

Améliorez l'intégration au système de votre programme avec les classes Desktop et SystemTray de Java 6

par [Baptiste Wicht \(home\)](#)

Date de publication : 25 Juillet 2007

Dernière mise à jour :

Cet article va vous expliquer comment bien intégrer votre application au système. Vous apprendrez à utiliser les applications par défaut et comment mettre une icône dans la barre de notification.

- I - Introduction
- II - Desktop
- III - SystemTray
- IV - Conclusion

I - Introduction

Java a toujours été très peu intégré au système, ce qui est surtout dû au fait qu'il est portable et que certaines choses ne sont faisables que sur certains systèmes. C'est notamment pourquoi, il fallait se compliquer la vie rien que pour ouvrir le navigateur par défaut d'un système ou pour intégrer notre application dans la barre de notifications.

On pouvait employer des bibliothèques externes comme **JDIC** pour pallier à ce manque, mais une solution standard est souvent plus pratique. Si vous n'avez pas Java 6, vous serez obligé de passer par une de ces bibliothèques.

Les développeurs de Java ont donc pensé à l'intégration au système avec Java 6 et nous ont fourni la classe Desktop qui permet de faire certaines choses plus proches du système tels que :

- Ouvrir le navigateur par défaut à une certaine adresse
- Imprimer un fichier
- Ouvrir un fichier avec le programme par défaut
- Ouvrir le client mail

On a également le droit à la classe SystemTray qui permet d'afficher notre programme dans la barre de notifications du système.

Comme certains systèmes ne permettent pas certaines actions, il faut utiliser des méthodes `isXXXSupported` pour vérifier la disponibilité d'une fonction sur le système et agir en conséquence.

Nous allons maintenant voir comment utiliser ces deux classes.

II - Desktop

Voilà ce que permet de faire la classe Desktop :

- Lancer le navigateur par défaut à une URI spécifique.
- Lancer le client mail par défaut avec une URI mailto optionnelle.
- Lancer une application par défaut pour modifier, ouvrir ou imprimer un fichier spécifique.

Utilisation : On utilise toujours la classe Desktop de la même manière. On commence par tester si Desktop est supporté sur le système :

```
if(Desktop.isDesktopSupported()){  
}
```

Ensuite, on teste si la fonction que l'on va exécuter est disponible sur le système, par exemple pour la fonction mail :

```
if(Desktop.getDesktop().isSupported(Desktop.Action.MAIL)){  
}
```

Et enfin, quand on est sûr que tout est correctement supporté par le système, on peut appeler notre méthode :

```
try {  
    Desktop.getDesktop().mail();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Il ne faut bien sûr pas oublier de traiter les exceptions. Voilà donc le code que cela nous donnera pour ouvrir le client mail :

```
if(Desktop.isDesktopSupported()){  
    if(Desktop.getDesktop().isSupported(Desktop.Action.MAIL)){  
        try {  
            Desktop.getDesktop().mail();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Vous pouvez aussi bien sûr faire quelque chose dans le cas où l'action n'est pas supportée.

Ouvrir le navigateur : Pour ouvrir le navigateur à une page précise, il suffit d'obtenir une URI et d'utiliser la méthode `browse`. Par exemple, pour ouvrir le navigateur sur la rubrique Java de [developpez.com](http://java.developpez.com) :

```
if(Desktop.isDesktopSupported()){  
    if(Desktop.getDesktop().isSupported(Desktop.Action.BROWSE)){  
        Desktop.getDesktop().browse(new URI("http://java.developpez.com"));  
    }  
}
```

Desktop.browse() permet aussi d'ouvrir une URI avec le logiciel associé selon le protocole et fonctionne même pour d'autre protocole que les protocoles internet. Par exemple on peut utiliser le protocole callto:// conjointement avec Skype.

Ouvrir le client mail : Vous avez vu comment ouvrir le client mail par défaut plus haut, on va maintenant voir comment l'ouvrir en lui donnant directement une adresse de destination.

```
if(Desktop.isDesktopSupported()){
    if(Desktop.getDesktop().isSupported(Desktop.Action.MAIL)){
        Desktop.getDesktop().mail(new URI("mailto:qqun@domain.com"));
    }
}
```

Vous pouvez aussi remplir les autres paramètres du protocole mailto (pour plus d'infos, je vous laisse consulter la [RFC2368](#)) par exemple en passant le sujet :

```
if(Desktop.isDesktopSupported()){
    if(Desktop.getDesktop().isSupported(Desktop.Action.MAIL)){
        Desktop.getDesktop().mail(new URI("mailto:qqun@domain.com?subject=Sujet"));
    }
}
```

Voilà les principaux paramètres utilisables avec le protocole mailto :

- subject : Définit le sujet du message
- to : Ajout un destinataire
- cc : Ajoute un destinataire en copie
- bcc : Ajoute un destinataire en copie cachée
- body : Définit le contenu du message

Interactions avec les fichiers : Pour ouvrir, éditer et imprimer un fichier, vous pouvez employer les méthodes suivantes, respectivement : open, edit et print, qui prennent toutes un objet File en paramètre. Par exemple, si vous voulez imprimer un fichier Desktop.txt placé sur le C, vous allez faire :

```
if(Desktop.isDesktopSupported()){
    if(Desktop.getDesktop().isSupported(Desktop.Action.PRINT)){
        Desktop.getDesktop().print(new File("C:\\Desktop.txt"));
    }
}
```

Toutes ces méthodes sont néanmoins limitées par le logiciel par défaut qui leur est attribué.

Voilà, nous avons maintenant vu tout ce qu'il était possible de faire avec la classe Desktop. Nous allons donc voir les possibilités de la classe SystemTray.

III - SystemTray

La classe SystemTray va vous permettre d'intégrer une icône et un menu dans la barre de notifications (nommée différemment en fonction du système, barre de statut chez Windows, System Tray chez KDE, ...).

En général, on utilise la barre de notifications pour réduire une application graphique mais pour la laisser à disposition pour un moment.

Utilisation : L'utilisation ressemble à celle de Desktop. On commence par tester si la fonction est supportée par le système :

```
if(SystemTray.isSupported()){  
}
```

Et ensuite on récupère l'instance du SystemTray. C'est à partir de cette instance qu'on va ensuite pouvoir interagir avec la barre de notification.

```
SystemTray tray = SystemTray.getSystemTray();
```

Ce qui donne donc :

```
if(SystemTray.isSupported()){  
    SystemTray tray = SystemTray.getSystemTray();  
}
```

On va maintenant construire notre objet qui va s'afficher dans la barre. Il s'agit d'un objet TrayIcon. Il lui faut trois choses pour fonctionner, une icône, un texte qui va s'afficher lors du survol de l'icône et un menu qui va s'afficher si on clique droit sur l'icône. On va donc créer notre TrayIcon avec ces différentes choses.

```
Image image = ImageIO.read(getClass().getClassLoader().getResource("icone"));  
PopupMenu popup = new PopupMenu();  
MenuItem openItem = new MenuItem("Ouvrir");  
openItem.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        //Ouvre la fenêtre  
    }  
});  
popup.add(openItem);  
MenuItem closeItem = new MenuItem("Fermer");  
closeItem.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        //Ferme l'application  
    }  
});  
popup.add(closeItem);  
TrayIcon trayIcon = new TrayIcon(image, "Notre application", popup);
```

Voilà, nous avons maintenant un menu avec un bouton Ouvrir et un bouton Fermer et une icône qui s'affiche dans la barre de notification. On va donc la rajouter dans le SystemTray :

```
tray.add(trayIcon);
```

Pour l'en ressortir, il suffit d'utiliser la méthode `remove(TrayIcon icon)`. Nous en avons fini avec la classe `SystemTray`. Nous pouvons néanmoins configurer quelques autres paramètres depuis la classe `TrayIcon`. Vous pouvez notamment ajouter des listeners sur la `TrayIcon` pour réagir à d'autres événements que le clic droit. Par exemple pour ouvrir la fenêtre lors du clic gauche :

```
trayIcon.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        //Clic droit  
    }  
});
```

Vous pouvez aussi demander à ce que l'image soit redimensionnée en fonction de la taille qui lui est allouée. Comme cette taille n'est pas la même sur tous les systèmes, c'est très pratique et ça vous permet de ne pas avoir des icônes de tailles différentes en fonction du système. Pour faire cela, il vous suffit d'utiliser la méthode `setIconAutoSize` :

```
trayIcon.setIconAutoSize(true);
```

Et enfin, vous pouvez aussi afficher des messages depuis le system tray. C'est un message popup qui va s'afficher en dessus de votre icône. Ce message sera fermé après un certain temps ou dès qu'un utilisateur va cliquer dessus. Le clic d'un utilisateur peut lancer un `ActionEvent`. Il faut lui donner trois paramètres :

```
trayIcon.displayMessage("Erreur", "Problème survenu", TrayIcon.MessageType.ERROR);
```

IV - Conclusion

Voilà, nous avons maintenant vu comment mieux intégrer une application au système. Vous savez désormais comment ouvrir le navigateur par défaut ou le client mail et vous savez également comment intégrer votre application dans la barre de notifications du système. Et tout cela grâce à Java 6.

Si vous n'avez pas Java 6, je vous invite à consulter [ce document](#) qui va vous faire découvrir JDIC qui permet de faire à peu près les mêmes choses que les classes Desktop et SystemTray.

