Creating plasmoids with JavaScript

Plasma, Baby

With KDE 4.4, plasmoids can now be written in JavaScript or QtScript, thus opening up a whole new class of applications. Marcel shows how easy it is to build JavaScript plasmoids.

By Marcel Gagné

As a non-programmer, I have an insane amount of respect for programmers. Part of it is because I'm married to a programmer (insert appropriate smiley here), but part of it is because they seem to have this superhuman ability to understand strange languages and turn them into things I either need or want to have. Programmers and coders, specifically those who create free and open source software, are responsible for the amazing Linux system I am using to write this. So, my thanks to you all.

Tens of thousands of programs and packages are available for Linux, and, granted, I use relatively few of these. Given the sheer number of men and women out there who toil away at coding, you would think that the odds would be pretty good that anything you or I might want already exists. Of course, that isn't true, and that's why somebody is always coming up with cool new stuff. Bigger doesn't always translate into better, however, and therein lies the big secret. An application does not have to be huge and complex to be useful, which is why KDE widgets, or plasmoids, are so much more than just desktop eye candy. They're a peek into the very future of computing, a future you can take part in. Even if you code like me.

Plasmoids

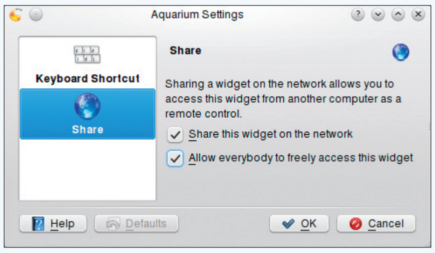
But don't just stop at the desktop. With KDE 4.4, plasmoids [1] can now be written in JavaScript [2], or QtScript if you prefer, thereby opening the door to a whole new class of applications. And devices.

Back in Issue 114 of *Linux Magazine*, Johan Thelin demonstrated the framework for building KDE plasmoids in C++, which is fine for superhuman programmers, but for people with only a modicum of programming ability, learning C++ seems a bit beyond reach. That said, even casual non-programmers like myself write the occasional shell script and dabble in simple coding languages like JavaScript.

The beauty of JavaScript is that it is easy to write, is safe to execute, and can be run anywhere. Take note of that word, "anywhere." This sits at the heart of the KDE strat-

AUTHOR

Marcel is an award-winning columnist, book author, public speaker, and radio and television personality and a well-known voice in the Linux and open source universe. He's also a published science fiction author and editor, a private pilot, and was once a Top 40 disc jockey.


Aquarium Settings

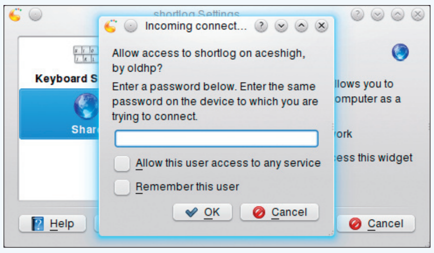
Keyboard Shortcut
Share

Share

Sharing a widget on the network allows you to access this widget from another computer as a remote control.

☑ Share this widget on the network
☑ Allow everybody to freely access this widget

Figure 1: Like that desktop widget? Want to share it?

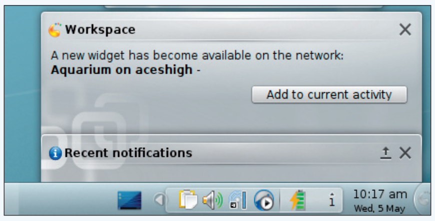

Incoming connect...

Allow access to shortlog on aceshigh, by oldhp?

Enter a password below. Enter the same password on the device to which you are trying to connect.

☐ Allow this user access to any service
☐ Remember this user

Figure 2: Plasmoids may be shared freely or protected by password.


Workspace

A new widget has become available on the network:
Aquarium on aceshigh -

Figure 3: When you choose to share a plasmoid on the network, other workstations will be notified.

egy to find a wide audience and deliver applications across the device spectrum: that means desktop computers, notebooks and netbooks, tablets, ebook readers, smartphones, televisions … in essence, the entire device spectrum!

Imagine walking into a convention with your smartphone or tablet at your hip. The network hosts a number of QtScript plasmoids, shared and free to access. One of these is an interactive program schedule, in which you can specify the things that interest you and be notified before the events start. Your Plasma device loads up the application, giving you the tools you need in the environment where it's needed. Best of all, the application server, which could be any number of devices, doesn't care what you are running either.

I'm going to start by showing you a rather cool desktop trick, after which I'll show how easy it is to create JavaScript plasmoids. Hey, if I can write one, so can you. To begin, take a plasmoid that resides on your desktop activity. If you right-click on the plasmoid (or widget), it offers you a chance to change its settings. What those settings are and how many settings can be affected vary with each plasmoid. One of those settings will be *Share* (Figure 1).

The first thing to do is check the box labeled *Share this widget on the network*. If you click *OK* here without checking the second box (*Allow everybody to freely access this widget*), you'll get a password request when someone first tries to add the widget to their desktop (Figure 2). After you enter a password, the remote user will have the opportunity to enter theirs, which allows them to add that widget to their desktop.

As you can see, this dialog has two checkboxes. One allows a user access to any service you are offering. The second remembers that choice beyond the current logged-in session.

When you do decide to share a plasmoid on the network, it pops up an alert in the messaging area of any workstation on the network stating that there is a new widget available for use (Figure 3). If you click on the button provided to get it and it has been password or PIN protected, you'll be asked for the credentials. When you enter the information, it will simply appear on your desktop.

What makes all of this particularly cool is that you can put JavaScript plasmoids on other devices that use Plasma workspaces. That includes netbooks or anything that uses the new Plasma Mobile shell, like certain smartphones. Also, you can use the Plasma Windowed app (which runs Plasmoids as full-screen apps on such devices). The Plasma Media Center [3] (currently in development) will similarly provide support for local and remote widgets, meaning you could add these widgets to your television. It's all part of the infrastructure.

In fact, the current Plasma mobile roadmap includes the ability to share widgets with your IM contacts over Telepathy "tubes" (akin to HTML5 Storage). There's also a dialog manager so when a Plasmoid pops up a window, such as for configuration, it can be handled properly for whatever device is calling it. On the desktop, it just shows a simple window as always, but on netbooks and mobile devices, it integrates the dialog with the full-screen interface.

Another item on the roadmap this year involves implementing cryptographic web-of-trust verification of payloads delivered over the network so you can confirm that, yes, the widget from your television was actually written by Big T.V. Manufacturer or that it really did come from my desktop to yours.

Now that you understand the power and potential of these little applications, let's get started.

Building a JavaScript Plasmoid

Plasmoids come packaged in a single archive file with a `.plasmoid` extension. That archive includes the code itself, images, configurations, etc. – whatever is required for the complete ap-

PLASMA MOBILE

Enter *plasma* and *mobile* in the YouTube search box for a handful of videos created to demonstrate the Plasma mobile shell on a variety of devices (e.g., Nokia N900). Very cool stuff.

plication to do what it needs to do. My demonstration plasmoid will be called *hello-linuxpro*, so I'll start by creating a directory by that name. In that directory is another directory called `contents` and another below that called `code`.

```
mkdir -p hello-linuxpro/contents/code
```

Truth be told, you don't need anything more than a `contents` directory, but common coding convention separates the various bits and pieces that make up the plasmoid to make it more readable and easier to work with. And, it looks cleaner.

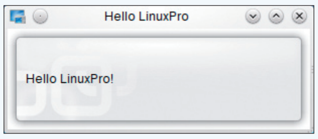


Figure 4: **A simple plasmoid that does nothing but display a friendly message.**

In the top-level directory (`hello-linuxpro`), create a file called `metadata.desktop`. Using your favorite editor (mine is still Vi, or Vim), enter the information as shown in Listing 1.

Much of what you find here makes sense. Besides being necessary (Plasma needs it to load the plasmoid), it lists some useful information about the plasmoid: what it's called, who wrote it, and so on. Name and Comment both show up in the Add Widget dialog. `Icon` can be something supplied with the plasmoid (in which case you would add the path) or just one of the standard KDE icons – I used KSnapshot's icon.

The next four lines, under `Type=Service`, tell Plasma what this thing is, what to do with it, and where to find the main code. The path specified at `X-Plasma-MainScript` is relative to the `contents` directory that you created at the beginning. It also specifies the initial display size of the widget as it appears on your desktop.

Take note of the `X-KDE-PluginInfo-Category`; you could call this whatever you like, but if you plan on releasing it to the community, the value there must be one of the category names found at `Projects/Plasma/PIG` [4]. Pretty much everything else just makes up the About information for the plasmoid.

And Now, the Code!

This widget will be really simple. In your favorite editor, create the main script as you defined it in the `metadata.desktop` definition, in this case `code/main.js` under `contents`. My script contains the following lines:

```
layout = new LinearLayout(plasmoid);

label = new Label();
layout.addItem(label);

label.text = 'Hello LinuxPro!';
```

I'm keeping this very simple, but I promise to give you pointers to the plasmoid QtScript API a little later in this column. For now, I'll use only a few basics. The word `plasmoid` is a global variable, and it represents your widget. Although it can be a lot more than just what's here, you'll be able to explore that in the API specs.

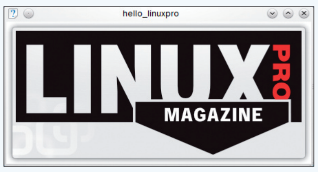


Figure 5: **Yes, it's the *Linux Pro Magazine* logo, displayed in a plasmoid.**

To begin, create the layout by passing `plasmoid` to `LinearLayout`. This attaches the layout to the widget. Also, I added a label to the layout and set the label text.

That is all there is to it. That said, it is probably premature to package this plasmoid, so I will test it first. To do that, I will use a little program called `plasmoidviewer`. Here's how it works:

```
plasmoidviewer /path/to/hello-linuxpro
```

The result looks like Figure 4.

Now I'll try a different example – something a little more interesting. This time, I'll load an SVG image file and display it.

LISTING 1: metadata.desktop

```
[Desktop Entry]
Name=Hello LinuxPro
Comment=My first plasmoid
Icon=ksnapshot

Type=Service
X-KDE-ServiceTypes=Plasma/Applet
X-Plasma-API=javascript
X-Plasma-MainScript=code/main.js
X-Plasma-DefaultSize=300,100

X-KDE-PluginInfo-Author=Marcel Gagne
X-KDE-PluginInfo-Email=marcel@marcelgagne.com
X-KDE-PluginInfo-Name=hello-linuxpro
X-KDE-PluginInfo-Version=1.0
X-KDE-PluginInfo-Website=http://plasma.kde.org/
X-KDE-PluginInfo-Category=Examples
X-KDE-PluginInfo-Depends=
X-KDE-PluginInfo-License=GPL
X-KDE-PluginInfo-EnabledByDefault=true
```

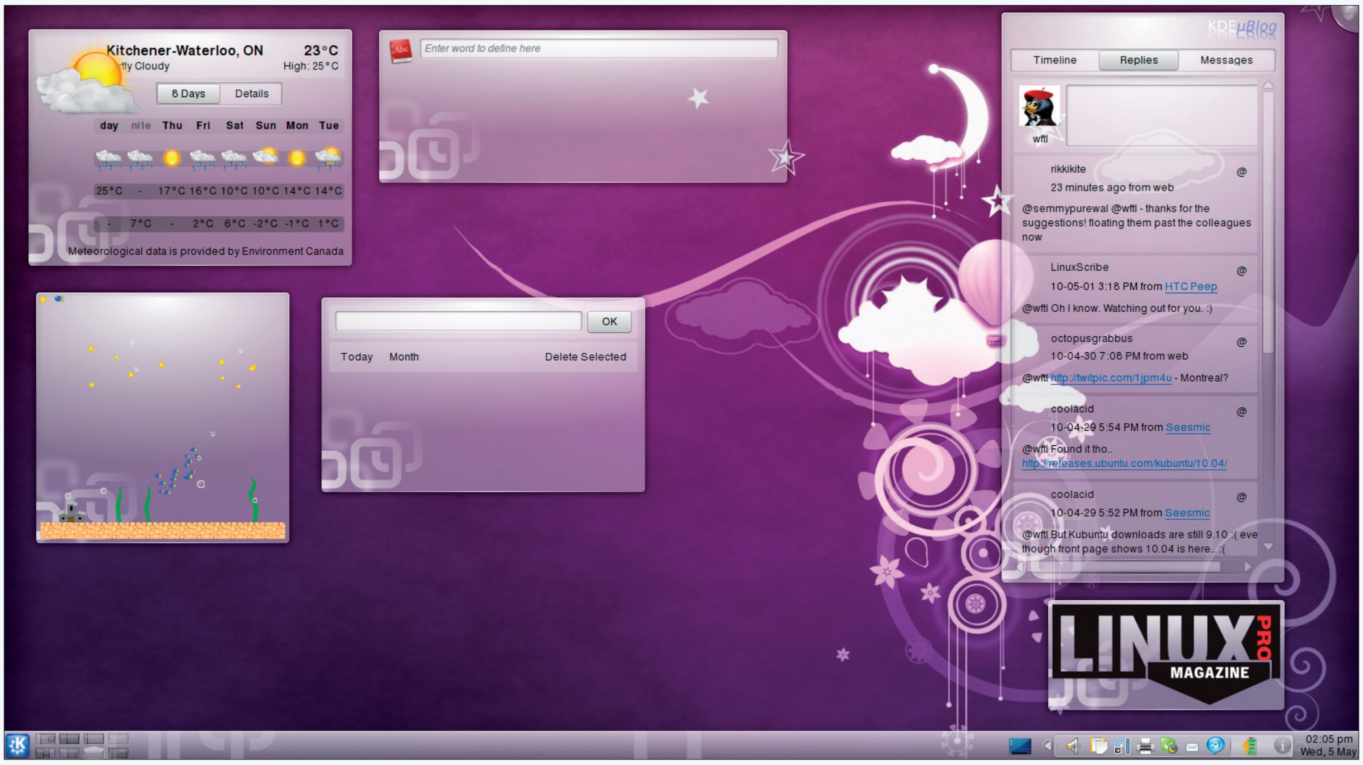
LISTING 2: Working with an SVG File

```
mainLayout = new LinearLayout(plasmoid);

svg = new PlasmaSvg('linuxpro_logo')

plasmoid.action_myAction = function()
{
    plasmoid.update()
}

plasmoid.paintInterface = function(painter)
{
    rect = plasmoid.rect
    svg.resize(rect.width, rect.height)
    svg.paint(painter, rect.x, rect.y)
}
```


Kitchener-Waterloo, ON
23°C
rtly Cloudy
High: 25°C
8 Days
Details
day nite Thu Fri Sat Sun Mon Tue
25°C - 17°C 16°C 10°C 10°C 14°C 14°C
- 7°C - 2°C 6°C -2°C -1°C 1°C
Meteorological data is provided by Environment Canada
Enter word to define here
OK
Today Month Delete Selected
KDE µBlog
Timeline
Replies
Messages
LINUX PRO MAGAZINE
02:05 pm
Wed, 5 May

wftl

rikkikite
23 minutes ago from web
@semmypurewal @wftl - thanks for the suggestions! floating them past the colleagues now

LinuxScribe
10-05-01 3:18 PM from HTC Peep
@wftl Oh I know. Watching out for you. :)

octopusgrabbus
10-04-30 7:08 PM from web
@wftl http://twitpic.com/1jpm4u - Montreal?

coolacid
10-04-29 5:54 PM from Seesmic
@wftl Found it tho..
http://releases.ubuntu.com/kubuntu/10.04/

coolacid
10-04-29 5:52 PM from Seesmic
@wftl But Kubuntu downloads are still 9.10 :(eve
though front page shows 10.04 is here.. :(

Figure 6: The *Linux Pro Magazine* logo now sits proudly on my desktop, just above the system tray and clock.

In this example, I'll leave the `metadata.desktop` file out and only show you the code (Listing 2).

The result of this, seen with the plasmoidviewer program, is shown in Figure 5.

When you are happy with the function of your plasmoid, it's time to install it, package it, or both. To install the widget so you can use it on your desktop, use the `plasmapkg` command with the `-i` flag:

```
plasmapkg -i /path/to/your-plasmoid
```

If everything goes well, you should see a message saying *Successfully installed /path/to/your-plasmoid*. Now you can add your widget to the desktop with the *Add Widgets* dialog. Right-click on the desktop or click the cashew in the top right-hand corner. If the desktop is currently locked down, you might have to click *Unlock Widgets* first. The new plasmoid will appear in the list and on your desktop when you click on it. My new logo plasmoid now sits just above my system tray as in Figure 6.

Should you be so happy with your work that you just have to share it with the world, it is now time to package it. This is simply a matter of zipping up the file and renaming it `something.plasmoid`:

```
cd hello-linuxpro
zip -r hello-linuxpro.zip .
mv hello-linuxpro.zip hello-linuxpro.plasmoid
```

The name you would use for the plasmoid must be the same name you used in the `metadata.desktop` file under `X-KDE-PluginName`. To install this as a package, you can use the same command as earlier. Instead of reading a directory, `plasmapkg` will read the package file:

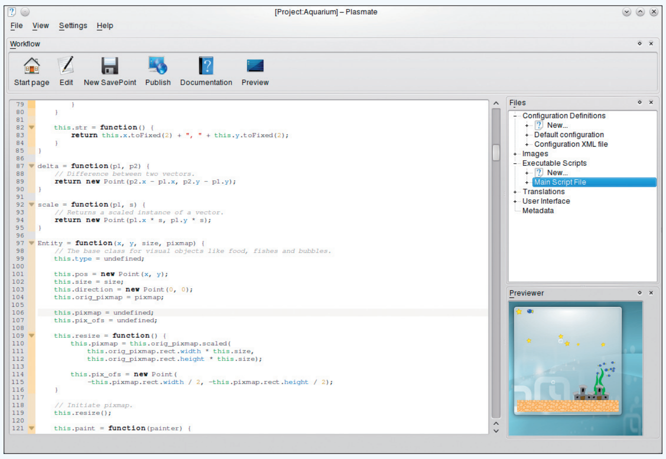
```
plasmapkg -i hello-linuxpro.plasmoid
```

That's it!

PlasMate

I've given you some very simple examples here, but if you want to get into some slightly more serious Plasmoid development, you will want to use PlasMate, a rather cool little tool that is in the early stages of development. PlasMate [5] is essentially an integrated development environment for writing plasmoids. It works with JavaScript, Ruby, and Python plasmoids (Figure 7).

PlasMate takes care of the entire project development cycle for you – from creating the structure for new projects, to handling revisions, to testing, packaging, and even publishing your plasmoids. It has a built-in previewer (so you don't have to run



```
            }
        }

        this.str = function() {
            return this.x.toFixed(2) + ", " + this.y.toFixed(2);
        }
}

delta = function(p1, p2) {
    // Difference between two vectors.
    return new Point(p2.x - p1.x, p2.y - p1.y);
}

scale = function(p1, s) {
    // Returns a scaled instance of a vector.
    return new Point(p1.x * s, p1.y * s);
}

Entity = function(x, y, size, pixmap) {
    // The base class for visual objects like food, fishes and bubbles.
    this.type = undefined;

    this.pos = new Point(x, y);
    this.size = size;
    this.direction = new Point(0, 0);
    this.orig_pixmap = pixmap;

    this.pixmap = undefined;
    this.pix_ofs = undefined;

    this.resize = function() {
        this.pixmap = this.orig_pixmap.scaled(
            this.orig_pixmap.rect.width * this.size,
            this.orig_pixmap.rect.height * this.size);

        this.pix_ofs = new Point(
            -this.pixmap.rect.width / 2, -this.pixmap.rect.height / 2);
    }

    // Initiate pixmap.
    this.resize();

    this.paint = function(painter) {
```

Figure 7: PlasMate is an integrated development environment for creating a variety of KDE plasmoids.

plasmoidviewer externally), and it has links to the QtScript API in its documentation, so you don't have to go looking for it.

The only catch with PlasMate is that it is still very early in development. Nevertheless, it is an extremely handy and useful package. If you are going to do any plasmoid development, I highly recommend that you check it out. Literally. To build `plasmate`, you'll need some core development tools, such as compilers, CMake, and so on. And, of course, you'll need the KDE development libraries. On my Kubuntu test system, I managed that with the following command:

```
sudo apt-get install build-essentials cmake kde-devel
```

Once all this was installed, I downloaded (or checked out) the latest PlasMate source from its anonymous SVN repository:

```
svn co svn://anonsvn.kde.org/home/kde/trunk/playground/base/
           plasma/plasmate
```

This created a folder called `plasmate`. All that remained was to build the package:

```
cd plasmate
cmake .
sudo make install
```

Building the package only took a few minutes, and installation went through without a hitch, so it looks pretty good. Once you start PlasMate, you have the option of opening an existing

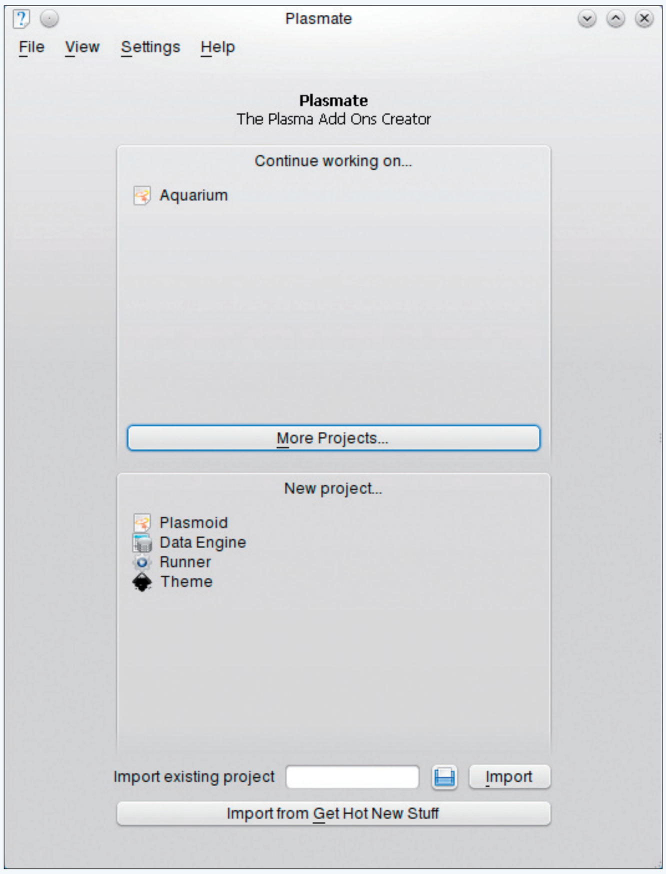
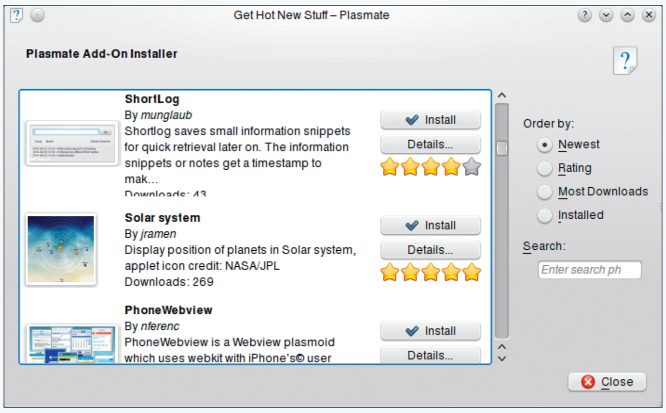


Figure 8: PlasMate handles the creation of new projects and can import from existing plasmoids.



ShortLog
By munglaub
Shortlog saves small information snippets for quick retrieval later on. The information snippets or notes get a timestamp to mak...
Downloads: 43

Solar system
By jramen
Display position of planets in Solar system, applet icon credit: NASA/JPL
Downloads: 269

PhoneWebview
By nferenc
PhoneWebview is a Webview plasmoid which uses webkit with iPhone's© user

Figure 9: What better starting point for learning the ropes than something that already exists and happens to be hot.

project or creating something entirely new from the "Home" screen (Figure 8). PlasMate handles the creation of plasmoids, data engines, runners, and themes.

Although JavaScript might be easier to work with than other languages, not everyone will just jump in and start coding. If you are just starting out and would like a little help, a rather interesting little button at the bottom of the PlasMate start screen says *Import from Get Hot New Stuff*. The best teacher is often someone else's work. Open source software is great for many reasons, one of which is that you get to see what others have done before you and, using the source code that is available, you have a great base from which to learn by modifying the code and adding your own touches.

When you click the *Import from Get Hot New Stuff* button, and you'll see the PlasMate Add-On Installer (Figure 9).

Scroll down the list, find something that interests you, and click *Install*. The package will be imported into PlasMate, into the appropriate structure. Now you are ready to start exploring, tweaking, and just plain old hacking. Or, to put it another way – have fun.

Looking Forward

The QtScript/JavaScript framework that is part of the new KDE is an amazing piece of forward-looking code wrapped up in seemingly playful eye candy. The next generation of plasmoids and widgets is already set to power a vast array of network-connected devices, from smartphone to televisions. It's also an exciting step into a world where applications are location sensitive and available on demand.

The relative simplicity and inherent security and portability of JavaScript makes this an exciting arena for aspiring coders, including those of us who just dabble. ■■■

INFO

[1] Plasma: *http://plasma.kde.org*

[2] Plasma JavaScript API: *http://techbase.kde.org/index.php?title=Development/Tutorials/Plasma/JavaScript/API*

[3] Plasma Media Center: *http://techbase.kde.org/Projects/Plasma/Plasma_Media_Center*

[4] Plasma interface guidelines: *http://techbase.kde.org/Projects/Plasma/PIG*

[5] PlasMate: *http://techbase.kde.org/Projects/Plasma/PlasMate*