How Android Draws Views

When an Activity receives focus, it will be requested to draw its layout. The Android framework will handle the procedure for drawing, but the Activity must provide the root node of its layout hierarchy.

Drawing begins with the root node of the layout. It is requested to measure and draw the layout tree. Drawing is handled by walking the tree and rendering each View that intersects the invalid region. In turn, each View group is responsible for requesting each of its children to be drawn (with the `draw()` method) and each View is responsible for drawing itself. Because the tree is traversed in-order, this means that parents will be drawn before (i.e., behind) their children, with siblings drawn in the order they appear in the tree.

Drawing the layout is a two pass process: a measure pass and a layout pass. The measuring pass is implemented in `measure(int, int)` and is a top-down traversal of the View tree. Each View pushes dimension specifications down the tree during the recursion. At the end of the measure pass, every View has stored its measurements. The second pass happens in `layout(int, int, int, int)` and is also top-down. During this pass each parent is responsible for positioning all of its children using the sizes computed in the measure pass.

> The framework will not draw Views that are not in the invalid region, and also will take care of drawing the Views background for you.
>
> You can force a View to draw, by calling `invalidate()`.

When a View's `measure()` method returns, its `getMeasuredWidth()` and `getMeasuredHeight()` values must be set, along with those for all of that View's descendants. A View's measured width and measured height values must respect the constraints imposed by the View's parents. This guarantees that at the end of the measure pass, all parents accept all of their children's measurements. A parent View may call `measure()` more than once on its children. For example, the parent may measure each child once with unspecified dimensions to find out how big they want to be, then call `measure()` on them again with actual numbers if the sum of all the children's unconstrained sizes is too big or too small (i.e., if the children don't agree among themselves as to how much space they each get, the parent will intervene and set the rules on the second pass).

The measure pass uses two classes to communicate dimensions. The `View.MeasureSpec` class is used by Views to tell their parents how they want to be measured and positioned. The base LayoutParams class just describes how big the View wants to be for both width and height. For each dimension, it can specify one of:

> To initiate a layout, call `requestLayout()`. This method is typically called by a View on itself when it believes that is can no longer fit within its current bounds.

- an exact number
- *FILL_PARENT*, which means the View wants to be as big as its parent (minus padding)
- *WRAP_CONTENT*, which means that the View wants to be just big enough to enclose its content (plus padding).

There are subclasses of LayoutParams for different subclasses of ViewGroup. For example, RelativeLayout has its own subclass of LayoutParams, which includes the ability to center child Views horizontally and vertically.

MeasureSpecs are used to push requirements down the tree from parent to child. A MeasureSpec can be in one of three modes:

- *UNSPECIFIED*: This is used by a parent to determine the desired dimension of a child View. For example, a LinearLayout may call `measure()` on its child with the height set to *UNSPECIFIED* and a width of *EXACTLY* 240 to find out how tall the child View wants to be given a width of 240 pixels.
- *EXACTLY*: This is used by the parent to impose an exact size on the child. The child must use this size, and guarantee that all of its descendants will fit within this size.
- *AT_MOST*: This is used by the parent to impose a maximum size on the child. The child must guarantee that it and all of its descendants will fit within this size.

← Back to User Interface