MailService

by Eric Celeste (AppTech)
version 1.1 released on 6 August 1992

This application is free to all. Source code should be available on the major ftp archive sites. Nothing about it is guaranteed, but I do use it on my own machine, so I believe it works! Enjoy.

All of us who dial into our university's computers and use the Terminal application to read our e-mail have to jump through a few hoops to get that mail moved over to NeXT's Mail application on our own machines. And if someone sends us NeXTmail, the hoops get even tougher. MailService is designed to make it easy for folks reading mail with the Terminal application to incorporate that mail into their NeXT Mailbox. This is achieved through a service added to the Mail submenu of the Services menu.

To send a message which you see in the terminal window to your own active mailbox (in the NeXT Mail sense of the term), you simply need to select the message and choose "Send As A Message" from the "Mail" submenu of the "Services" menu. I recommend that you select the message including its headers so that you can later tell who the message came from and when it was sent. If the message includes a NeXT attachment (in other words, it looks like gobbledygook in the terminal window), select it (gobbledygook and all) and send it anyway Ð the NeXT Mail application will correctly interpret the attachment present the message in legible form.

MailService can be used from any application which has a Services menu and lets you select text. This means you can automatically send messages from WriteNow and Edit, in addition to Terminal.

If you want the messages sent by MailService to go to someone other than yourself, just change the user name in the MailService window. Any subsequent messages sent by MailService will go to the user listed in that window.

There are two additional sections to this readme file, the first describes how to install this application and the second describes what went into writing it and offers some hints about writing NeXTSTEP services.

Installation

1. *Make sure you are running System Release 2.0 or later.* If not, get the upgrade and then come back to this installation.

2. If you received an application called MailService with this readme file, then skip to step 6. If you don't have that application yet, but you have a directory with the source files, then go on to step 3. If you have neither, then you are out of luck!

3. Make sure you have the System Release 2.0 Extended edition (or later) installed. The important point here is that you will need to have the development system online or you won't be able to go on with this process.

4. Open a terminal window and change directory (`cd`) to the directory which contains the MailService code.

5. Type `make` and hit return. Wait. When you get a shell prompt again (usually `%`) you will also have the MailService application in this directory.

6. Use the File Viewer to copy or link the MailService application into either your own `Apps` directory or the `/NextApps` directory.

7. Log out and then log back in. (The system only recognizes new services when logging in.)

There! You've done it. MailService is now installed. To test it, do the following:

A. Open up the Edit or WriteNow application.

B. Create or open a document and select some text in it.

C. Choose "Send As A Message" from the "Mail" submenu of the "Services" menu.

D. Watch as neat, cool things happen.

E. Open up the Mail application from the Dock or the `/NextApps` directory.

F. Look for a new message from yourself to yourself containing the text you had selected earlier.

If this process creates results, then the MailService application is working.

Writing Services

If you do not have the document "Welcome to 2.0: An Entry Point for Developers" Ð get it! This is a terrific little doc that makes many things about 2.0 more comprehensible. Appendix G of that document is also online as `/NextLibrary/Documentation/NextDev/ReleaseNotes/AppKit.rtf`. Both of these documents give very valuable instructions in the creation of services. Of course, I still spent all night trying to figure out why some things worked the way they did, so this little note is to try to explain some of these gottchas to you ahead of time.

The source code for MailService should include:

```
IB.proj
MailCarrier.h
MailCarrier.m
MailService.iconheader
MailService.icon.tiff
MailService.nib
MailService.services
MailService_main.m
```

```
Makefile
Makefile.preamble
```

The first step in creating a service is to create its description file. The description file for MailService is called `MailService.services`, though the name could be anything. The format of this description file is given in the `AppKit.rtf` document, so I won't go into here except to note that this is where I specified that the MailService menu item should appear as a submenu of the Mail item. Clustering similar services together will help us avoid cluttered Services menus.

The next step is to insert a reference to the service description file in the Mach-O file. I had no idea what a Mach-O file was, but learned that it is, essentially, the executable version of the application, and that the way to insert things there is through modifications of the `Makefile`. I chose to do this by creating a `Makefile.preamble` document which the unaltered `Makefile` automatically includes if present. This preamble simply makes sure that the compiler will include the contents of my service description file in the final executable code. (If you do a `strings MailService` you can see that it really is in the code!)

Now came the time for the real work. Services are built on the foundation of Speaker/Listener. MailService had to make sure an object that knew what to do was getting service messages from the applications Listener, `appListener`. I chose to call that object `MailCarrier`. It implements a method called `carryMail:userData:error:` which you will also see referred to as `carryMail` in the MailService.services description file. The message named in the description file *must* be the first part of the method's name in the implementing object. Equally important, the full method name *must* include the `userData` and `error` parameters even if you do not make use of them.

The way to let the application know that your new object should get service messages is by telling the `appListener` that it is the services delegate. At first I had `MailCarrier` do this in the `init` method. As I learned after a long and frustrating search, the `init` method of `MailCarrier` got called before the `appListener`

was ready to set its services delegate. I had to wait until the whole application was done with its initialization process before sending the `appListener` the `setServicesDelegate:` method. This is why you will see the little juggling tournament with `appDidInit:` in the source code. If anyone knows a better way to do this, please tell me! The rest of the source code should explain itself pretty well.

The final hurdle to developing a service is testing it. I could find no document from NeXT which describes exactly which directories are searched when the Services menu is created. Unfortunately, the list of directories does not seem to include `/LocalApps`. I could not get MailService recognized by the Services menu while it was only installed in `/LocalApps`. I also learned that changes in the services description file would not force a recompile unless some other file (like the `.nib`) were also touched. I instructed IB to use `make` instead of `make debug` (so that the Listener port would have the correct name) and then linked (with File Viewer) a copy of the application from my development directory over to my `Apps` directory (so that the Services menu would know where to find it). I learned that re-logging in is only necessary after changes are made to the service description file. I also learned that the workspace caches what it knows about services between logins and so I had to delete the files called `~/.NeXT/services/.cache` and `~/.NeXT/services/.applist` whenever I changed the service description or else the new description would not be read from the Mach-O file!

I hope all this helps anyone else developing services out there. I think services are a great addition to NeXTSTEP, and I hope we see a lot of them. You can reach me at the address below Ð if you have any questions or answers I'd love to hear from you.


```
 ________
|    |    |  Eric Celeste
|    |    |
\____
     \       Internet: efc@mit.edu
     /
,---- (:
```

5 Exeter Street
Belmont, MA 02178

617-484-5040