The following code demonstrates a simple example of making use of customized coordinates where a ruler is drawn showing centimeter units (see Figure 3.22).

A blank plot region is set up first and calculations are performed to establish the relationship between user coordinates in the plot and physical centimeters.*

```
> plot(0:1, 0:1, type="n", axes=FALSE, ann=FALSE)
> usr <- par("usr")
> pin <- par("pin")
> xcm <- diff(usr[1:2])/(pin[1]*2.54)
> ycm <- diff(usr[3:4])/(pin[2]*2.54)
```

Now drawing can occur with positions expressed in terms of centimeters. The ruler itself is drawn with a call to `rect()` to draw the edges of the ruler, a call to `segments()` to draw the scale, and calls to `text()` to label the scale.

```
> rect(0, 0, 1, 1, col="white")
> segments(seq(1, 8, 0.1)*xcm, 0,
           seq(1, 8, 0.1)*xcm,
           c(rep(c(0.5, rep(0.25, 4),
                   0.35, rep(0.25, 4)),
                 7), 0.5)*ycm)
> text(1:8*xcm, 0.6*ycm, 0:7, adj=c(0.5, 0))
> text(8.2*xcm, 0.6*ycm, "cm", adj=c(0, 0))
```

*R graphics relies on having accurate information on the physical size of the natural units on the page or screen (e.g., the physical size of pixels on a computer screen). The physical size of output when producing PostScript and PDF files (see Section 9.1) should always be correct, but small inaccuracies may occur when specifying output with a physical size (such as inches) on a graphics window on screen.

The coordinate systems recognized by the base graphics system.

Name	Description
`"user"`	The scales on the plot axes
`"inches"`	Inches, with (0, 0) at bottom-left
`"device"`	Pixels for screen or bitmap output, otherwise 1/72in.
`"ndc"`	Normalized coordinates, with (0, 0) at bottom-left and (1, 1) at top-right, within the entire device
`"nic"`	Normalized coordinates within the inner region
`"nfc"`	Normalized coordinates within the figure region
`"npc"`	Normalized coordinates within the plot region

There are utility functions, `xinch()` and `yinch()`, for performing the inches-to-user coordinates transformation (plus `xyinch()` for converting a location in one step and `cm()` for converting inches to centimeters). More powerful still are the `grconvertX()` and `grconvertY()` functions, which can be used to convert locations between any of the coordinate systems that the base graphics engine recognizes (see Table 3.5).

One problem with performing coordinate transformations like these is that the locations and sizes being drawn have no memory of how they were calculated. They are specified as locations and dimensions in user coordinates. This means that if the graphics window is resized (so that the relationship between physical dimensions and user coordinates changes), the locations and sizes will no longer have their intended meaning. If, in the above example, the graphics window is resized, the ruler will no longer accurately represent centimeter units. This problem will also occur if output is copied from one device to another device that has different physical dimensions. The `legend()` function performs calculations like these when arranging the components of a legend and its output is affected by resizing a device and copying between devices.*