

The devNull graphics device

Paul Murrell

November 8, 2004

Introduction

The devNull package implements a graphics device which produces absolutely no graphical output. Furthermore, the device cannot be queried (e.g., to ask for its size, or the size of text on the device).

The main purpose of this device is to create a graphics device object within the graphics engine, mainly so that graphical operations can be recorded on the device (even though they do not produce any output). This is useful for recording what a graphics function does without actually drawing anything. It is used by the grid graphics package (at least).

The secondary purpose of this device is to provide documentation for writing a third-party graphics device for R. This document describes the necessary steps for creating a graphics device and the C-level API for graphics devices. The API is pretty crumbly and inconsistent in places, and is not easily extensible, but at least this provides a description of its current state.

Getting started

The sensible way to organise a third-party graphics device is as a standard add-on package for R. See the “Writing R Extensions” manual for detailed information about how to set up the directories and locate files for developing an R package.

A graphics device will require both R code and C code; this package is called `devNull` so the R code will go in a directory called `devNull/R` and the C code in `devNull/src`.

R code

The R code for a device should be fairly small and straightforward.

First of all, you need a function to start a new device. This typically takes arguments to initialise various aspects of the device, such as size, name (if

it produces output in a file on disk, rather than on-screen), and the default font. There are plenty of examples for the standard R graphics devices; see `postscript()`, `pdf()`, and `x11()`.

For devices which allow multiple fonts, functions should be provided to map the standard font families, `"serif"`, `"sans"`, `"mono"`, and `"symbol"`, to device-specific fonts. See, for example, `postscriptFonts()` and `X11Fonts()`.

The null device has a fixed size, produces no files, and has a single, fixed font, so there are no arguments for the user to specify. The `devNull()` function does, however, perform the most crucial operation of a device function, which is to call C code to create the device and let R know that the device exists.

```
devNull <- function() {
  .Call("do_devNULL", PACKAGE="devNull")
}

.First.lib <- function(lib, pkg) {
  library.dynam("devNull", pkg, lib)
}
```

There is no need to write R code for closing/destroying the device; the R-level interface for this is provided by the `dev.off()` function. You will, however, need to handle finalising the device in C code.

C code

The entry point for C code is the function call made from R to create the device; in the current case, `do_devNull`.

In most cases, there will be arguments to handle — if you have used `.Call` as above, these will be R objects (SEXPs), from which you will need to extract the more familiar C values. See “Writing R Extensions” for information on working with R objects in C code.

The `do_devNull` function below shows the standard pattern for graphics devices. The steps are:

- `R_CheckDeviceAvailable()` makes sure that a device can be created (there is a hard-coded maximum of 64 devices open at one time). It will fail with an error if the device can’t be created.
- A new device structure is allocated. This device structure is defined in `$R_HOME/src/include/R_ext/GraphicsDevice.h`. We will see a lot more of it later.
- the device display list is initialised. The display list is used for redrawing the most recent “page” of output on the device (or for copying that output

to another device). Your device will not have to do anything else to maintain the display list; it is all handled by R's graphics engine.

- the call to `nullDeviceDriver` (which we will look at next) does all of the initialisation of the device. If this fails, we must deallocate the device structure before bailing out with an error.
- if we successfully initialise the device, we finish up by making sure that R knows about the device.

```
#include <Rinternals.h>
#include <Rgraphics.h>
#include <Rdevices.h>
#include <R_ext/GraphicsDevice.h>
#include <R_ext/GraphicsEngine.h>

static Rboolean nullDeviceDriver(NewDevDesc *dev);

SEXP do_devNULL() {
    NewDevDesc *dev = NULL;
    GEDevDesc *dd;

    R_CheckDeviceAvailable();
    if (!(dev = (NewDevDesc *) calloc(1, sizeof(NewDevDesc))))
        return R_NilValue;
    dev->displayList = R_NilValue;
    if (!nullDeviceDriver(dev)) {
        free(dev);
        error("unable to start NULL device");
    }
    gsetVar(install(".Device"), mkString("NULL"), R_NilValue);
    dd = GEcreateDevDesc(dev);
    Rf_addDevice((DevDesc*) dd);
    GEinitDisplayList(dd);
    return R_NilValue;
}
```

The device driver

The most important call in `do_devNull` above is to a function, `nullDeviceDriver`, that initialises the device structure. In most cases, there will be several arguments coming in here from the R code specifying things such as device size and fonts and such, but the most important is the pointer to the device structure, `dev`. This structure has many different elements which must be initialised:

Device-specific information It is possible to store device-specific information in an arbitrary structure attached to the `deviceSpecific` slot. Most devices use this slot to hold a set of “local” device settings (such as current pen colour).

It is important that the device-specific structure is deallocated if this creation function has to bail out with an error at any point. The structure should also be deallocated in the `close` device function (i.e., when the device is destroyed).

Device functions R will call these functions either so that the device can produce appropriate graphical output (e.g., draw a rectangle) or in order to query the device for its size or for font size information. We will look in more detail at these functions later.

Initial graphical settings There are several graphical settings that must be initialised; the code shows some standard values.

Start device The `NULL_Open` device function is not used by R's graphics engine; it is only called in the device driver to do further setting up of the device. There are no firm rules describing whether to put set up code in the driver or the "open" function, but one guideline is to put everything in `NULL_Open` that needs to be undone in `NULL_Close` (e.g., opening/closing a file).

Device physical characteristics These are used to query the device. Some standard values have been filled in and others just chosen arbitrarily (e.g., the device size). The units on several of these values are device-specific

Device capabilities The R graphics engine needs to know what the device is capable of. For example, if the device can't do clipping, R will do its best to clip itself.

The `newDevStruct` slot must be set to 1.

```
static Rboolean nullDeviceDriver(NewDevDesc *dev) {
    dev->deviceSpecific = NULL;
    /*
     * Device functions
     */
    dev->open = NULL_Open;
    dev->close = NULL_Close;
    dev->activate = NULL_Activate;
    dev->deactivate = NULL_Deactivate;
    dev->size = NULL_Size;
    dev->newPage = NULL_NewPage;
    dev->clip = NULL_Clip;
    dev->strWidth = NULL_StrWidth;
    dev->text = NULL_Text;
    dev->rect = NULL_Rect;
    dev->circle = NULL_Circle;
    dev->line = NULL_Line;
    dev->polyline = NULL_Polyline;
    dev->polygon = NULL_Polygon;
    dev->locator = NULL_Locator;
    dev->mode = NULL_Mode;
```

```

dev->hold = NULL_Hold;
dev->metricInfo = NULL_MetricInfo;
/*
 * Initial graphical settings
 */
dev->startfont = 1;
dev->starttps = 10;
dev->startcol = R_RGB(0, 0, 0);
dev->startfill = R_TRANWHITE;
dev->startlty = LTY_SOLID;
dev->startgamma = 1;
/*
 * Start device
 */
if(!NULL_Open(dev)) {
    return FALSE;
}
/*
 * Device physical characteristics
 */
dev->left = 0;
dev->right = 1000;
dev->bottom = 0;
dev->top = 1000;
dev->cra[0] = 10;
dev->cra[1] = 10;
dev->xCharOffset = 0.4900;
dev->yCharOffset = 0.3333;
dev->yLineBias = 0.1;
dev->ipr[0] = 1.0/72;
dev->ipr[1] = 1.0/72;
/*
 * Device capabilities
 */
dev->canResizePlot= FALSE;
dev->canChangeFont= FALSE;
dev->canRotateText= TRUE;
dev->canResizeText= TRUE;
dev->canClip = TRUE;
dev->canHAdj = 2;
dev->canChangeGamma = FALSE;
dev->displayListOn = TRUE;

dev->newDevStruct = 1;
return TRUE;
}

```

Device functions

All of the null graphics device functions will do nothing. Some comments are provided for what could be done in other cases and, in a couple of cases, it is important how nothing gets done. The null device functions also show what parameters the graphics engine will call these functions with. The file `$R_HOME/src/include/R_ext/GraphicsDevice.h` contains more detailed information about what each function should do.

Graphical output The majority of the device functions are concerned with producing graphical output (rectangles, lines, text). The `R_GE_gcontext` points to a structure containing graphical parameter settings (line width, colours, fonts). The file `$R_HOME/src/include/R_ext/GraphicsEngine.h` contains a definition of this structure.

```
static void NULL_Circle(double x, double y, double r,
                        R_GE_gcontext *gc,
                        NewDevDesc *dev) {
}
static void NULL_Line(double x1, double y1, double x2, double y2,
                      R_GE_gcontext *gc,
                      NewDevDesc *dev) {
}
static void NULL_Polygon(int n, double *x, double *y,
                         R_GE_gcontext *gc,
                         NewDevDesc *dev) {
}
static void NULL_Polyline(int n, double *x, double *y,
                           R_GE_gcontext *gc,
                           NewDevDesc *dev) {
}
static void NULL_Rect(double x0, double y0, double x1, double y1,
                      R_GE_gcontext *gc,
                      NewDevDesc *dev) {
}
static void NULL_Text(double x, double y, char *str,
                      double rot, double hadj,
                      R_GE_gcontext *gc,
                      NewDevDesc *dev) {
}
static void NULL_NewPage(R_GE_gcontext *gc,
                         NewDevDesc *dev) {
}
```

Controlling the device `NULL_Close` should undo anything that `NULL_Open` did, especially any memory allocation. As mentioned, `NULL_Open` is not called by the graphics engine.

```
static void NULL_Close(NewDevDesc *dev) {
```

```

}
Rboolean NULL_Open(NewDevDesc *dev) {
    return TRUE;
}

```

Device state Several functions provide a way to affect the current state of the device.

```

static void NULL_Activate(NewDevDesc *dev) {
}
static void NULL_Clip(double x0, double x1, double y0, double y1,
    NewDevDesc *dev) {
}
static void NULL_Deactivate(NewDevDesc *dev) {
}
static void NULL_Mode(int mode, NewDevDesc *dev) {
}

```

Querying the device These functions provide a way for the graphics engine to query the device for information. If the device does not support font metric information, return 0 for all values. `NULL_Locator` is a crude, modal way of obtaining a mouse-click from a device; return `FALSE` if the device does not support mouse interaction.

```

static Rboolean NULL_Locator(double *x, double *y, NewDevDesc *dev) {
    return FALSE;
}
static void NULL_MetricInfo(int c,
    R_GE_gcontext *gc,
    double* ascent, double* descent,
    double* width, NewDevDesc *dev) {
    *ascent = 0.0;
    *descent = 0.0;
    *width = 0.0;
}
static void NULL_Size(double *left, double *right,
    double *bottom, double *top,
    NewDevDesc *dev) {
    *left = dev->left;
    *right = dev->right;
    *bottom = dev->bottom;
    *top = dev->top;
}
static double NULL_StrWidth(char *str,
    R_GE_gcontext *gc,
    NewDevDesc *dev) {
    return 0.0;
}

```

I still don't know what `NULL_dot` and `NULL_Hold` are (or were) for.

```
static void NULL_dot(NewDevDesc *dev) {  
}  
static void NULL_Hold(NewDevDesc *dev) {  
}
```