Découverte de l'API de NetBeans Platform

par Mhamed Ben Jmaa (home page)

Date de publication : 16/05/2008

Dernière mise à jour : 16/05/2008

Dans cet article, nous allons découvrir les principales API de NetBeans Platform, permettant de développer une application en utilisant le Framework de l'IDE.

0 - Introduction

La plateforme netbeans est un outil très puissant pour la réalisation d'applications java swing (Reportez vous **ici** pour de plus amples informations sur cette plateformes) , elle offre un gain de temps en matière de productivité, reste à se familiariser avec ses outils rapidement, je vais donc vous faire part de quelques trucs et astuces pour vous éviter beaucoup d'heures de recherche dans les forums et autres articles.

I - Mise en place du projet

1. Nous allons commencer par créer un nouveau projet, File -> new Project -> NetBeans plug-in modules -> Module suite Project

Le module suite est un containeur de modules vous pourrez y déployer vos modules afin de les tester.

RQ : il est tout à fait possible de passer outre cette étape et aller directement à l'étape 2 si vous voulez que vos modules soient directement greffés dans l'IDE Netbeans lui-même.

2. Cliquez sur next, donnez un nom à votre module suite, on le nommera " masuite ", cliquez sur " finish ".

3. Lancez l'application. (Appuyez sur F6)

Première remarque :
C'est la copie conforme de netbeans, tout les modules sont là, ils sont chargés par défaut, on ne va pas avoir besoin de tous ces modules on va donc les enlever et travailler avec une plateforme basique, on ajoutera les modules dont on aura besoin au fur et à mesure.

4. Fermez l'instance de " masuite ".

5. Click droit sur le projet -> properties -> applications -> Create a standalone application. Vous aurez un avertissement qui vous demandera d'exclure tout les modules relatif à Netbeans, cliquez sur " exlude ", vous pourrez désormais changer le titre, l'icône, le splash screen " écran de démarrage " de votre application.

Une fois que vous aurez fini, cliquez sur OK, vous pouvez relancer l'application, on voit bien que le temps de démarrage se réduit considérablement, cela est dû au fait que il ya beaucoup moins de modules à charger lors du lancement.

5. Une fois que le modulesuite est en place, on va créer un module dans le quel on placera notre travail :
File->new project ->netbeans plug-in modules->Module Project, cliquez sur Next,

Vous appelez votre module comme vous voulez, dans mon cas j'ai choisie " moduledessai ". On va placer notre module dans le module suite " masuite " que l'on vient juste de mettre au point.

RQ : si vous avez choisie de ne pas crée de module suite, il suffit de placer votre module dans " Standalone Module ".

6. Click next, donnez un " code base name " à votre module, prenez l'habitude de bien nommer vos modules, afin d'avoir une bonne structuration et pour éviter des confusions.

7. Click finish.

II - L'Utilisation des actions :

Les actions sont un excellent moyen d'interagir avec le composant que l'on crée que se soit des éditeurs, des wizards (comme nous allons le voir dans ce qui suit), ou autres, c'est le système qui gère la communication avec ces composants grâce au fichier XML (Layer.XML), cela nous évite donc d'écrire du code.

On va commencer par ajouter une action dans le menu 'File', c'est une Action à titre introductif elle se contentera d'afficher un message.

1. Click droit sur " moduledessai ", new -> file folder -> netbeans module devlopment -> Action.

2. Click next, on dira que notre action peut être appelée à tout moment, on va donc choisir " always enabled ".

3. Click next,

On passe alors à la configuration de notre Action, sa catégorie, son emplacement, que ce soit, dans la barre de menus, dans la barre d'outils, ou même les deux, on peut éventuellement ajouter un raccourci clavier pour l'appel de notre Action.

J'ai choisi de mettre mon action dans la catégorie Tool, dans le menu 'File' et que cette dernière soit suivie d'un séparateur.

4. Click next :

On va donner un nom de class à cette Action, pour ma part j'ai choisi " MonAction ", il faut aussi choisir le texte qui va s'afficher dans le menu, le nom de package, vous pouvez ajouter une icône au menu, pour un résultat plus esthétique, sachez que le format est supporté, vous pouvez donc créer des icônes avec des effets de transparences.

5. Click finish.

On obtient un fichier JAVA en plus du fichier bundle et le layer qui sont créés à la création du module :

Le fichier Bundle.properties dans lequel on place les informations concernant l'action dans notre cas ça sera le nom.

Ce nom va être extrait grâce à cette méthode :

```
public String getName() {
 return NbBundle.getMessage(MonAction.class, "CTL_MonAction");
}
```

Le fichier layer.xml qui va être modifié contient la description et la configuration de notre action : son emplacement dans le menu, l'instance, la catégorie etc#, ce fichier contiendra également la configuration de tout ce que vous allez ajouter dans la barre de menus.

Enfin, le fichier java dans lequel on pourra implémenter notre code.

Nous allons utiliser l'objet NotifyDescriptor pour afficher un Dialog.

On va donc ajouter ce code dans la methode performAction()

```
public void performAction() {
    String msg = "Aimez-vous Netbeans ?";
    NotifyDescriptor d = new NotifyDescriptor.Confirmation(msg,
 NotifyDescriptor.INFORMATION_MESSAGE,NotifyDescriptor.YES_NO_CANCEL_OPTION);
    Object o = DialogDisplayer.getDefault().notify(d);

    if(o==NotifyDescriptor.YES_OPTION)
     DialogDisplayer.getDefault().notify(new NotifyDescriptor.Message("Vous aimez Netbeans !!"));

    if(o==NotifyDescriptor.NO_OPTION)
        DialogDisplayer.getDefault().notify(new
 NotifyDescriptor.Message("Vous aimez pas Netbeans !!?"));

    if(o==NotifyDescriptor.CANCEL_OPTION)
        DialogDisplayer.getDefault().notify(new
 NotifyDescriptor.Message("Vous n'avez pas donné d'avis !!"));
}
```

Première remarque, DialogDisplayer n'est pas détecté, il faut donc aller l'ajouter dans la bibliothèque :

Pour ajouter le support de notifydescriptor et dialogdisplayer :

6. Click droit sur le module ->properties->libraries->add dependency->Dialogs API.

RQ : il est aussi possible de taper le nom de la classe dans la zone du filtre pour que l'interface réduise le choix

7. Click OK

8. Click OK, on revient a MonAction.java on fait un fix imports (ou alt+maj+f), et le problème et résolu !.

9. On lance le projet et on essaye !!

III - Réalisation d'un simple Wizard :

Maintenant qu'on a clarifié les actions passons au Wizard :

1. Click droit sur " moduledessai " new -> file folder ->Netbeans Module development ->Wizard.

On utilisera :

- Registration Type : Custom.
- Step Sequence : Static.
- Number of wizard panel : 2.

2. Click next.

3. Click finish.

Première Remarque : Netbeans a crée pour nous un fichier Action, on va donc pouvoir placer notre wizard, dans le barre de menu en ajoutant quelques lignes dans le fichier Layer.xml :

Dans la balise <folder name = action> on ajoute notre wizard dans la catégorie Window :

```
<folder name="Window">
            <file name="org-yourorghere-moduledessai-monwizard-MonWizardWizardAction.instance"/>
 </folder>
```

Dans la balise <folder name = Menu> on ajoute

```
<folder name="Window">
 <file name="org-yourorghere-moduledessai-monwizard-MonWizardWizardAction.shadow">
         <attr name="originalFile" stringvalue="Actions/Window/org-yourorghere-moduledessai-monwizard-
MonWizardWizardAction.instance"/>
     </file>
     <attr name="org-yourorghere-moduledessai-monwizard-MonWizardWizardAction.shadow/org-yourorghere-
moduledessai-monwizard-separatorAfter.instance" boolvalue="true"/>
     <file name="org-yourorghere-moduledessai-monwizard-separatorAfter.instance">
         <attr name="instanceClass" stringvalue="javax.swing.JSeparator"/>
     </file>
     <attr name="org-yourorghere-moduledessai-monwizard-separatorAfter.instance/org-netbeans-modules-
project-ui-logical-tab-action.shadow" boolvalue="true"/>
</folder>
```

RQ : les lignes que vous ajouterez dans le fichier XML dépendent du nom du package que vous avez choisie, en ce qui me concerne c'est " org.yourorghere.moduledessai.monwizard ", dans le fichier XML il faut remplacer les " . " par des " - ".

Désormais notre wizard apparaîtra dans le menu Window.

Pour changer le titre du menu il suffit de changer dans le fichier action le renvoi de la méthode :

```
public String getName() {
    return "Start Sample Wizard";
}
```

On va plutôt mettre :

```
public String getName() {
    return "Démarrer mon Wizard";
}
```

Si on veut faire les choses en respectant la philosophie de Netbeans, il faut créer un fichier Bundle.Properties et attribuer un nom, comme c'était le cas pour MonAction vu en première partie.

Jetons un oeil sur le fichier MonWizardWizardAction.java.

La méthode performAction contient l'initialisation du Wizard :

La mise en place du titre, il faudra changer différentes parties du code :

La mise en place du titre, il faudra changer différentes parties du code :

```
wizardDescriptor.setTitle("Your wizard dialog title here");
```

on changera ça par :

```
setTitle("Premier Wizard");
```

Ensuite vient une condition très importante c'est " if(! Canceled) " si cette condition est vérifiée autrement dit si l'utilisateur a cliqué sur finish, alors le traitement peut commencer, il ne faut faire aucun traitement tant que l'utilisateur n'a pas fini, c'est l'esprit du wizard.

Toutes les informations rentrées pendant les étapes du wizard serons stocker grâce à l'objet wizardDescriptor, on va voir ça en détails ultérieurement.

Commençons par mettre au point nos VisualPanel :

MonWizardVusalPanel1 ressemblera à ça :

Jetons un oeil sur le code source :
On va changer le nom de l'étape

```
public String getName() {
 return "Saisie du nom";
}
```

On ajoutera aussi un getter pour le JTextField, moi je l'ai nommé nomTxt, je vais avoir donc ceci,

```
public JTextField getNomTxt() {
 return nomTxt;
}
```

Il suffit de faire ctrl+espace dans l'éditeur, netbeans vous propose de créer le getter.

Le second ressemblera à ça :

Il faudra ajouter un ButtonGroup et de placer le ouiRdo et le nonRdo dedans.

Il faut aussi ajouter un getter pour le ouiRd, le nonRdo et le nomLbl. (on placera dans ce label, le nom qu'aura saisi l'utilisateur dans l'étape précédente).

Par la même occasion on changera le titre de l'étape :

On remplace

```
public String getName() {
 return "Step #2";
}
```

Par

```
public String getName() {
 return "Question à propos du tuto";
}
```

Occupons nous de la transmission d'informations d'une étape à une autre:

Dans le fichier MonWizardWizardPanel1.java

On remarque que la méthode getComponent renvoie un nouveau objet de type MonWizardVisualPanel1 si ce dernier n'existe pas, on va donc changer la déclaration de component, ce n'est plus de type Component mais de type MonWizardVisualPanel1, on aura besoin de l'instance de cet objet pour avoir accès à la méthode getNomTxt(), qu'on a créé tout à l'heure dans MonWizardVisualPanel1, pour pouvoir stocker les informations de l'utilisateur.

Affin de stocker ou récupérer ces informations on utilisera la méthode, storeSettings et readSettings comme suit :

```
public void readSettings(Object settings) {
    }

public void storeSettings(Object settings) {
    ((WizardDescriptor)settings).putProperty("Le nom",component.getNomTxt().getText());
}
```

En ce qui concerne MonWizardWizardPanel2.java, on fera la même chose mais pour le ouiRdo cette fois ci :

MonWizardWizardPanel2.java

```
public class MonWizardWizardPanel2 implements WizardDescriptor.Panel {

    /**
     * The visual component that displays this panel. If you need to access the
     * component from this class, just use getComponent().
     */
    private MonWizardVisualPanel2 component;

    // Get the visual component for the panel. In this template, the component
    // is kept separate. This can be more efficient: if the wizard is created
    // but never displayed, or not all panels are displayed, it is better to
    // create only those which really need to be visible.
    public Component getComponent() {
        if (component == null) {
```

MonWizardWizardPanel2.java

```
            component = new MonWizardVisualPanel2();
        }
        return component;
    }

    public HelpCtx getHelp() {
        // Show no Help button for this panel:
        return HelpCtx.DEFAULT_HELP;
        // If you have context help:
        // return new HelpCtx(SampleWizardPanel1.class);
    }

    public boolean isValid() {
        // If it is always OK to press Next or Finish, then:
        return true;
        // If it depends on some condition (form filled out...), then:
        // return someCondition();
        // and when this condition changes (last form field filled in...) then:
        // fireChangeEvent();
        // and uncomment the complicated stuff below.
    }

    public final void addChangeListener(ChangeListener l) {}
    public final void removeChangeListener(ChangeListener l) {}
    /*
    private final Set<ChangeListener> listeners = new HashSet<ChangeListener>(1);
    public final void addChangeListener(ChangeListener l) {
        synchronized (listeners) {
            listeners.add(l);
        }
    }
    public final void removeChangeListener(ChangeListener l) {
        synchronized (listeners) {
            listeners.remove(l);
        }
    }
    protected final void fireChangeEvent() {
        Iterator<ChangeListener> it;
        synchronized (listeners) {
            it = new HashSet<ChangeListener>(listeners).iterator();
        }
        ChangeEvent ev = new ChangeEvent(this);
        while (it.hasNext()) {
            it.next().stateChanged(ev);
        }
    }
     */

    // You can use a settings object to keep track of state. Normally the
    // settings object will be the WizardDescriptor, so you can use
    // WizardDescriptor.getProperty & putProperty to store information entered
    // by the user.
    public void readSettings(Object settings) {
        //récupération du nom de l'utilisateur
        component.getNomLbl().setText("Votre nom est : "+(String)
 ((WizardDescriptor)settings).getProperty("Le nom"));
    }
    public void storeSettings(Object settings) {
        Boolean reponse = new Boolean(component.getOuiRdo().getModel().isSelected());
        ((WizardDescriptor)settings).putProperty("Reponse",reponse);
    }

}
```

Maintenant qu'on a fini de tout mettre au point on va rajouter la partie traitement dans MonWizardWizardAction.java :

On affichera un Dialog récapitulatif de notre saisie, on ajoutera donc ce bout de code dans le condition (if(!cancelled)) :

```
if (!cancelled) {
    boolean reponse = ((Boolean)wizardDescriptor.getProperty("Reponse")).booleanValue();
    String
 msg = "Votre nom est : "+(String)wizardDescriptor.getProperty("Le nom")+" et votre réponse fut : "+
          (reponse?"Oui":"non");
    NotifyDescriptor d = new NotifyDescriptor.Message(msg);
    DialogDisplayer.getDefault().notify(d);
}
```

On lance l'application, dans le menu tools -> on voie le menu " Démarrer mon Wizard ", notre wizard s'affiche et notre récapitulatif est valable mais :

Le passage d'une étape à une autre se fait sans vérification, on peut donc cliquer sur next sans entrer aucune donnée, on va corriger ça :

Jetons un #il du coter de MonWizardWizardPanel1.java

Il ya une méthode isValid() ; qui renvoie constamment un true, c'est le renvoie de cette méthode qui détermine si oui ou non, il est possible de passé à l'étape suivante, il faut donc enlever le return true, et ajouter des listener sur les composants pour signaler les modifications que fait l'utilisateur et vérifier la validité des entrées :

On ajoute implements " DocumentListener ".

On ajoute la déclaration d'une variable boolean isValid ;

On enlève les commentaires sur les méthodes " addChangeListener ", " removeChangeListener " " fireChangeEvent() ".

Vous allez devoir upgrader les sources vers la version 1.5 pour ajouter le support des " generics " click droit sur moduledessai dans l'onglet " project " -> properties -> sources -> source level -> 1.5. -> enable Warnigns -> Click OK

Et on ajoute une méthode de vérification, on obtient un résultat comme suit :

MonWizardWizardPanel1

```
public class MonWizardWizardPanel1 implements WizardDescriptor.Panel, DocumentListener {

    private MonWizardVisualPanel1 component;
    //la variable qui décide de la validité de Panel
    private boolean isValid;


    public Component getComponent() {
        if (component == null) {
            component = new MonWizardVisualPanel1();
            //grâce à cette ligne on pourra écouter les changements sur le JTextField!
            component.getNomTxt().getDocument().addDocumentListener(this);
        }
        return component;
    }

    public HelpCtx getHelp() {
        // Show no Help button for this panel:
        return HelpCtx.DEFAULT_HELP;
        // If you have context help:
        // return new HelpCtx(SampleWizardPanel1.class);
    }

    public boolean isValid() {
        // on va enlever ce revoie car la validité n'est pas toujours vérifiée
        //return true;
        return isValid;
```

MonWizardWizardPanel1

```
    }
    //on enlève ces deux déclarations
    //public final void addChangeListener(ChangeListener l) {}
    //public final void removeChangeListener(ChangeListener l) {}

    //voila pourquoi il faut upgrader vers la version1.5
    private final Set<ChangeListener> listeners = new HashSet<ChangeListener>(1);
    public final void addChangeListener(ChangeListener l) {
        synchronized (listeners) {
            listeners.add(l);
        }
    }
    public final void removeChangeListener(ChangeListener l) {
        synchronized (listeners) {
            listeners.remove(l);
        }
    }
    protected final void fireChangeEvent() {
        Iterator<ChangeListener> it;
        synchronized (listeners) {
            it = new HashSet<ChangeListener>(listeners).iterator();
        }
        ChangeEvent ev = new ChangeEvent(this);
        while (it.hasNext()) {
            it.next().stateChanged(ev);
        }
    }

    public void readSettings(Object settings) {
    }
    public void storeSettings(Object settings) {
        ((WizardDescriptor)settings).putProperty("Le nom",component.getNomTxt().getText());
    }

    public void insertUpdate(DocumentEvent e) {
        verifierLaValidite();
    }

    public void removeUpdate(DocumentEvent e) {
        verifierLaValidite();
    }

    public void changedUpdate(DocumentEvent e) {
        verifierLaValidite();
    }

    private void verifierLaValidite() {
        String currentText = component.getNomTxt().getText();
        boolean isValidInput = currentText != null && currentText.length() > 0;
        if (isValidInput) {
            setValid(true);
        } else {
            setValid(false);
        }
    }

    private void setValid(boolean b) {
        if (isValid != b) {
            isValid = b;
            //il faut absolument appeler cette méthode pour que le panel rappel
            //la methode isValid
            fireChangeEvent();
        }
    }

}
```

Il faudra ajouter la même chose pour le MonWizardWizardPanel2, sauf qu'on ne mettra pas de " DocumentListener " mais plutôt un " ItemListener " on obtient un résultat comme suit :

```
public class MonWizardWizardPanel2 implements WizardDescriptor.Panel, ItemListener {


    private MonWizardVisualPanel2 component;
    private boolean isValid;

    public Component getComponent() {
        if (component == null) {
            component = new MonWizardVisualPanel2();
            component.getOuiRdo().addItemListener(this);
            component.getNonRdo().addItemListener(this);
        }
        return component;
    }

    public HelpCtx getHelp() {
        // Show no Help button for this panel:
        return HelpCtx.DEFAULT_HELP;
        // If you have context help:
        // return new HelpCtx(SampleWizardPanel1.class);
    }

    public boolean isValid() {
        return isValid;
    }

    //public final void addChangeListener(ChangeListener l) {}
    //public final void removeChangeListener(ChangeListener l) {}

    private final Set<ChangeListener> listeners = new HashSet<ChangeListener>(1);
    public final void addChangeListener(ChangeListener l) {
        synchronized (listeners) {
            listeners.add(l);
        }
    }
    public final void removeChangeListener(ChangeListener l) {
        synchronized (listeners) {
            listeners.remove(l);
        }
    }
    protected final void fireChangeEvent() {
        Iterator<ChangeListener> it;
        synchronized (listeners) {
            it = new HashSet<ChangeListener>(listeners).iterator();
        }
        ChangeEvent ev = new ChangeEvent(this);
        while (it.hasNext()) {
            it.next().stateChanged(ev);
        }
    }



    public void readSettings(Object settings) {
        component.getNomLbl().setText("Votre nom est : "+(String)
 ((WizardDescriptor)settings).getProperty("Le nom"));
    }
    public void storeSettings(Object settings) {
        Boolean reponse = new Boolean(component.getOuiRdo().getModel().isSelected());
        ((WizardDescriptor)settings).putProperty("Reponse",reponse);
    }

    public void itemStateChanged(ItemEvent e) {
        verifierLaValidite();
    }
```

```
    private void verifierLaValidite() {
        boolean oui = component.getOuiRdo().getModel().isSelected();
        boolean non = component.getNonRdo().getModel().isSelected();
        boolean isValidInput = oui != non;
        if (isValidInput) {
            setValid(true);
        } else {
            setValid(false);
        }
    }

    private void setValid(boolean b) {
        if (isValid != b) {
            isValid = b;
            fireChangeEvent();
        }
    }


}
```

On lance notre application, on remarque bien que le bouttons next et finish ne sont plus toujours actifs.

IV - L'utilisation de l'Output

Dans cette section on verra comment on peut utiliser la fameuse fenêtre Output de Netbeans, pour ce faire on va utiliser le résultat du wizard qu'on a créé précédemment mais au lieu de l'afficher dans un dialog, on va placer le résultat dans la fenêtre Output :

On va d'abord ajouter les modules nécessaires :

Click droit sur masuite -> propoerties -> librairies -> n#ud plateforme6 -> et on coche " IO/APIs ". -> on coche aussi OutPutWindow -> Click OK.

Click droit sur moduledessai -> properties -> Add Dependency -> IO/APIs -> Clik OK -> required Tokens Add# -> org.openide.windows.IOProvider -> Click ok -> Clik ok.

On remplace le code de la condition (if(!canceled))

```
if (!cancelled) {
    //cette instruction recherche une instance de InputOutput, si elle n'existe pas elle en crée une
    // le false c'est pour que l'affichage dans l'output window se fait dans l'onglet "Mon Wizard"
    //si il existe, si on met true, alors un nouvel onglet apparaitra.
    InputOutput io = IOProvider.getDefault().getIO("Mon Wizard", false);
    io.select();
    boolean reponse = ((Boolean)wizardDescriptor.getProperty("Reponse")).booleanValue();
    String
 msg = "Votre nom est : "+(String)wizardDescriptor.getProperty("Le nom")+" et votre réponse fut : "+
            (reponse?"Oui":"non");
    OutputWriter w;
    if (reponse)
        //pour un affichage normal
        w = io.getOut();
    else
        //pour afficher une information importante, le texte apparaîtra en rouge
        w = io.getErr();
    w.println(msg);
}
```

Ajoutez les imports nécessaires, click droit sur masuite->clean and build all.

Lancer le projet, et voila le travail.

V - Remerciements

Un grand merci pour l'équipe developpez.com pour sa patience, et pour David Gimelle pour la relecture.

Cordialement

Ben Jmaa Mhamed