟩ in each case. (We also give examples of ⟨slide⟩s.)

3i. A ⟨slide⟩ will move the position a dimension further along the connection at the picked position. For straight connections (the only ones kernel XY-pic provides) this is the same as adding a vector in the tangent direction, *i.e.*, `?`…`/`A`/` is the same as `?`…`+/`A`/`.

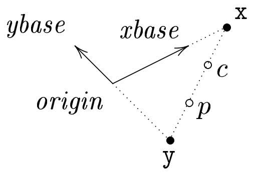
3j. This special ⟨place⟩ finds the point where the last connection intercepts with the line from p to c as setup by the ⟨pos⟩, thus usually this will have the form `!{`⟨coord⟩`;`⟨coord⟩`}`[5], for example, **Bug:** Only works for straight arrows at present.

```
\xy <1cm,0cm>:
 (0,0)*=0{+}="+" ;
 (2,1)*=0{\times}="*" **@{.} ,
 (1,0)*+{A} ; (2,2)*+{B} **@{-}
 ?!{"+";"*"} *{\bullet}
\endxy
```

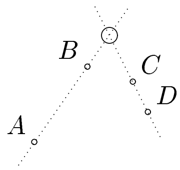
will typeset



3k. The positions denoted by the *axis intersection* ⟨coord⟩inates `x` and `y` are the points where the line through p and c intersects with each axis. The following figure illustrates this:



Exercise 4: Given predefined points A, B, C, and D (stored as objects `"A"`, `"B"`, `"C"`, and `"D"`), write a ⟨coord⟩ specification that will return the point where the lines $\overline{AB}$ and $\overline{CD}$ cross (the point marked with a large circle here):



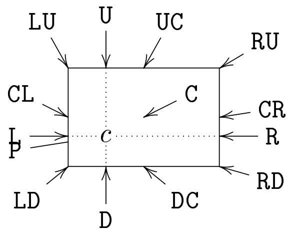
3l. A ⟨pos⟩ ⟨decor⟩ *grouped* in `{}`-braces[6] is interpreted in a local scope in the sense that any p and *base* built within it are forgotten afterwards, leaving only the c as the result of the ⟨coord⟩. **Note:** Only p and *base* are restored – it is not a TEX group.

Exercise 5: What effect is achieved by using the ⟨coord⟩inate "`{;}`"?

3m. The vector `/`Z`/`, where Z is a ⟨dimen⟩sion, is the same as the vector `<`$Z\cos\alpha$`,`$Z\sin\alpha$`>` where α is the angle of the last direction set by a connection (*i.e.*, with `**`) or subsequent placement (`?`) position.

It is possible to give a ⟨direction⟩ as described in the next section (figure 3, note 4l in particular) that will then be used to set the value of α. It is also possible to omit the ⟨dimen⟩ in which case it is set to a default value of `.5pc`.

3n. A ⟨corner⟩ is an offset from the current `<`X_c`,`Y_c`>` position to a specific position on the edge of the c object (the two-letter ones may be given in any combination):



---

[5] The braces can be replaced by `(*`…`*)` *once*, *i.e.*, there can be no other braces nested inside it.