IBM

developerWorks.

Introducing KDE 4 plasmoids

Plasma, the desktop shell of KDE 4, improves the desktop experience with its simple Plasma applets, plasmoids

Skill Level: Intermediate

Martyn Honeyford (ibmmartyn@gmail.com)
Senior Java Developer

14 Jun 2009

KDE 4 includes many exciting new technologies, including Plasma, a feature that forms the desktop shell of KDE 4. See how to write simple Plasma applets (known as *plasmoids*) to greatly improve the desktop experience and how to turn a plasmoid into a simple memory monitor.

KDE, the K Desktop Environment, is a free software project based on a desktop environment for UNIX®-like systems. It provides basic desktop functions and applications for daily needs, as well as tools and documentation for developers to write standalone applications for the system. KDE software is based on the Qt toolkit. The centerpiece of the fourth series of KDE is a redesigned desktop and panels, collectively called Plasma, which integrates the functionality of Kicker, KDesktop, and SuperKaramba into one piece of technology.

KDE 4 offers new hope

The fourth major version of the K Desktop Environment (KDE) was released in January 2008 to mixed reactions. The release marked a major undertaking:

- Much of the KDE 3 codebase was completely rewritten.
- A large number of new technologies were created.
- Most of the codebase was refactored in some way, shape, or form.

© Copyright IBM Corporation 2009. All rights reserved.

While the initial 4.0 release showed great potential, the scope of the changes meant that the release would not achieve feature parity with KDE 3. Therefore, it was widely reported as being unstable in certain areas, and some users were reluctant to make the switch immediately. Many of these shortcomings were addressed with the 4.1 release, but there were still notable feature omissions.

With the January 2009 release of KDE 4.2, however, most of these concerns were addressed, and many users decided to start using 4.2 as their main desktop environment.

Now is a good time to look at these new features and technologies in depth, since so many users have access to them.

KDE 4.2 revives the desktop

This article concentrates on one of the most exciting new technologies: Plasma and Plasma applets (which are often referred to as *plasmoids*).

The idea behind Plasma is simple: You have one or more Plasma *containments*, which are displayable elements capable of containing individual items such as widgets/plasmoids and/or other Plasma containments.

This concept doesn't sound particularly ground-breaking until you realize that under KDE 4, the entire desktop shell (the main desktop that is presented to the user) is in fact a Plasma containment, and all of the familiar controls such as the task bar, task list, clock, task switcher, k Menu, quick-launch icons, etc., are all implemented either as Plasma applets (like the clock) or Plasma containments (the task bar).

This is extremely exciting. It gives KDE 4 users amazing potential for customization, allowing developers and/or individuals to change almost any aspect of the desktop experience, completely altering its behavior where necessary. For instance, if you want a very powerful application launcher with more capability that the default one, use a launcher such as Lancelot, which is currently in development. If space is scarce, write an application launcher that is more like those seen in classic lightweight DEs (such as XFCE) where a simple pop-up is displayed with a right click, and remove the regular launcher to provide more screen real estate.

This simple idea should allow KDE 4 to grow into a one-size-fits-all desktop, because distributions and users can adapt KDE to diverse configurations. For example, high-powered machines can benefit from large, complex applets and all the extras , while machines with less horsepower can rely on simpler/lightweight versions of the required components and remove unnecessary components. Thus, you no longer need to replace KDE 4 in favor of a more specialized, lightweight DE.

Although these Plasma applets can be somewhat complicated, they are intended to

© Copyright IBM Corporation 2009. All rights reserved.

be small, utility-style applications that you would want open all the time rather than full applications. They are typically embedded in your desktop and/or taskbar; some classic examples are system monitors, instant messengers, social networking tools, etc.

Another common design feature is their relatively small use of screen real estate. You don't want many plasmoids open on the desktop at all times without overlapping. They don't appear in the window manager as separate windows, so you can't quickly move between them or minimize them.

In addition to the basic Plasma containment technology, the KDE developers have also gone out of their way to make writing powerful Plasma applets as easy and flexible as possible. Features include:

- **First-class support for scalar vector graphics (SVG) for all GUI elements.** All applications/applets can thus be resolution-independent with smooth scaling. The same application will look good and behave well on both a 30-inch high-definition monitor and an 8-inch netbook screen—with no additional work required by the developer.
- **Excellent support for theming.** Developers are actively encouraged to reference all resources such as icons, backgrounds, etc., using relative paths so that KDE can take care of locating these at runtime based on the currently selected theme. This should allow all KDE 4 apps to take advantage of a consistent look across different themes, again with no additional work for the developer.
- **Multiple language support.** KDE 4 plasmoids (like KDE apps in general) can be written in a number of different programming languages. This article focuses on C++, but other languages are in development including Ruby, Python, and Javascript.
- **Component reuse.** KDE has excellent support for allowing components to reuse the services provided by each other.

Now that you have a bit of background on Plasma, let's get our hands dirty by writing a simple plasmoid to add to our KDE 4.2 desktop.

The KDevelop 4 IDE

These instructions assume that you are running a KDE 4.2 desktop. Some or all of the instructions may be applicable to KDE 4.1, but I haven't confirmed this.

First things first: You need to install all of the KDE development libraries, headers, etc.

© Copyright IBM Corporation 2009. All rights reserved.

I am using the current stable release of Kubuntu 8.10 (Intrepid Ibex), which ships with KDE 4.1 by default, so I had to enable the PPA repositories to get KDE 4.2 and headers (see Resources for details). This step is not necessary for users of Kubuntu 9.04 (but for installing g++ and kdesdk, etc., it is necessary). Note that this version of KDE is not currently supported by Canonical, so it's best to use a development environment rather than your main machine.

Once the PPA repository was added, I simply installed the kdesdk, g++, cmake, and some kde-dev packages, like so:

Listing 1. Building the KDE development environment

```
sudo apt-get update
sudo apt-get install kdesdk
install cmake
sudo apt-get install g++
sudo apt-get install libphonon-dev libplasma-dev
```

This pulled in everything I needed on my machine. If you're using another distribution, you'll need to determine how to install these packages.

KDevelop 4 has a nice template to automatically generate a basic plasmoid. Unfortunately, at the time of writing, KDevelop 4 is still in beta, and there is no binary release available for Intrepid, so in order to go down this route, you need to build it from source (see Resources for instructions).

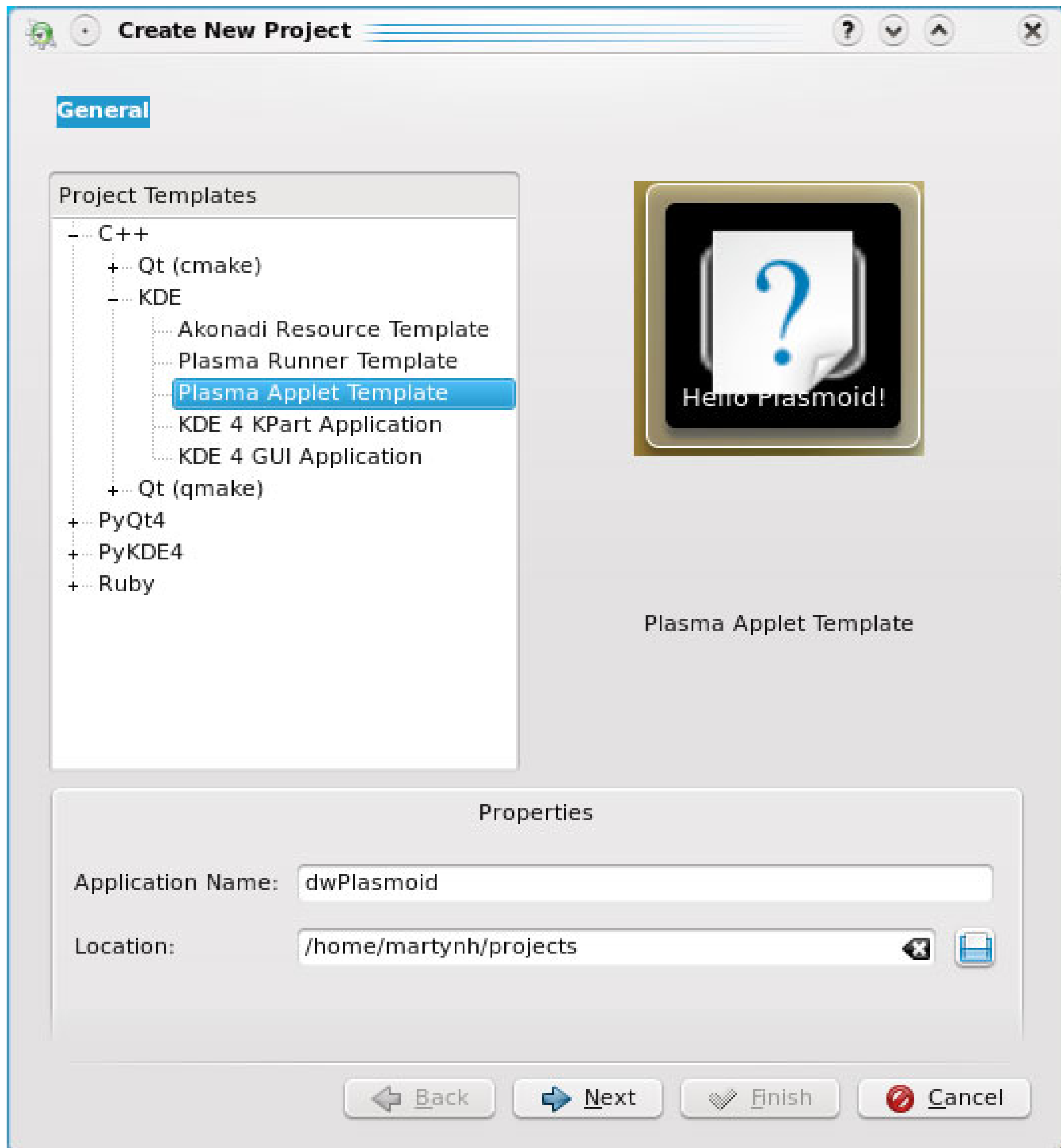
When KDevelop is not your choice

If you do not wish to use KDevelop, I recommend the excellent KDE techbase tutorial "Getting Started" (see Resources for a link). If you follow those steps, you should wind up with a very similar project to the one we are creating with KDevelop, and you can skip to the section "Modifying the sample."

Once you have KDevelop running as shown in Figure 1, open the **Project** menu, and select **New From Template**. When the dialog appears, expand the **C++**, then **KDE** branches and select **Plasma Applet Template**. Next fill in the **Application name** (I suggest dwPlasmoid). Then click **Next** and **Finish**.

Figure 1. Project creation

© Copyright IBM Corporation 2009. All rights reserved.



The final dialog box asks you to choose an **Installation Prefix** directory (I recommend *$HOME/plasmoids*) and a build type (choose *Release* or *Debug*).

This should create the project. Take some time to go through the well-documented .cpp and .h files. In addition, the "Getting started" tutorial (see Resources for a link) has further explanations about what the code is doing.

Now that you have created the project, you're ready to build it. Do this by opening the **Project** and choosing **build** (or by pressing F8). The first time through, this will

© Copyright IBM Corporation 2009. All rights reserved.

call `configure`, and after that, it will just build.

Assuming the build was successful, select **Install** from the **Project** menu. This copies the installation files to the directory you chose previously. If this completed without incident, you are ready to test.

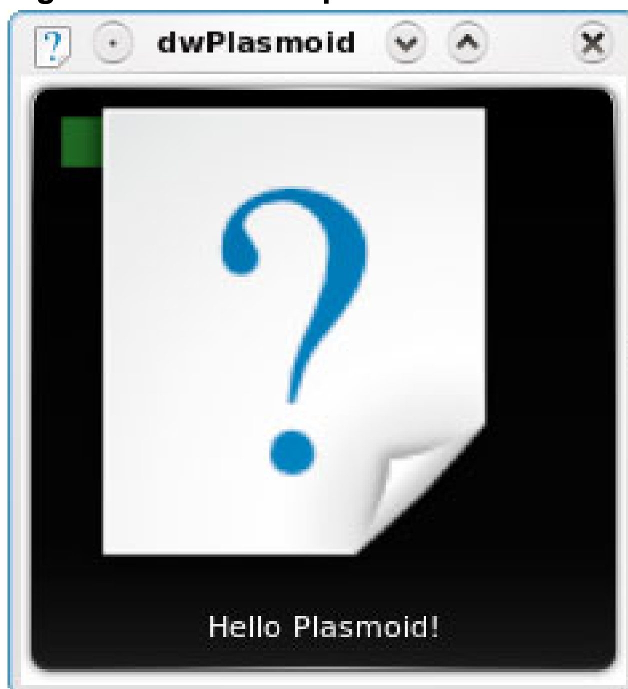
Before you can run the new plasmoid, you need to tell KDE of its existence. First, you need to make KDE aware of the directory where the plasmoid is installed ($HOME/plasmoids). Do this by opening a "konsole" to set the `KDEDIRS` environment variable and then by running the `kbuildsyscocoa4` command. Open a terminal and type the lines in Listing 2:

Listing 2. Making KDE aware of the plasmoid location

```
export KDEDIRS=$KDEDIRS:$HOME/plasmoids
kbuildsyscocoa4
```

Now that KDE knows about the plasmoid, you can attempt to run it. `plasmoidviewer` is a handy utility that you can use for testing purposes. Get it by typing `plasmoidviewer dwplasmoid`. You should see the sample plasmoid displayed in a window as shown in Figure 2:

Figure 2. Your basic plasmoid



Since this is a plasmoid, you can also embed it in the desktop. However, if you try to add the plasmoid now (using the **Add widgets** menu item on the desktop), you'll see *dwPlasmoid* in the list of plasmoids, but if you try to add it, it will not work! While

© Copyright IBM Corporation 2009. All rights reserved.

KDEDIRS was set correctly when you ran kbuildsyscocoa4, it was not set correctly in the environment when the Plasma desktop was started, and so it cannot find all of the necessary files.

You can fix that by quitting the current Plasma instance that's running and restarting it from the command line (use the one we used earlier so that KDEDIRS is still set correctly; otherwise, reset it before running the following commands). Note that this will gracefully end Plasma and restart it, so be sure you aren't doing anything that will be disrupted by this (for example, save any unsaved work in any other plasmoids). You quit the current Plasma instance as shown in Listing 3.

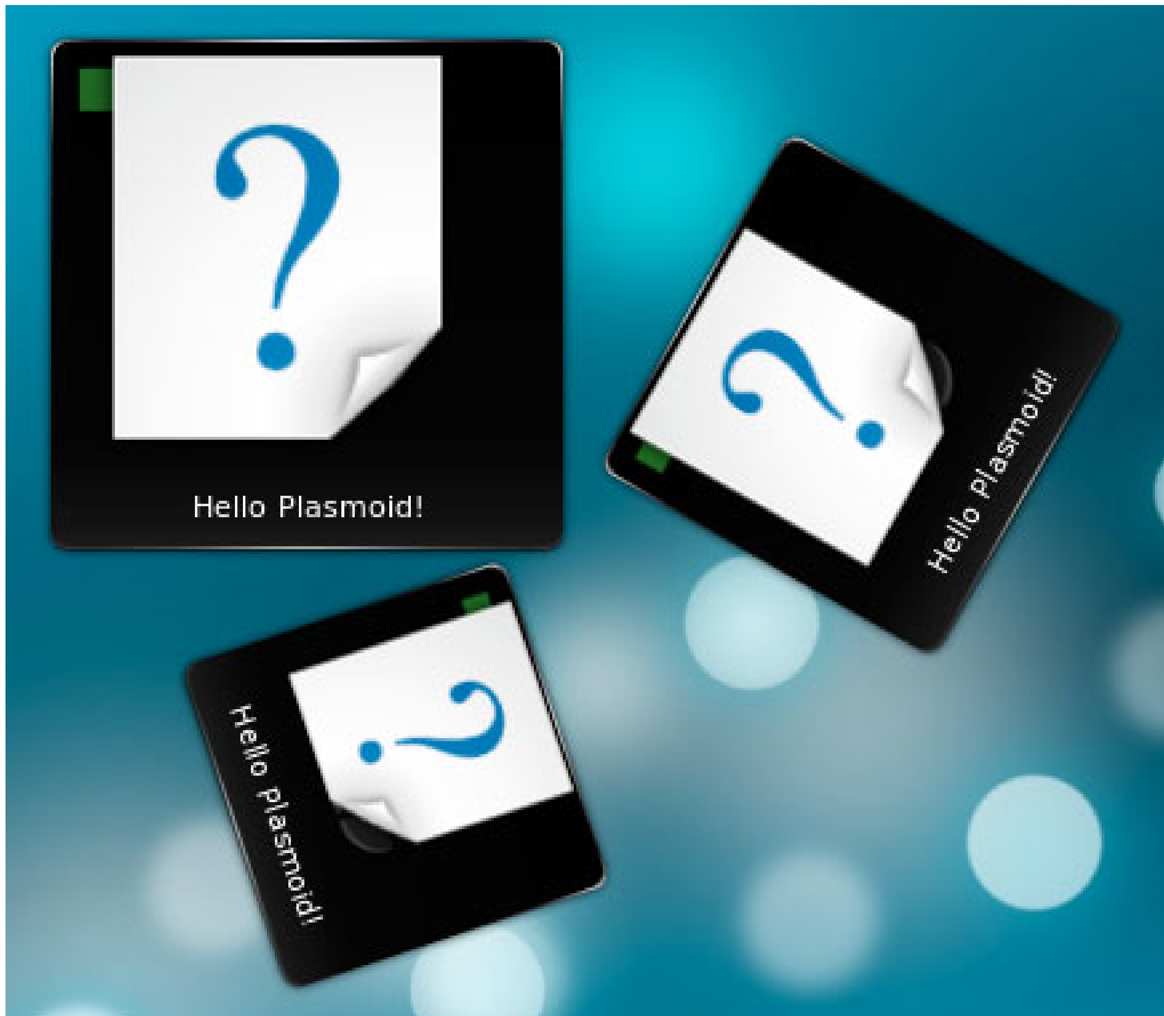
Listing 3. Quitting the current Plasma instance

```
kquitapp plasma && kstart plasma
```

You should now be able to add the plasmoid to the desktop, and it will sit there displaying its simple icon and message all day long. You can even add three of them and take advantage of the built-in scaling and rotation features as shown in Figure 3:

Figure 3. Multiple instances of plasmoid embedded in the desktop

© Copyright IBM Corporation 2009. All rights reserved.



An alternative to what I just showed you is to set the install directory to /usr (assuming that is the directory under which KDE is installed on your system). This will cause the install step to fail unless you run KDevelop as a superuser (or you do the install manually as a superuser by changing to the build directory, $HOME/projects/dwPlasmoid/build, and running `sudo make install`).

Note: If you wish to keep these plasmoids on your desktop, and want them to work after a reboot without stopping and starting Plasma again, you need to tell KDE 4 to look in your plasmoid directory on startup. One way to do this is to add a line to your kde4rc file (/etc/kde4rc), as shown in Listing 4:

Listing 4. Telling KDE to look in plasmoid directory on startup

```
[Directories]
prefixes=/home/martynh/plasmoids
```


© Copyright IBM Corporation 2009. All rights reserved.

Modifying the sample

Okay, now that we have a simple plasmoid, let's make it do something a little more interesting than just displaying static text. Let's transform the plasmoid into a simple memory monitor.

First, try to remove some of the clutter from the window by removing the icon and the background. In the constructor (`dwPlasmoid::dwPlasmoid`), you are going to comment out the code that sets the background, as shown in Listing 5:

Listing 5. Commenting out code that sets background

```
//    setBackgroundHints(DefaultBackground);
//    m_svg.setImagePath("widgets/background");
```

Next, in the paint interface method (`dwPlasmoid::paintInterface`), comment out the part that adds the icon, as shown in Listing 6:

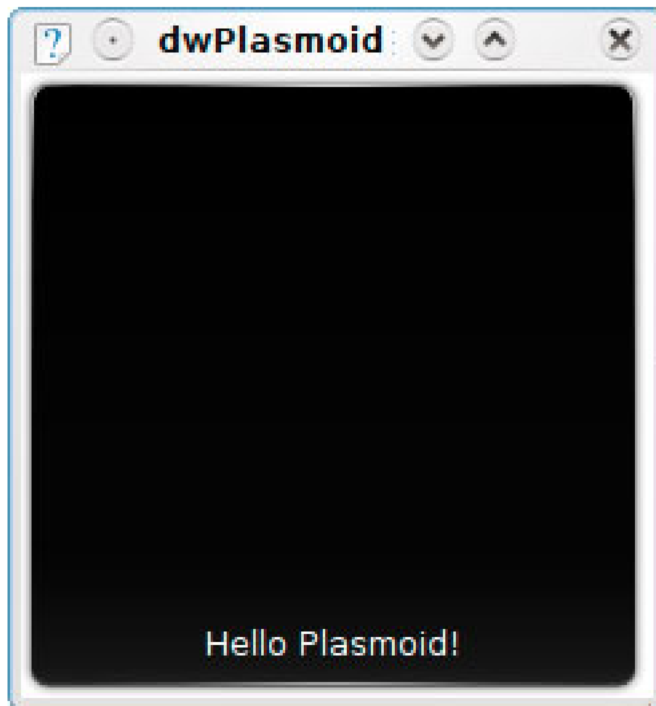
Listing 6. Commenting out code that adds icon

```
//    p->ldrawPixmap(7, 0, m_icon.pixmap((int)contentsRect.width(),
      (int)contentsRect.width()-14));
//    p->save();
```

If you rebuild and rerun this as before, you will see that the window is now just a black window with the "Hello Plasmoid" text, as shown in Figure 4:

Figure 4. Plasmoid with background and icon removed

© Copyright IBM Corporation 2009. All rights reserved.



Next you are going to replace the text with something a little more useful: some real-time information about the system memory use.

If you are familiar with Linux®, you may know that there are a number of special files under the /proc directory that give real-time information about the running system. The file we are interested in is /proc/meminfo. You can examine the contents of this file by typing this command: `cat /proc/meminfo`. This command shows a lot of information about the current state of memory on your system.

Let's alter the program we've built to periodically read from this file and display the information within the plasmoid. Start by reading the file and extracting the information. Thankfully, the Qt toolkit makes this relatively painless.

You're going to create a new method and add some new member variables to the class. In the header file, you'll add the method as a slot, as shown in Listing 7 (you will see why shortly):

Listing 7. Adding method as a slot

```
public slots:
    void updateMemoryCount();
```

Next, add some new private member variables, as shown in Listing 8:

Listing 8. Adding new private member variables

© Copyright IBM Corporation 2009. All rights reserved.

```
    private:
       Plasma::Svg m_svg;
       KIcon m_icon;
       int m_totalMemory;
       int m_freeMemory;
       int m_swapMemory;
       int m_freeSwapMemory;
```

Then, put the implementation in the cpp file, as shown in Listing 9:

Listing 9. Adding the implementation to the cpp file

```
#include <QFile>
...
void dwPlasmoid::updateMemoryCount()
{
   // open the file
   QFile file("/proc/meminfo");
   if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
   {
        exit(-1);
   }

   QTextStream in(&file);
   QString line = in.readAll();

   // remove any extraneous characters
   line = line.simplified();

   // extract the information we are interested in using a regular expression
   QRegExp regex("MemTotal: (\\d+).*MemFree: (\\d+).*SwapTotal: (\\d+)"
    ".*SwapFree: (\\d+)");   regex.indexIn(line);
   m_totalMemory = (regex.cap(1).toInt());
   m_freeMemory = (regex.cap(2).toInt());
   m_swapMemory = (regex.cap(3).toInt());
   m_freeSwapMemory = (regex.cap(4).toInt());

   // force a re-draw
   update();
}
```

This method opens the file, reads the contents out, and then extracts the values we are interested in using a regular expression.

Next, you need to change the paint method (`dwPlasmoid::paintInterface`) to add this information rather than the existing message, as shown in Listing 10:

Listing 10. Changing the paint method

```
   int percentageFreeMem = (m_freeMemory * 100) / m_totalMemory;
   int percentageFreeSwap = (m_freeSwapMemory * 100) / m_swapMemory;

   char message[256];

   sprintf(message,"Total Memory: %d\nFree Memory: %d\nPercent free mem: %d\n"
                   "Total Swap: %d\nFree Swap: %d\nPercent free swap: %d",
                   m_totalMemory, m_freeMemory, percentageFreeMem,
                   m_swapMemory, m_freeSwapMemory, percentageFreeSwap);
```

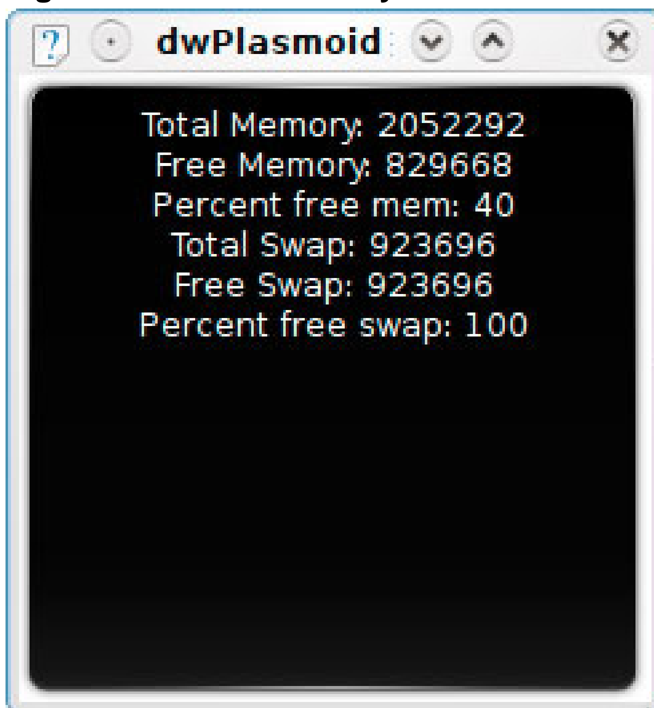

© Copyright IBM Corporation 2009. All rights reserved.

```
    p->setPen(Qt::white);
    p->drawText(contentsRect,
                Qt::AlignTop | Qt::AlignHCenter,
                message);
    p->restore();
```

Finally, put a call to `updateMemoryCount` into the constructor to read initial values for the memory figures: `updateMemoryCount();`.

If you compile and run this, you can see that it will display the correct memory usage as of when the plasmoid starts, as shown in Figure 5:

Figure 5. A static memory monitor



This is progress, but a static display of the system memory usage isn't all that interesting. The final piece of the puzzle is to make the plasmoid periodically update to provide current readings throughout its life cycle. This is easy because you declared the update method as a *slot*. You just need to add the following to the `init` method, as shown in Listing 11:

Listing 11. Making the plasmoid update periodically

```
#include <QTimer>
...
  // kick off a refresh timer
  QTimer* m_timer = new QTimer(this);
  connect(m_timer, SIGNAL(timeout()), this, SLOT(updateMemoryCount()));
  m_timer->start(1000);
```


© Copyright IBM Corporation 2009. All rights reserved.

This starts a timer that fires every second and calls the `updateMemoryCount` slot.

If you compile this and run it, you will see that it will automatically update every second!

Figure 6. Dynamically updating memory plasmoids embedded in the desktop

Notice that the values are not necessarily in sync. These instances of the plasmoid were added at different times, so each one started its timer at a different time.

Conclusion

As you can see, in just a short time, you can write a reasonably sophisticated

© Copyright IBM Corporation 2009. All rights reserved.

memory monitor to embed in your desktop.

Plasmoid technology allows you to write powerful, general-purpose utilities/monitors. Just as importantly, though, plasmoids are simple enough that you can write very specialized utilities that are useful only to you and that integrate into your desktop environment perfectly.

I hope that this article will inspire you to go out and create interesting plasmoids of your own. Use the comments section below to tell me and other readers about your experience and results.

© Copyright IBM Corporation 2009. All rights reserved.

Downloads

Description	Name	Size	Download method
Code samples for this article	code.zip	23KB	HTTP

Information about download methods

© Copyright IBM Corporation 2009. All rights reserved.

Resources

Learn

- To enable the PPA repositories to get KDE 4.2 and headers, see kubuntu.org for details.
- Build KDevelop 4 from source by following these relatively straightforward instructions on compiling KDevelop 4.
- The KDE UserBase wiki for Plasma is the best place to start learning about the Plasma desktop interface, one of the "Pillars of KDE." You'll also find an excellent, extremely detailed "Getting started" tutorial on creating your first plasmoid.
- The KDevelop-Project was originally designed to build an easy-to-use IDE for KDE.
- Starting with the first in a five-part series, "IBM open collaboration client solution: An overview" (developerWorks, October 2008), learn how to plan for and implement the OCCS, IBM's best of breed collaboration software and associated services for Linux.
- Check out this extensive listing of KDE articles on developerWorks.
- In the developerWorks Linux zone, find more resources for Linux developers, and scan our most popular articles and tutorials.
- See all Linux tips and Linux tutorials on developerWorks.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Kubuntu is the free, user-friendly operating system based on KDE and the award-winning Ubuntu.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Get involved in the My developerWorks community; with your personal profile and custom home page, you can tailor developerWorks to your interests and interact with other developerWorks users.

About the author

Martyn Honeyford

© Copyright IBM Corporation 2009. All rights reserved.

Martyn Honeyford graduated from Nottingham University with a BSc in Computer Science in 1996. He has worked as a software engineer in various guises ever since. Since his international snowboarding career shows no sign of taking off, and he has yet to find a method of paying the bills by playing video games, his day job is as a Senior Java™ Developer for an online derivatives trading company in London.

© Copyright IBM Corporation 2009. All rights reserved.