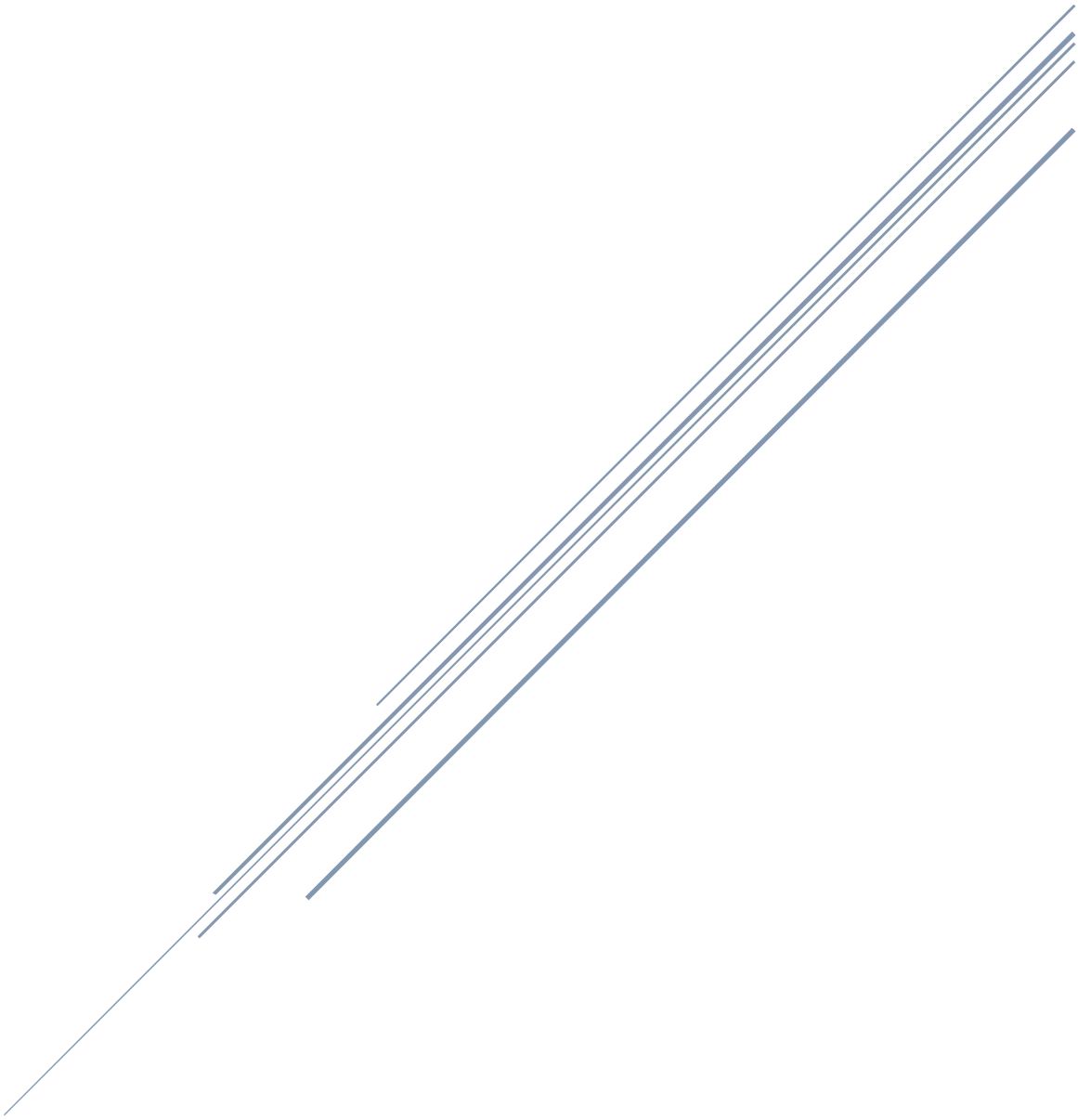
GENERATEUR DE FORMULAIRE GUI

Documentation Programmeur



Team Infotik

Projet S3 – 2017/2018

Introduction

Cette documentation vous permettra d'obtenir toutes les informations nécessaires pour modifier les formulaires selon vos besoins.

Elle se fera sous la forme d'une présentation des différentes actions effectuées lors des appels de l'utilisateur.

Il est conseillé pour une meilleur compréhension d'avoir les sources ouvertes à coté afin de pouvoir suivre le chemin effectué.

Table des matières

I. Les fonctions principales

Nous allons voir dans cette partie quel sont les chemins suivis par le programme lors des actions principales tel que le création d'un nouveau formulaire.

1) La fonction showMessage

Si cette fonction est en première partie, c'est par-ce-qu'elle est utilisée dés qu'il est nécessaire de communiquer avec l'utilisateur par l'utilisation d'une popup, il est donc nécessaire de la connaître pour la suite de la lecture :

```
public static void showMessage( String titre , String message )
{
	JOptionPane.showMessageDialog( null, message, titre, JOptionPane.ERROR_MESSAGE );
}
```

Elle se situe dans la classe iut.algo.form.FormController.

2) La création d'un formulaire

Lors de la création d'un nouveau formulaire que ce soit par l'appel de **FormController.createForm(String xmlPath)** ou de **FormController.createAndGetForm(String xmlPath)**, nous commençons par vérifier que le fichier existe et qu'il possède bien une extension en .xml :

```
File xmlFile = new File(filePath);
if (!xmlFile.exists())
{
	showError("Le fichier ciblé n'existe pas");
	return -1;
}
else if ( !xmlFile.getName().toUpperCase().endsWith(".XML") )
{
	showError("Le fichier ciblé ne correspond pas à un fichier XML");
	return -1;
}
```

Nous en créons ensuite une copie du fichier xml sous forme de fichier temporaire auquel on enlève toutes les lignes précédant l'ouverture de la balise « <form> » afin de ne pas pénaliser l'utilisateur s'il se trompe sur cette partie du xml, puis on ajoute « <?xml version="1,0"?><!DOCTYPE form SYSTEM "dtdFile.dtd"> » sur la même ligne que la balise « <form> » avant celle-ci. Le fichier « dtdFile.dtd » que nous utilisons est un fichier temporaire créé par la méthode FormController.createDtdFile lors de l'exécution afin que l'on puisse valider le xml même si l'utilisateur indique un autre fichier dtd.

Nous arrivons ensuite à la validation du fichier dans la méthode validXml(File xmlFile) qui à l'aide du package javax.xml valide en accord avec la dtd comme le ferais la commande « xmllint –valid » cette méthode lève des erreurs qui sont gérées par le SimpleErrorHandler (du package iut.algo.form.job) les affichant dans une popup. Si aucune erreur n'est détectée à ce point nous appellons la méthode FormController.verifType(Element root) sinon la création du formulaire est abandonnée.

VerifType permet de d'appeler récursivement sur chaque nœud du fichier xml la méthode attributeOk qui vérifie les valeurs des différents attributs de l'élément et indique les éventuelles erreurs par l'utilisation d'une popup.

Une fois tout cela effectué si aucune erreur n'a été rencontrée, nous arrivons à la fin de la méthode d'origine (FormController.createForm) où nous créons la fenêtre en correspondance avec l'objet xml validé.

3) La création de la fenêtre

Nous somme ici dans la méthode iut.algo.form.view.Frame.createFrame(Element xmlRoot), nous somme ici sûr de la structure de l'objet xml, celle-ci ayant été vérifiée précédemment.

Nous commençons par détecter la langue choisies par l'utilisateur suivis de la lecture des attributs de la fenêtre, suivi par la création de la fenêtre en accord avec ces attributs (titre, longueur, largeur, positionX, positionY), nous parcourrons ensuite le reste de l'objet xml afin de lire et créer tous les contrôles en les ajoutant à la fenêtre au fur et à mesure. A chaque ajout de composant la présence des attributs « x » et « y » sont vérifiés, en cas d'absence, on indique à la suite que les contrôles sont à placer automatiquement.

```
if ( !isPlacedAutomatically && (x == -1 || y == -1) )
        isPlacedAutomatically = true;
```

Suite à l'ajout de tous les composants, nous appellons la méthode addKeyListenerToAllComponents() qui permet d'ajouter sur chaque composant de la fenetre une instance unique de iut.algo.form.view.FormKeyListener, le KeyListener gérant les raccourcis ctrl+i et ctrl+t permettant respectivement d'afficher et de cacher les id et les types des contrôles.

Retour à la méthode createFrame, nous trions les contrôles par ordre d'id, afin de les placer dans le bon ordre s'il doivent être placé automatiquement (en colonne). S'ils doivent être placé automatiquement, cela est géré par la méthode placeControlsAutomatically qui les place l'un au dessus de l'autre puis les rend visible, sinon ils sont juste rendu visible.

La fenêtre ainsi créée est alors retournée pour que le FormController correspondant puisse la référencé au besoin.

4) La récupération des valeurs

La récupération des valeurs se fait par des primitives d'instance de FormController tel quel getInt(String id) ou encore getString(String id).

Ces méthodes sont sous la forme :

```
public Integer getInt (String id)
{
	if (intMap != null)
	{
		return intMap.get(id);
	}
	return null;
}
```

On vois qu'elles vont chercher les valeurs dans une Map, cette Map est initialisé lorsque l'utilisateur appuie sur le bouton « valider » par l'appel de la méthode windowValidated() trouvée dans la classe FormController, cette méthode récupère pour chaque contrôle de la fenêtre sa valeur et la met dans la liste correspondant à son type.

La récupération des tableaux est un peu différente puisque les méthodes nécessitent que l'utilisateur passe un tableau du type et de la taille de celui à récupérer :

```
public <T> boolean getArray (String id, T[][] res)
{
	try
	{
	T[][] tmp = (T[][]) arrayMap.get(id);

	for (int i = 0; i < res.length && i < tmp.length; i++)
		for (int j = 0; j < res[i].length && j < tmp[i].length; j++)
		{
			res[i][j] = tmp[i][j];
		}

		return true;
	}
	catch (Exception e) {  }
	return false;
}
```

Les valeurs sont alors écrient à l'intérieur de ce tableau, le retour sert à indiquer à l'utilisateur si l'action est un succès ou un échec.

II. Les classes

1) Le package form

Il s'agit du package principal de l'archive, c'est ce package que l'utilisateur doit importer, il ne contient que la classe FormController, étant la classe générant la création des formulaires.

FormController

La classe appelée par un programme extérieur afin de créer les formulaires et les gérer.

C'est également cette classe qui est le point d'entré de l'archive, dans ce cas elle ne sert qu'à générer un fichier dtd en utilisant la même méthode que celle utilisée pour le fichier dtd temporaire à la validation du xml (voir I. 2) La création d'un formulaire).

2) Le package form.job

Le package Job est un package contenant les différentes classes utilisées à travers tous le programme.

BaseType

Énumération des 5 type de bases (int, String, boolean, double et character), elle permet de connaître le type de retour de chaque contrôle.

Language

Énumération des 2 langages disponibles (Français et Anglais), elle permet de savoir la langue utilisée pour la fenêtre et ainsi d'indiquer les labels correspondant

SimpleErrorHandler

Classe permettant la gestion des erreurs levée par la validation du xml. L'affichage des erreurs se fait par la méthode FormController.showError

3) Le package form.view

Frame

Classe gérant la création et les interaction avec les fenêtre d'affichage. Cette classe permet entre autre d'obtenir tous les composant qu'elle possède mais c'est également elle qui gère le positionnement des contrôles (voir I. 3) La création de la fenêtre).

FormKeyListener

Classe fille de KeyListener, elle détecte lorsque l'utilisateur utilise les raccourcis ctrl+i et ctrl+t et indique à la Frame à laquelle il est lié d'afficher ou cacher les identifiants/type pour chacun de ses composants

Control

Classe abstraite mère de tous les contrôles qui sont affiché sur la Frame, c'est également elle qui gère le positionnement des contrôles.

Elle possède des méthodes tel que getType ou encore obtainWidth utilisées lors de la création et la validation des formulaires.

Label

Contrôle de type String, il est le contrôle le plus simple n'étant qu'un affichage de texte, son getValue retourne le texte écrit.

Text

Contrôle de type String ou Int, il est le contrôle affichant une zone dans laquelle l'utilisateur rentre des valeurs au claviers, son getValue retourne le texte écrit.

TextKeyListener

Classe utilisée pour permettre l'utilisation du raccourci ctrl+z dans les contrôles de type Text, ce raccourci permet de retourné à la dernière valeur enregistrée. La valeur est enregistrée à chaque fois que le Text perd le focus.

Buttons

Contrôle de type Int, c'est le contrôle gérant un groupe de boutons radios.

Checkbox

Contrôle de type Boolean, comme son nom l'indique, il affiche une checkbox.

Calendar

Contrôle de type String, il ne fait que contrôler un DateTextField.

DateTextField

Classe permettant à l'utilisateur de rentrer une date, sa valeur par défaut étant la date d'aujourd'hui.

Il est possible de changer le format de la date en modifiant DEFAULT_DATE_FORMAT.

Le panneau s'affichant lorsque l'on clique sur le TextField est généré dans createWeekAndDayPanel

Dropdown

Contrôle de type String, ce contrôle est un menu déroulant, son getValue retourne le texte sélectionné

Array

L'Array est le contrôle qui permet d'afficher un tableau, il affiche à coté de lui un contrôle correspondant au type de retour qu'on lui demande (Text pour String ou Int par exemple)

Il implémente KeyListener pour permettre à l'utilisateur de se déplacer dans le tableau affiché à l'aide des touches fléchées.