

KDE Scripting with DCOP

Boost your efficiency

There are features in some software products which only few people know about. DCOP is one of those undervalued features. It lets you easily write scripts that can control your whole KDE desktop and helps you save much work.

BY SCOTT WHEELER

Being good Linux evangelists, we're all too familiar with the Windows scripting vulnerability of the week. We read about them and snicker at the unenlightened masses. However, it may have happened – even just once, and surely at a weak moment – that we've pondered the useful aspects of scriptability, especially if the scripts can be abstracted away from the headaches that have made desktop scripting a buzzword of infamy.

DCOP Finally Leaves the Niche

Have the forces of good produced nothing to fill this void? I mean, surely we aren't interested, but maybe some sort of scripting framework would be just the thing we need to get those last few Windows users to switch to our favorite OS.

Well, rest assured that your friendly gang of Open Source developers hasn't abandoned you: I give you DCOP, and it's been there all along (well, since KDE 2.0 anyway), you just probably didn't know it. And hopefully – with a little patience – you'll see that desktop scripting, far from being universally evil, can in fact be pretty groovy.

DCOP is KDE's scripting back-end and the thing that KDE applications use to communicate with each other. It's been



hiding in the background telling your applications to update themselves when you change the settings and firing up KMail when you click on email addresses since the days of KDE 2.0.

But, as much as it would thrill me, you're probably not reading this to hear me go on and on about the wonders of KDE's internal design. Fortunately, there's something in it for you: DCOP isn't just for behind the scenes work! It can be used for anything from automating your business processes to setting the background image on your brother's computer to something embarrassing when his girlfriend is over – even from another continent. Root access is of course required to be able to use the DCOP interfaces of applications other than those running as your own user.

All KDE applications provide a basic DCOP interface that will let you do things like resize the main window or set it to "always on top". Most of them also provide an interface that will let you drive a good deal of the application: In KMail you can check your mail or create

a new message with a specific subject and body. In Konqueror you can open a new window to a specific URL, and in KWord you are able to change the properties of your document. The list goes on, but hopefully you're starting to get a feel for the power of DCOP.

The Tip of the Iceberg

DCOP is powerful, but it's also a little overwhelming at first glance. We'll start off small and work our way to some relative wizardry. Bundled with KDE is a tool called *KDCOP*. You won't find it in your menus, but it too has been there for a while. You can start it up either from the command line, or by choosing *Run Command* in the K-Menu. Also start up an instance of Konqueror and let's see what *KDCOP* has to tell us about it.

As you can see in Figure 1, *KDCOP* organizes things into a hierarchy. The top level shows the running KDE applications. When we click to see what DCOP knows about a specific application, we find a list of interfaces for that application and in each interface a list of DCOP methods. Additionally, some KDE applications, called unique applications, will only appear once in the menu (for example you can only run one instance of KMail at a time); others will append the process ID and may be listed multiple times. I'll refer to these terms a few times as we go along.

We'll let our first task be opening a URL with Konqueror. Find Konqueror in *KDCOP*'s list of applications, and click to expand it, so that you can see the interfaces it provides. One of these will be called *KonquerorIface (Default)*. Again, expand this item to reveal the methods that are available. A little more than halfway through the list of meth-

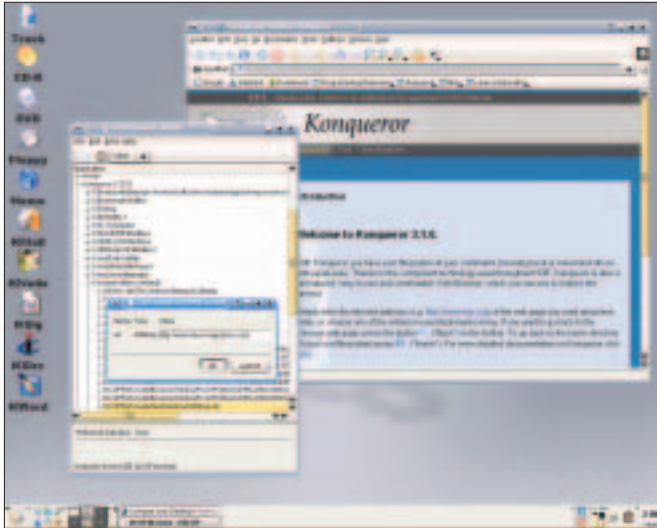


Figure 1: KDCOP in action. This nifty tool provides you with a nice GUI for controlling DCOP components. Here, we let the Konqueror application visit an URL which we inputted via the DCOP

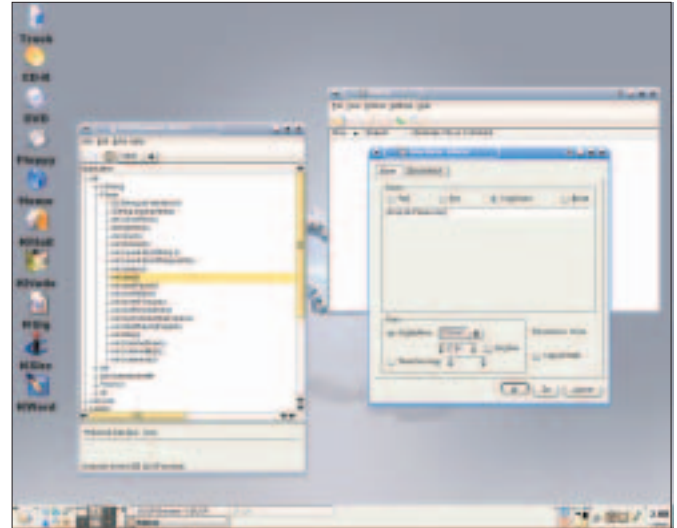


Figure 2: DCOP lets you control KDE applications from the command line. For example, with the help of KAlarm you can wake up in the morning with JuK playing your favourite music

ods, you will find `createNewWindow(QString url)`. Double click and you'll be prompted for – hey – a URL (see Figure 1). Enter your URL of choice and let the magic begin. Congratulations, hopefully with no battle wounds, you've just made your first DCOP call.

While you're in there, poke around a bit. I feel confident that you'll be able to figure out a few useful DCOP calls in your favorite applications.

Everybody Loves Drag & Drop

As exciting as clicking on things in KDCOP is, it's not particularly useful. To build up to our first example of DCOP at work, let's say that you wanted to have your computer wake you up in the morning, and since you're by now playing all of your music in this wonderful new application called JuK (OK, I'm biased; I wrote it.) that's going to be part of KDE 3.2, you decide that it would be nice to be able to combine the two. Now, JuK doesn't have an alarm clock feature, but thanks to the wonders of DCOP, it really doesn't need one. Here's why:

Listing 1: Desktop viewer

```
#!/bin/bash
while true; do
  dcop kwin KWinInterface
  nextDesktop
  sleep 1
done
```

If you look in your K-Menu, somewhere under the Utilities section, you'll find an application called KAlarm which causes the events you key in to happen at certain times. Click on the KAlarm icon in your system tray and you'll be presented with a list of events, probably empty. One more click on *New* and you're at the event entry screen. Select the *Command* option from the list of event types. Now, with KDCOP still on the same screen, drag an event into the available line. I chose `juk / Player / play()` – the Noatun equivalent is `noatun-[pid] / Noatun / play()`. In my case the text `dcop juk Player play` appears. Finish configuring the event to happen at your whim, and presto, your first useful DCOP call. Figure 2 shows an example of how to configure KAlarm.

Now, you could drag this command just about anywhere. Dragging it into Konsole you notice text as above appear at your command line. The clever reader is feeling the magic and has started with notions of writing shell scripts with DCOP.

Door Number Three

Now I think I've found the key to convincing even the Linux crew that we're on to something. Yes, that's right folks, if this wasn't clear from my hint before, you can control KDE applications from the command line. Before we jump into scripting, a brief introduction to the DCOP command line tool seems appropriate.

By simply calling `dcop` from the command line with a running instance of KDE, you'll be presented with the same list of running KDE applications that you saw at the top level in KDCOP. Then, if you provide just an application name, you will be given a list of interfaces. Try the following to see what I mean:

```
$ dcop kwin
qt
KDebug
KWinInterface (default)
MainApplication-Interface
```

You'll notice that the output is quite similar to what you would see looking at the list of interfaces in KDCOP. Following this pattern, if you provide just an application and interface, you'll be given a list of methods. Thus we've built up a command line representation of the same hierarchy that we saw in KDCOP. Now let's try to call one of those methods:

```
$ dcop kwin KWinInterface >
setCurrentDesktop 2
```

With that, you'll jump to virtual desktop two. Drag a few more methods from KDCOP into Konsole and you should get a feel for how it works.

Now let's try a small bash script that loops through all of our desktops, one per second. See Listing 1.

I used a less annoying version of the above last year at LinuxTag to show off

various KDE applications (and DCOP when folks asked what was going on) by bringing each running application into the foreground in a 30 second cycle.

Swiss-Army Knife

We all have a favorite scripting language – mine is Perl. If Perl isn't your thing, also included in the `kdebindings` package are DCOP bindings for Python, C, Objective-C and Java (C++ and the command line tool mentioned above are part of the core KDE distribution).

Since you've stayed with me so far, you're probably ready to see some non-toyish application of DCOP and a preview of how DCOP can be used with scripting languages.

Recently there was a mail to my LUG with a simple question, "Is there a way to put the contents of a file into the clipboard from the command line?" I, having been enlightened in the ways of KDE, set to the task of producing a simple Perl/DCOP script to do just that. See Listing 2.

In those eight lines of Perl, I was able to get the job done and win a few guru points on the LUG mailing list.

Let's take a look at this in a bit more detail. Some knowledge of Perl is assumed.

```
use DCOP;
```

Here we're importing the DCOP Perl module from the `kdebindings` module.

```
while($line = <>) {
    $contents .= $line;
}
```

This block of code reads the entire contents of either the standard input or that of a file passed as an argument while calling the script into the variable `$contents`.

```
$client = new DCOP();
$client->attach();
```

Here we create a new DCOP client and attach it to the running DCOP server. You



Figure 3: Klipper and DCOP team up to let you control the contents of KDE's clipboard from a Perl script

need to include these lines in any Perl script that uses the DCOP bindings.

```
$klipper = $client->createObject("klipper", "klipper");
```

Here we get to the part where we can put the things we've learned about DCOP so far to good use. Klipper is KDE's clipboard daemon; it stays docked in KDE's system tray. Looking back at `kdcop`, browse through the klipper interface. Here we are creating a DCOP object that we will call DCOP methods on later.

The first argument for the `createObject()` method is the application name; the second argument is the appropriate DCOP interface that contains the method we are interested in. In this case both of the arguments have the same name. After you've established your connection to the DCOP server you can create multiple DCOP objects for different tasks.

```
$klipper->setClipboardContents($contents);
```

By the wonders of Perl's dynamic nature, it now knows about the methods that are available in the object we just created. There is a direct mapping here, from the `setClipboardContents` DCOP method we saw, to a Perl method of the same name that acts on our `$klipper` object. Here we set the clipboard contents to the data that we read earlier into `$contents`.

Other methods follow this same pattern. There is one small quirk with returning values from DCOP methods. Using the Perl bindings, DCOP methods always return a list of values – even though there's generally only one item in the list. Suppose that we wanted to fetch the most recent entry in the clipboard history, instead of setting the contents. We would then substitute the last line of the previous example script for:

```
@history = $klipper->getClipboardHistoryItem( 0 );
print "$history[0]\n";
```

Here we ask for the most recent entry – item zero, which returns a list. On the next line we print the first (and only) item in that list.

Need more information on the Perl bindings for DCOP? If you have the `kdebindings` package installed, you should be able to view the more detailed documentation with `man:/DCOP` in Konqueror (or with the standard `man DCOP` from the command line).

The basics and syntax are quite simple, but DCOP does require a bit of patience in getting to know your way around. However, after this small investment in time, the number of truly silly things that you can make your desktop do are endless. Fortunately, the set of productive things that you can do with DCOP is pretty big too. ■

Listing 2: File contents to Clipboard

```
#!/usr/bin/perl
use DCOP;
while($line = <>) {
    $contents .= $line;
}
$client = new DCOP();
$client->attach();
$klipper = $client->createObject("klipper", "klipper");
$klipper->setClipboardContents($contents);
```