Tutorial: Création d'une application sans base de données avec le Visual Web Pack

par Valère Déjardin (Creabeans)

Date de publication : 02/12/2006

Dernière mise à jour : 15/12/2006

Creator et son héritier, le Visual Web Pack de Netbeans, sont des formidables outils pour réaliser rapidement des applications web tout en profitant de la puissance de java côté serveur. Cependant, lorsque les données à traiter ne proviennent pas du triptyque DB/EJB/WS, les choses se compliquent...

I - Introduction
II - Installation
III - Gérer les fichiers
 III.A - Attributs, getters et setters
 III.B - Rendre notre classe Comparable
 III.C - Rendre notre classe Serializable
 III.D - Les classes terminées
 III.E - Lire et gérer les données en session
IV - Construire un dropdown
V - Remplir une table
 V.A - Créer un dataProvider pour nos données
 V.B - Récupérer les données de voyages relatives à une personne.
VI - Rajouter une seconde page et de la navigation
VII - Conclusion
VIII - Sources et Crédits

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

I - Introduction

Les fonctionnalités de Java Studio Creator sont maintenant disponibles dans Netbeans via le Visual Web Pack (VWP), ce qui facilite considérablement la vie du développeur qui peut maintenant se reposer sur les nombreuses fonctionnalités de Netbeans, y compris les nouveautés de Java EE 5. Mais qu'en est-il si on souhaite travailler directement avec des fichiers comme source de données?

La version finale du VWP vient à peine d'être publiée, et je n'ai pas encore suffisamment de recul pour vérifier que toutes les instabilités touchant la version Technology Preview ont bien été corrigées. En particulier, si le mode design ne fonctionne plus ou n'affiche pas vos tableaux, une solution: arrêtez Tomcat, faîtes un clean et rebuild de votre projet, puis fermez et rouvrez le projet dans NB.

Pour ce tutorial, je vais (dans une certaine mesure) recréer le projet créé par le tutorial **Insertion, mise à jour et suppression** de Creator, tout en me contentant de travailler en lecture seule.

Pour commencer, j'ai récupéré et un peu modifié les données des tables PERSON et TRIP de l'exemple fourni - en rajoutant dans la table TRIP les données de la table TRIPTYPE. J'obtiens en résultat deux fichiers **person.txt** et **trip.txt**, avec le caractère tabulation comme délimiteur de champs.

Le tutorial est rédigé en suivant les règles du J2SE 1.4 (pas de generics, pas d'autoboxing). Lorsqu'on créé un projet VWP dans Netbeans, en choisissant Tomcat 5 comme cible, Netbeans recommande de fixer le *source level* à 1.4 (et je n'ai pas souhaité modifier ces options par défaut). Sachez simplement que l'utilisation des fonctionnalités du JDK 1.5 est tout à fait possible même avec Tomcat 5.

Le projet Netbeans fini est **disponible ici**.

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

II - Installation

Les éléments suivants sont requis pour faire mettre en pratique ce tutorial (à installer dans l'ordre). Netbeans et le JDK existent aussi en bundle, c'est à dire un téléchargement pour installer les deux programmes.

- JDK 1.5 minimum, requis pour faire fonctionner Netbeans (**téléchargement**)
- Netbeans 5.5 (**téléchargement**)
- Visual Web Pack pour Netbeans 5.5 (téléchargement accessible depuis la page précédente)

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

III - Gérer les fichiers

III.A - Attributs, getters et setters

Dans NB, commençons par créer un nouveau projet Visual Web Pack: Menu File | New Project... | Web | Visual Web Application. Appelons ce projet nodb, et pour limiter les risques de collision définissons le package par défaut com.developpez.dejardin.nodb.

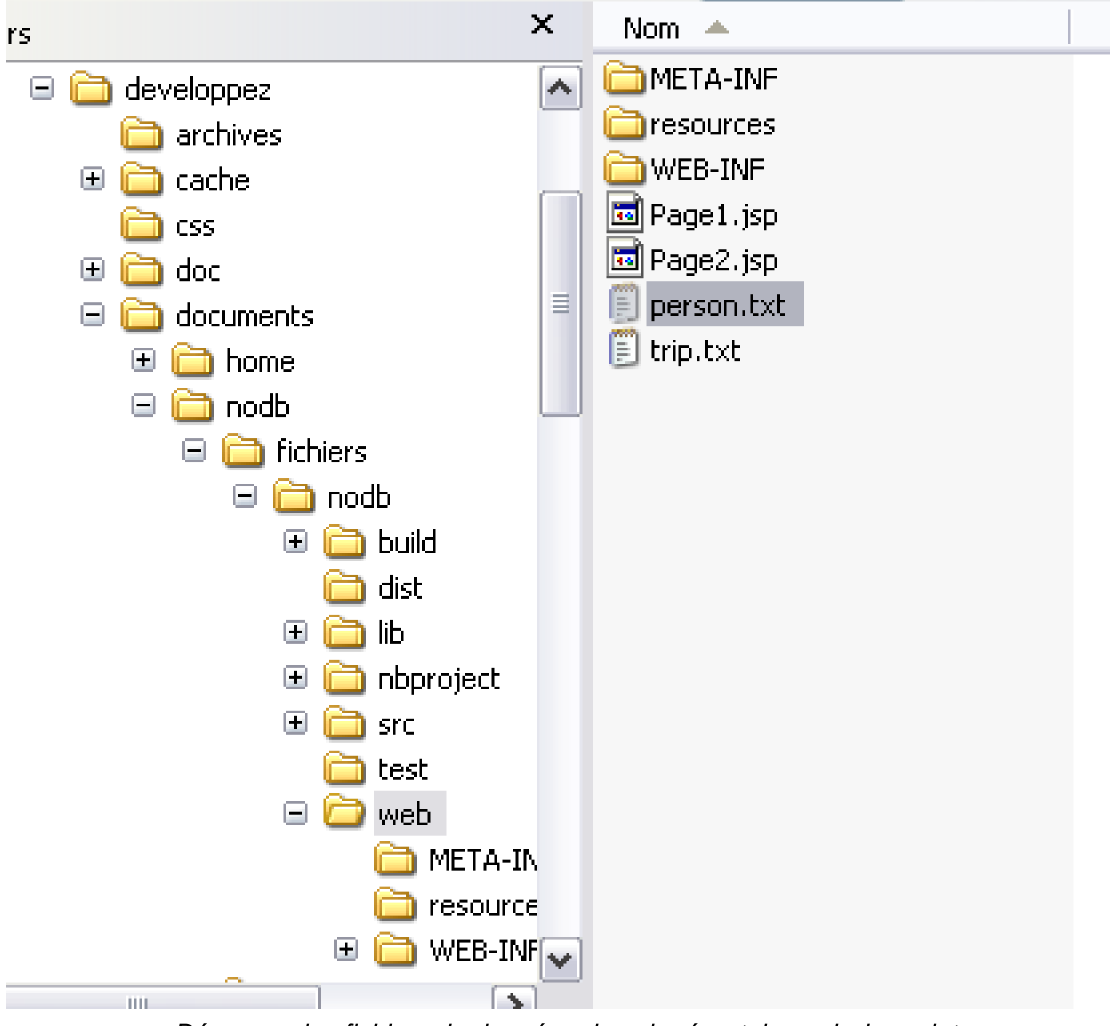
Création du projet

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Création du projet (2)

A l'aide de notre explorateur de fichiers externe, déposons les fichiers person.txt et trip.txt dans le répertoire web de notre projet.

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.



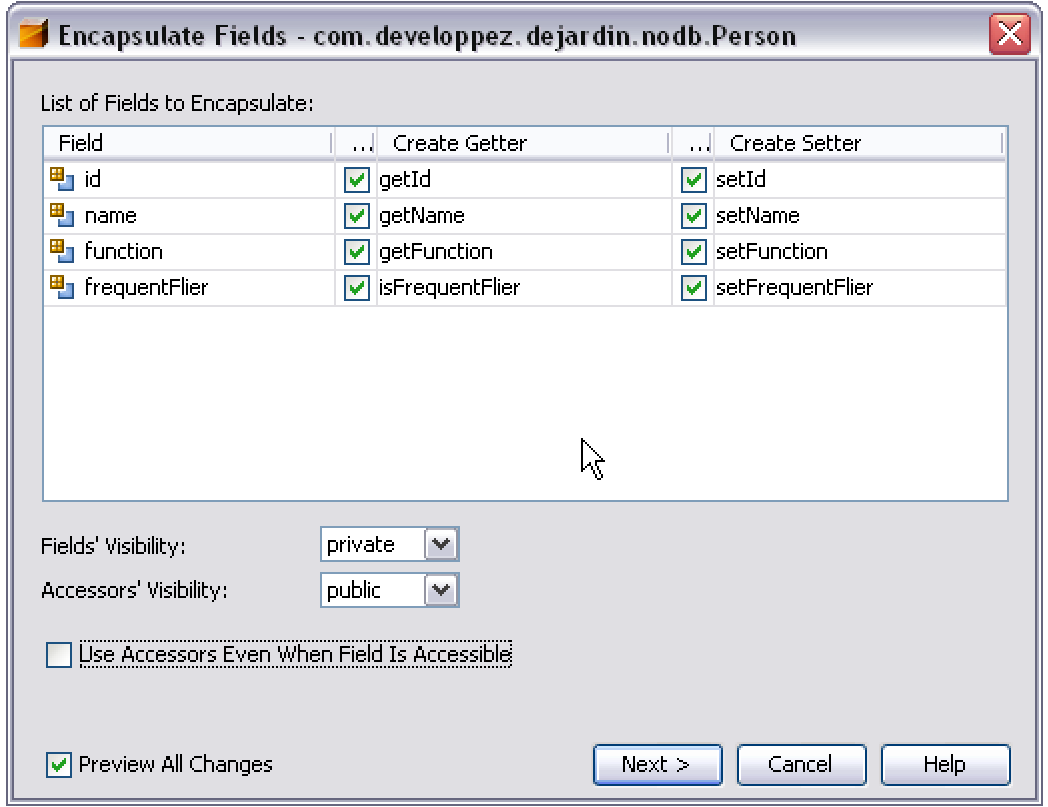
Déposons les fichiers de données dans le répertoire web du projet

Ensuite, pour commencer occupons nous de la lecture et de la récupération des données. Créons un nouveau fichier java, Person.java: Menu File | new File... | Java Class | Class name: Person et package com.developpez.dejardin.nodb, et initialisons le rapidement avec les noms des colonnes.

```
package com.developpez.dejardin.nodb;

public class Person {
    int id;
    String name;
    String function;
    boolean frequentFlier;

    /** Creates a new instance of Person */
    public Person(int id, String name, String function, boolean frequentFlier) {
        this.id = id;
        this.name = name;
        this.function = function;
        this.frequentFlier = frequentFlier;
    }

}
```


Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Un petit refactoring va s'occuper de créer nos getters et setters (ainsi que passer les attributs en private): plaçons le curseur dans la classe, Menu Refactor | Encapsulate field. Sélectionnons le cas échéant tous les getters et setters, dévalidons "Use accessor even when field is accessible", affichons la prévisualisation des changements et appliquons. Il ne manque plus que la javadoc à créer (votez pour le bug **http://www.netbeans.org/issues/show_bug.cgi?id=48296**). Vous pouvez constater que Netbeans a pris soin de régler l'accessibilité des champs sur *private*.



La fonction Encapsulate field

J'ai recommandé de décocher l'option "Use accessor even when field is accessible" car cette option remplace l'usage direct des noms des champs dans la classe par le getter ou le setter. Par exemple, elle remplace *this.id = id* en *this.setId(id)*. Je trouve cette encapsulation excessive et je préfère ne pas l'appliquer.

Voici le résultat:

```
package com.developpez.dejardin.nodb;

public class Person {
    private int id;
    private String name;
    private String function;
    private boolean frequentFlier;

    /** Creates a new instance of Person */
    public Person(int id, String name, String function, boolean frequentFlier) {
        this.id = id;
        this.name = name;
        this.function = function;
        this.frequentFlier = frequentFlier;
    }
```


Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getFunction() {
        return function;
    }

    public void setFunction(String function) {
        this.function = function;
    }

    public boolean isFrequentFlier() {
        return frequentFlier;
    }

    public void setFrequentFlier(boolean frequentFlier) {
        this.frequentFlier = frequentFlier;
    }

    public int compareTo(Object o) {
        return compareTo((Person) o);
    }

    public int compareTo(Person person) {
        return (this.name).compareTo(person.getName());
    }
}
```

III.B - Rendre notre classe Comparable

Maintenant, nous allons implémenter l'interface Comparable. De cette manière, il sera facilement possible d'affecter un ordre par défaut à nos données (l'ordre alphabétique sur les noms dans notre cas), au lieu de se contenter de l'ordre dans lequel les données sont présentes dans le fichier person.txt. Modifions la déclaration de votre classe en

```
public class Person implements Comparable {
```

Netbeans va alors signaler en rouge que notre classe contient une erreur: en effet, nous indiquons que notre classe implémente Comparable, mais les méthodes nécessaires ne sont pas présentes. Cliquons sur l'ampoule que Netbeans affiche, et sélectionnons la correction proposée: *Implements all abstract methods*. Netbeans rajoute alors la méthode *compareTo()*... mais encore une fois soulignée de rouge: en effet la méthode est censée retourner un int, mais ici elle ne retourne rien.

Nous pouvons terminer d'implémenter Comparable comme suit: c'est la valeur de l'attribut name de Person qui va permettre de les classer. Utilisons également la méthode toLowerCase() pour éviter les erreurs de tri si des personnes ne sont pas toutes saisies de la même manière.

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
    public int compareTo(Object o) {
        return compareTo((Person) o);
    }

    public int compareTo(Person person) {
        return (this.name.toLowerCase()).compareTo(person.getName().toLowerCase());
    }
```

III.C - Rendre notre classe Serializable

Lorsque Tomcat s'arrête, il essaie de conserver les informations relatives aux sessions des utilisateurs, afin de pouvoir redémarrer plus tard dans les mêmes conditions que précédemment. Pour ce faire, Tomcat va tenter de sérialiser tous les objets en mémoire. Afin d'éviter de remplir nos fichiers de logs avec de disgracieuses *java.io.NotSerializableException*, nous allons déclarer que notre classe implémente l'interface *Serializable*.

Complétons la classe Person en rajoutant Serializable dans la liste des interfaces:

```
public class Person implements Comparable, Serializable {
```

L'ampoule et le surlignage rouge signalant une erreur apparaissent. Cliquons sur l'ampoule, une suggestion apparaît signalant de déclarer la classe java.io.Serializable. Pas de problème, on applique cette suggestion. Puis, rien plus d'erreur. A la différence de l'interface Comparable, Serializable ne nécessite pas d'implémenter de classe abstraites, l'interface sert uniquement de marqueur pour signaler que la classe est bien sérialisable (ce qui est le cas car tous les champs sont eux-mêmes sérialisables). Java s'occupe de tout.

Toutefois, une petite modification du code est nécessaire: en effet, une classe sérialisable doit posséder un constructeur vide, ce qui n'est pas le cas de la notre. De plus, il est fortement recommandé de rajouter un champ *private static final long serialVersionUID* dans notre classe pour lui attribuer un numéro de version unique. Complétons donc notre méthode avec le code suivant:

```
    private static final long serialVersionUID = 1L;

    /** Creates a new instance of Person */
    public Person() {
    }
```

Plus d'information sur la sérialisation:

- *La **FAQ** de développez.com*
- *Article **La sérialisation binaire en Java** par Yann D'ISANTO sur developpez.com*
- *La **javadoc***

III.D - Les classes terminées

Notre fichier Person.java est désormais terminé. Nous pouvons le retrouver Source **ici**. De la même manière, il faut également créer le fichier Trip.java, que nous trouverons Source **ici**. Voici ci dessous le descripteur de la classe ainsi que ses attributs.

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
// skipping package and import...

public class Trip implements Comparable, Serializable {
    private int tripId;
    private int personId;
    private Date departureDate;
    private String fromCity;
    private String toCity;
    private String tripType;
    private String tripDesc;

// Skipping contructors, getters, setters, compareTo()...
```

III.E - Lire et gérer les données en session

Il est maintenant nécessaire de s'occuper de la gestion des fichiers par l'application. Un peu à la manière des datasources "databases" de Creator, nous allons travailler depuis le bean SessionBean1.java créé avec le projet. Il faut d'abord indiquer à l'application où se trouvent les fichiers de données. Pour cela nous allons utiliser les paramètres de contexte de l'application web. En effet, une application JSF reste avant tout une application JSP/Servlet, elle dispose d'un fichier web.xml qu'il est possible d'exploiter.

Dans le projet, cliquer sur "Configurations files" et ouvrir le web.xml. Dans Général | context parameters, ajoutons des paramètres indiquants que les fichiers sont dans le répertoire racine de l'application (une fois déployée, ce qui correspond au répertoire web du projet).

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Page1 ✕ | Page2 ✕ | SessionBean1.java ✕ | web.xml ✕ | Person.java ✕ | Trip.java ✕

General | Servlets | Filters | Pages | References | Security | XML | General

General

Display Name:

Description:

Distributable

Session Timeout: min.

Context Parameters

Param Name	Param Value	Description
javax.faces.STATE_SAVING_ME...	server	
javax.faces.CONFIG_FILES	/WEB-INF/navigation.xml,/WEB-...	
com.sun.faces.validateXml	true	
com.sun.faces.verifyObjects	false	
person	/person.txt	
trip	/trip.txt	

Add... | Edit... | Remove

Web Application Listeners

Indiquons à notre application où trouver ses données

Il serait inutile de relire et reparser les fichiers à chaque requête de l'utilisateur. Nous allons donc créer deux méthodes, une pour chaque fichier, pour trouver dans le contexte de l'application le chemin des fichiers, lire leur date de dernière modification, la comparer avec une valeur stockée en session, et le cas échéant ordonner une mise à jour.

```
/**
 * last time the person file was modified
 */
private Date personDate = new Date(0);

/**
 * last time the trip file was modified
 */
private Date tripDate = new Date(0);

public void checkPerson() {
```


Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
            // Get the parameter person set in web.xml
            // This parameter value is relative to the root of the application
            String personPath = getExternalContext().getInitParameter("person");

            // compare last modified time of the file to the internal last value
            if (personPath != null) {
                // getting servlet context
                ServletContext context = (ServletContext) getExternalContext().getContext();
                // getting the full path of the file, depending of the place from
                // where you run this example
                personPath = context.getRealPath(personPath);
                File personFile = new File(personPath);

                if ((personFile.exists()) && (personFile.lastModified() > personDate.getTime())) {
                    managePerson(personFile);
                }
            }
        }

        // idem for checkTrip()
```

Toujours dans Session1.java, nous allons rajouter la logique pour parser et gérer les fichiers. Chaque ligne des fichiers person.txt et trip.txt sera bien sûr mémorisée respectivement dans un objet Person et Trip, mais comment organiser ces données? Il nous faut une collection.

Souvenons-nous de Person.java. Nous avions pris soin d'implémenter l'interface Comparable dans cette classe, c'est le moment de l'utiliser. Nous choisissons la collection TreeSet. L'avantage de cette collection est que chaque nouvel élément qui y est rajouté est trié en respectant l'ordre *naturel* des éléments, c'est à dire l'ordre obtenu par le biais de l'interface Comparable. De cette manière, même si les personnes ne sont pas rangées dans l'ordre dans notre fichier person.txt, elles seront dans l'ordre alphabétique en mémoire et dans toutes les fonctions qui y feront appel.

C'est décidé, les personnes seront stockées dans un TreeSet. Mais en ce qui concerne les Trips? Réfléchissons à l'usage que nous souhaitons en faire... Nous voulons afficher les voyages réalisés par une personne. Il serait donc intéressant de pouvoir dans un premier temps regrouper tous les voyages réalisés par une personne. Comme pour Person.java, Trip.java implémente Comparable, utilisons à nouveau un TreeSet pour stocker les voyages (triés par date de départ cette fois). Et pour regrouper tous ces TreeSet, une simple Hashtable, avec l'id des personnes comme clef, fera l'affaire.

Dans cet exemple, la gestion des exceptions est rudimentaire: toute ligne de données qui est mal formée sera simplement ignorée.

```
        /**
         * handle the contens of the person file
         */
        private TreeSet personTreeSet = new TreeSet();

        /**
         * handle the content of the trip file, keyed by personId
         */
        private Hashtable tripHashtable = new Hashtable();

        /**
         * parses the person file and store results in personTreeSet
         * @param personFile the File where to find data
         */
        private void managePerson(File personFile) {
            String strLine = null;

            try {
                // getting file to work with
```


Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
            FileInputStream in = new FileInputStream(personFile);
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));

            // for each line of the file
            while ((strLine = reader.readLine()) != null) {
                int personId = 0;
                String name = null;
                String function = null;
                boolean frequentFlier = false;

                // parsing each line -- take care, bad exceptions handling
                String[] elements = strLine.split("\\t");

                try {
                    personId = Integer.parseInt(elements[0]);
                    name = elements[1];
                    function = elements[2];
                    frequentFlier = ("1".equals(elements[3])) ? true : false;
                } catch (NumberFormatException nfe) {
                    // Exception handling here
                    continue;
                }

                // creating new Person object and adding it to the treeset
                // I used treeset and the interface comparable in the Person
                // class to have a sorted collection
                Person person = new Person(personId, name, function, frequentFlier);
                personTreeSet.add(person);
            } // end while

            this.personDate = new Date(personFile.lastModified());
        } catch (FileNotFoundException ex) {
            log("Error finding person.txt file: " + ex.toString());
            error("Error finding person.txt file: " + ex.toString());
            ex.printStackTrace();
        } catch (IOException ex) {
            log("Error parsing person.txt file: " + ex.toString());
            error("Error parsing person.txt file: " + ex.toString());
            ex.printStackTrace();
        }
    }

    /**
     * parses the trip file and store results in tripHashtable
     * @param tripFile the File where to find data
     */
    private void manageTrip(File tripFile) {
        String strLine = null;
        DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT, Locale.FRANCE);

        try {
            // getting file to work with
            FileInputStream in = new FileInputStream(tripFile);
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));

            // for each line of the file
            while ((strLine = reader.readLine()) != null) {
                int tripId = 0;
                int personId = 0;
                Date departureDate = null;
                String fromCity = null;
                String toCity = null;
                String tripType = null;
                String tripDesc = null;

                // parsing each line -- take care, bad exceptions handling
                String[] elements = strLine.split("\\t");
```


Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
                    try {
                        tripId = Integer.parseInt(elements[0]);
                        personId = Integer.parseInt(elements[1]);
                        departureDate = df.parse(elements[2]);
                        fromCity = elements[3];
                        toCity = elements[4];
                        tripType = elements[5];
                        tripDesc = elements[6];
                    } catch (NumberFormatException nfe) {
                        // Exception handling here
                        continue;
                    } catch (ParseException ex) {
                        // Exception handling here
                        continue;
                    }

                    // creating new Trip object
                    Trip trip = new Trip(tripId, personId, departureDate, fromCity, toCity, tripType,
    tripDesc);

                    // storing Trip object into a hashtable of treeset
                    TreeSet trips4user = (TreeSet) tripHashtable.get(new Integer(personId));

                    if (trips4user == null) {
                        trips4user = new TreeSet();
                    }

                    trips4user.add(trip);
                    tripHashtable.put(new Integer(personId), trips4user);
                } // end while

                this.tripDate = new Date(tripFile.lastModified());
            } catch (FileNotFoundException ex) {
                log("Error finding trip.txt file: " + ex.toString());
                error("Error finding trip.txt file: " + ex.toString());
                ex.printStackTrace();
            } catch (IOException ex) {
                log("Error parsing trip.txt file: " + ex.toString());
                error("Error parsing trip.txt file: " + ex.toString());
                ex.printStackTrace();
            }
        }
```

Maintenant, il suffit de rajouter l'appel à ces fonctions lors de l'initialisation du bean de session, dans Init():

```
        // Perform application initialization that must complete
        // *after* managed components are initialized
        // TODO - add your own initialization code here
        checkPerson();
        checkTrip();
```

Enfin, comme ces données devront être accessibles depuis la Page1.java, nous rajoutons un getter sur personTreeSet et TripHashtable.

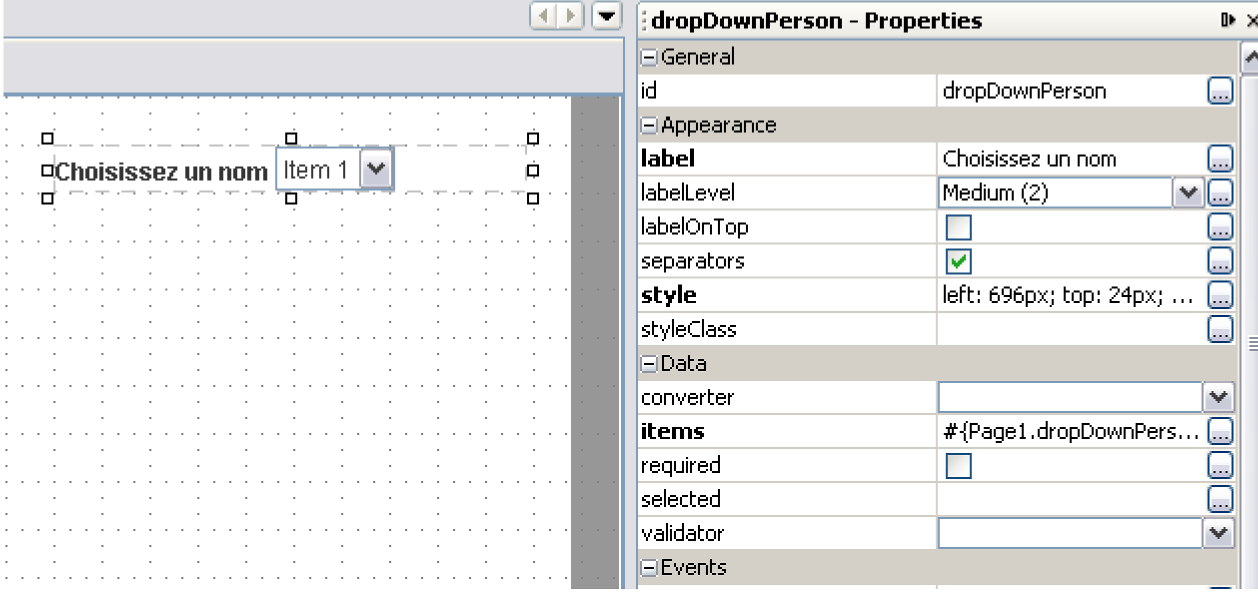
```
    public Hashtable getTripHashtable() {
        return tripHashtable;
    }

    public TreeSet getPersonTreeSet() {
        return personTreeSet;
    }
```


Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

IV - Construire un dropdown

Concentrons nous désormais sur quelque chose qui concerne davantage le VWP: nous devons rajouter un dropdown sur notre page d'accueil. Avec une base de données, pas de problème: il suffit de glisser-déposer une table sur un dropdown en mode design. Ici, c'est un peu plus compliqué, mais ce n'est pas non plus irréalisable!

Plaçons d'abord un dropdown sur notre page, appelons-le dropdownPerson, ajoutons lui un label "Choisissez un nom".



Création et initialisation de notre dropdown.

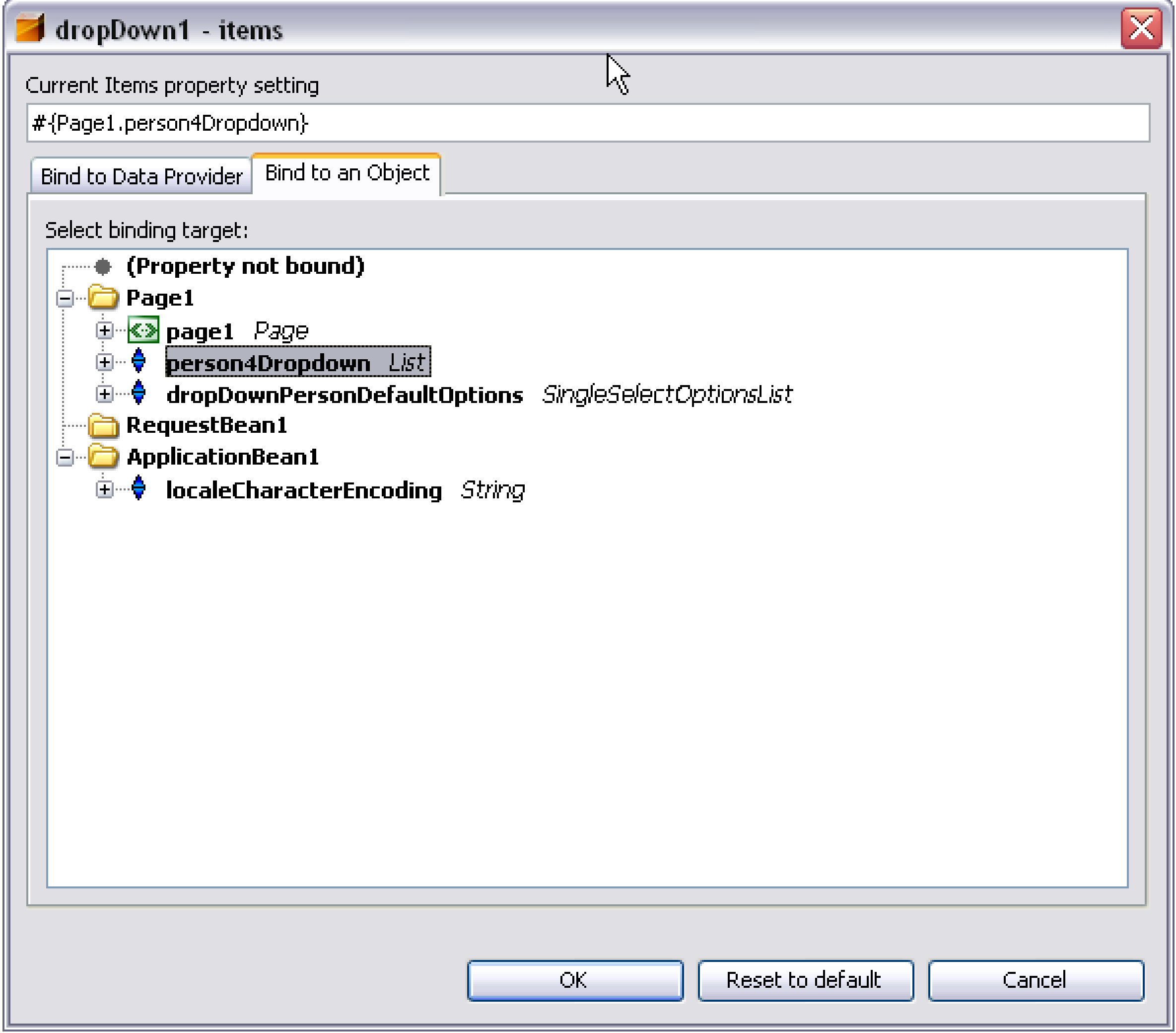
Passons en mode java et rajoutons une méthode getPerson4Dropdown():

```
    /**
     * Return a list of options describing a dropdown elements. The value of the option is an
Integer, while the text
     * displayed shall be the name of the person.
     */
    public List getPerson4Dropdown() {
        List returnValue = new ArrayList();
        for (Iterator it = this.getSessionBean1().getPersonTreeSet().iterator(); it.hasNext();) {
            Person elem = (Person) it.next();
            Integer id = new Integer(elem.getId());
            String name = elem.getName();


            // creating option: value = id, label = name
            Option option = new Option(id, name);


            // adding the new option to the list
            returnValue.add(option);
        }
        return returnValue;
    }
```

Désormais, il reste à lier cette méthode au dropdown. En mode design, sélectionnons le dropdown, puis cliquons sur les ... de l'attribut items. Nous pouvons désormais lier (en anglais "to bind") les éléments de notre dropdown

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

à la propriété person4Dropdown de la classe Page1. Notez que le VWP avait pris le soin de créer une propriété dropDownPersonDefaultOptions. Dès que nous affecterons le dropdown à notre propre méthode, cette propriété "par défaut" sera supprimée automatiquement.



Lions notre dropdown

Il est désormais possible de tester notre dropdown, simplement en cliquant sur l'icône Run: Tomcat se lance, et notre page apparaît:

Premier aperçu de notre page

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

V - Remplir une table

Maintenant, rajoutons une table sur notre page web. Appelons la "tableTrips", et modifions son title en "Voyages".

V.A - Créer un dataProvider pour nos données

Pour pouvoir afficher nos données dans une table, il faut utiliser un objet dataProvider. Le VWP est livré avec plusieurs types de dataProviders, il suffit d'étendre celui qui nous intéresse. Ainsi, créons une nouvelle classe TripArrayDataProvider qui étend ObjectArrayDataProvider: Menu File | new File | Java Classes | Java Class | name = TripArrayDataProvider, package = com.developpez.dejardin.nodb. Modifions le fichier comme suit:

```
package com.developpez.dejardin.nodb;

import com.sun.data.provider.impl.ObjectArrayDataProvider;

import java.util.Date;


/**
 *
 * @author valere dejardin
 */
public class TripArrayDataProvider extends ObjectArrayDataProvider {
    /** Creates a new instance of TripArrayDataProvider */
    public TripArrayDataProvider() {
        Date date = new Date();
        Trip trip1 = new Trip(1, 2, date, "Paris", "Anvers", "CONF", "Javapolis");
        Trip trip2 = new Trip(3, 4, date, "Paris", "San Francisco", "CONF", "Javaone");
        this.setArray(new Trip[] { trip1, trip2 });
    }
}
```

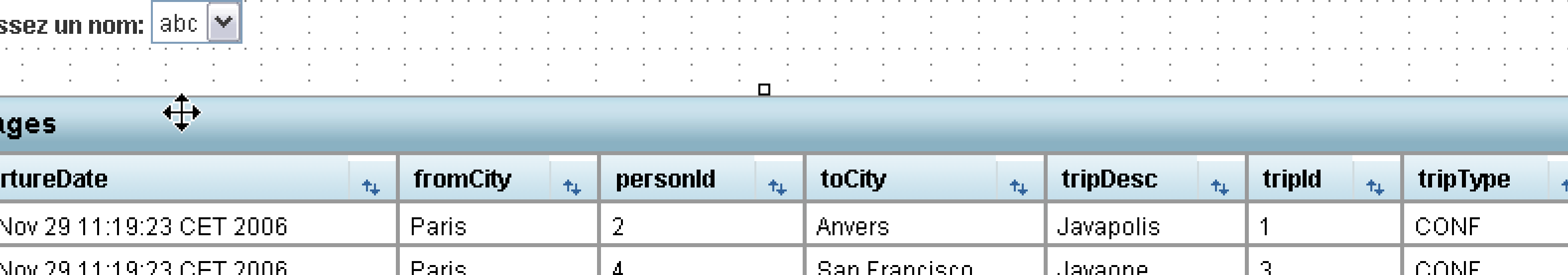
Ensuite, rajoutons une référence à ce nouvel objet dans Page1.java:

```
    private TripArrayDataProvider tripArrayDataProvider = new TripArrayDataProvider();

    public TripArrayDataProvider getTripArrayDataProvider() {
        return tripArrayDataProvider;
    }
```

La manière dont le constructeur est initialisé joue deux rôles: elle permet au VWP d'afficher un look par défaut à une table qui utilise ce dataprovider. Ensuite, le VWP récupère le premier élément du dataprovider et examine ses beans patterns pour déterminer quelles sont les données à afficher. Repassons en mode design, sélectionnons la nouvelle table et affichons sont table layout (clic droit | Table layout). Nous avons alors un dropdown qui nous propose le choix entre notre nouveau tripArrayDataProvider et le DefaultTableDataProvider. Choisissons le tripArrayDataProvider puis OK, la magie du VWP entre en oeuvre!

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Associons notre table au nouveau data provider



Le VWP a reconnu notre objet Trip

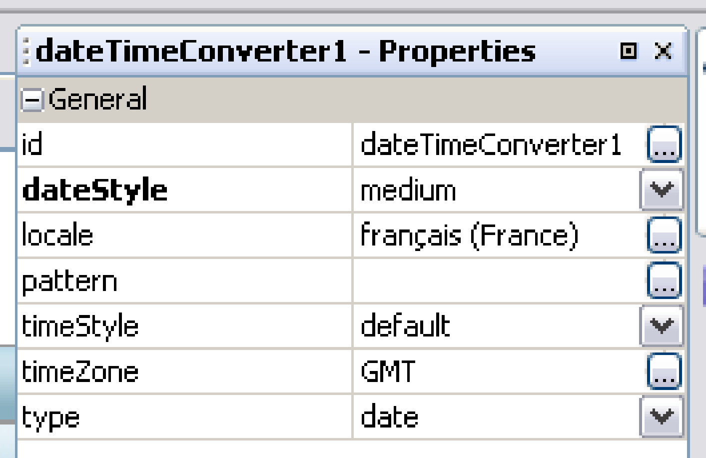
Astuce: la version Technology Preview du VWP présentait un problème à ce niveau: le mode design du VWP ne listait pas le nouveau dataProvider. La méthode pour résoudre ce problème était Menu Build | Clean and Rebuild Main Project, puis fermer et rouvrir le

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

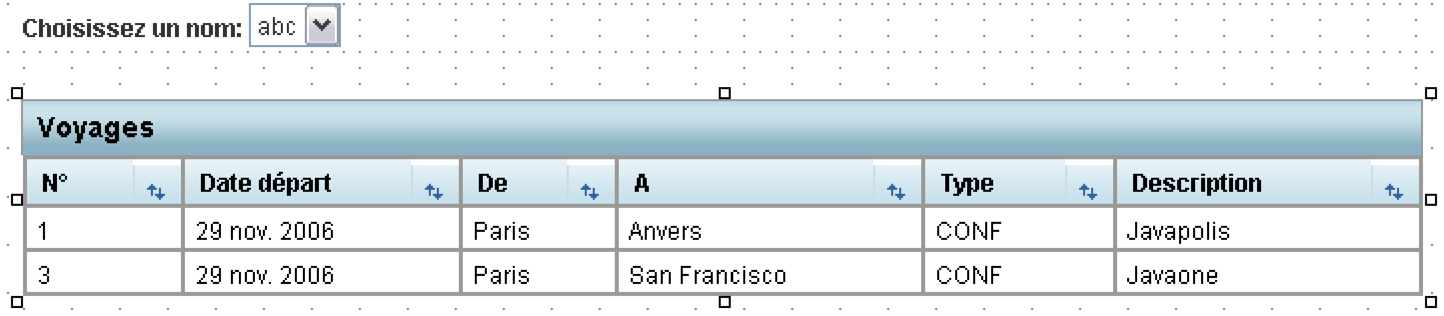
projet. De plus, pour faire un clean du projet il est parfois nécessaire de stopper Tomcat. La version finale qui vient de sortir semble corriger ce défaut, mais vous disposez toujours de cette solution.

Il ne reste plus qu'à faire un peu de mise en page du tableau. Dans le Table Layout, Modifiez les noms des colonnes, et classez- les de manière plus appropriée. Il reste une petite contrariété: la date du départ est affichée dans un format long assez disgracieux. Nous allons la mettre au format français. Sélectionnons, en mode design, la case affichant la date (attention à sélectionner un composant de type *staticField* et non *tableColumn*). Dans le panneau Properties de droite sélectionnons la valeur *(new DateTimeConverter)*.

Un nouveau composant, dateTimeConverter1, a été rajouté à notre page. Sélectionnons le dans le panneau Outline en bas à gauche, et modifions ses propriétés comme suit:



Configuration du dateTimeConverter1



N°	Date départ	De	A	Type	Description
1	29 nov. 2006	Paris	Anvers	CONF	Javapolis
3	29 nov. 2006	Paris	San Francisco	CONF	Javaone

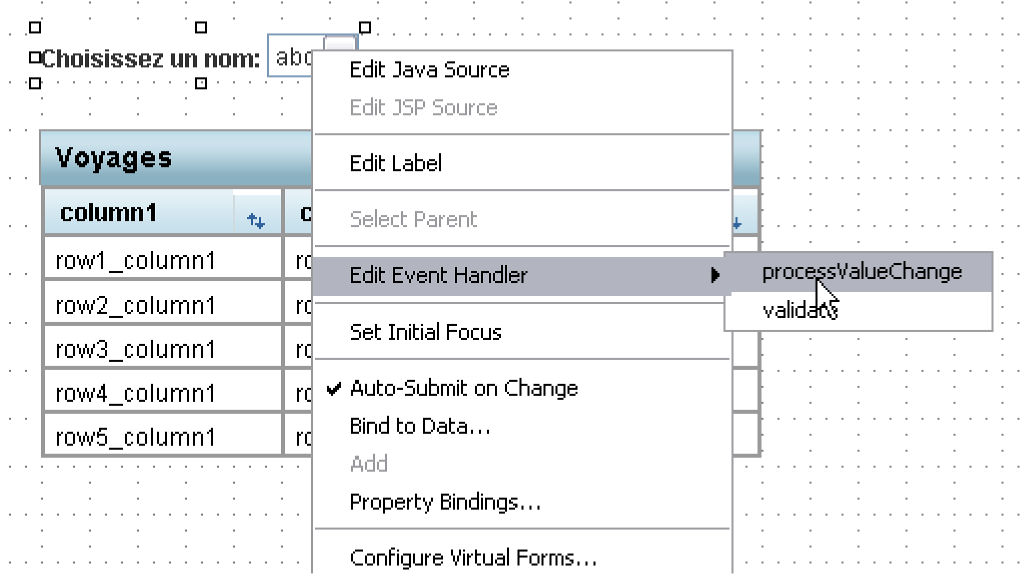
Le résultat final pour notre table en mode design

V.B - Récupérer les données de voyages relatives à une personne.

A cette étape, nous avons une jolie table affichée en mode design, mais si nous l'exécutons, la table restera affichée dans le navigateur avec les valeurs par défaut, ce qui n'est pas trop le but souhaité. Il faut donc remplir proprement le dataProvider.

Toujours en mode design, sélectionnons le dropdown, clic droit et validons "Submit value on change". Puis de nouveau clic droit | Edit Event Handler | Process Value Change. Nous passons en mode java et le VWP nous rajoute une méthode dropDownPerson_processValueChange().

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.



Créons la méthode qui va gérer les évènements sur le dropdown

Rajoutons le code suivant dans le corps de cette méthode:

```
    public void dropDownPerson_processValueChange(ValueChangeEvent event) {
        String value = (String) dropDownPerson.getValue();
        Integer iValue;

        if (value == null) {
            // if value is null, then we get the first value of the persons collection
            iValue = new Integer(((Person)
 this.getSessionBean1().getPersonTreeSet().first()).getId());
        } else {
            iValue = new Integer(Integer.parseInt(value));
        }

        // we get the treeset of trips patchin the given personId
        Set set = (Set) this.getSessionBean1().getTripHashtable().get(iValue);
        Trip[] trips2Display = (Trip[]) set.toArray(new Trip[0]);

        // we store the array and the personId into the session for further use
        this.getSessionBean1().setTrips2display(trips2Display);
        this.getSessionBean1().setCurrentPersonId(iValue);

        // we update the dataprovider with the new array
        tripArrayDataProvider.setArray(trips2Display);
    }
```

Vous pouvez constater que cette méthode va stocker le tableau des voyages à afficher dans le bean de session, ainsi que la référence à la valeur actuelle du dropdown. Nous procédons de cette manière pour simplifier la navigation lorsque plusieurs pages sont impliquées. Il faut donc rajouter quelques lignes de code dans SessionBean1.java:

```
    private Trip[] trips2display = null;

    private Integer currentPersonId = null;

    public Trip[] getTrips2display() {
        return trips2display;
    }
```


Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
    public void setTrips2display(Trip[] trips2display) {
        this.trips2display = trips2display;
    }

    public Integer getCurrentPersonId() {
        return currentPersonId;
    }

    public void setCurrentPersonId(Integer currentPersonId) {
        this.currentPersonId = currentPersonId;
    }
```

Il reste simplement à initialiser le tableau lors du premier affichage. Profitons-en également pour vérifier si les données n'ont pas été modifiées avec de nouvelles versions des fichiers de référence. Dans init() de Page1.java rajoutons les lignes suivantes.

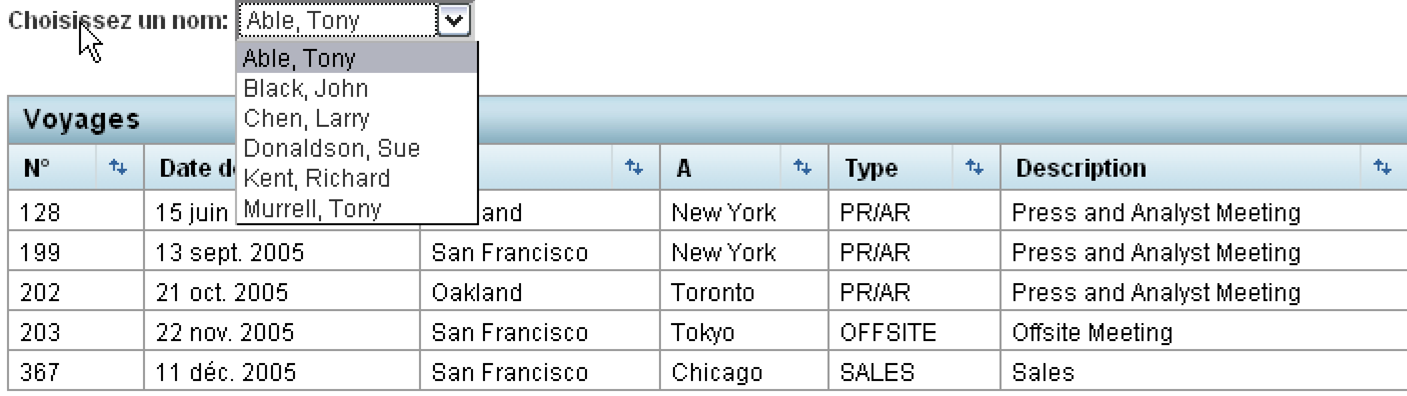
```
        // Perform application initialization that must complete
        // *after* managed components are initialized
        // TODO - add your own initialization code here
        this.getSessionBean1().checkPerson();
        this.getSessionBean1().checkTrip();
        tripArrayDataProvider.setArray(this.getSessionBean1().getTrips2display());
```

Et dans prerender, rajoutons ce qui suit.

```
    public void prerender() {
        String person = (String) dropDownPerson.getValue();
        if (person == null) {
            if (this.getSessionBean1().getCurrentPersonId() == null) {
                // this is the fisrt call to the page, nothing is initialized
                Integer personId = new Integer(((Person)
  this.getSessionBean1().getPersonTreeSet().first()).getId());
                Set tripSet2Display = (TreeSet)
  this.getSessionBean1().getTripHashtable().get(personId);
                Trip[] trips2Display = (Trip[]) tripSet2Display.toArray(new Trip[0]);

                // we store the array into the session for further use
                this.getSessionBean1().setTrips2display(trips2Display);
                this.getSessionBean1().setCurrentPersonId(personId);
                dropDownPerson.setValue(personId.toString());

                // we update the dataprovider with the new array
                tripArrayDataProvider.setArray(trips2Display);
            } else {
                // we are arriving here from another page (Page2?) The dropdown default value has
  to be set with
                // the session stored value.
                Integer personId = this.getSessionBean1().getCurrentPersonId();
                dropDownPerson.setValue(personId.toString());
            }
        }
    }
```

Nous pouvons désormais exécuter notre page pour avoir le résultat suivant:

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.



La table affiche bien les informations souhaitées

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

VI - Rajouter une seconde page et de la navigation

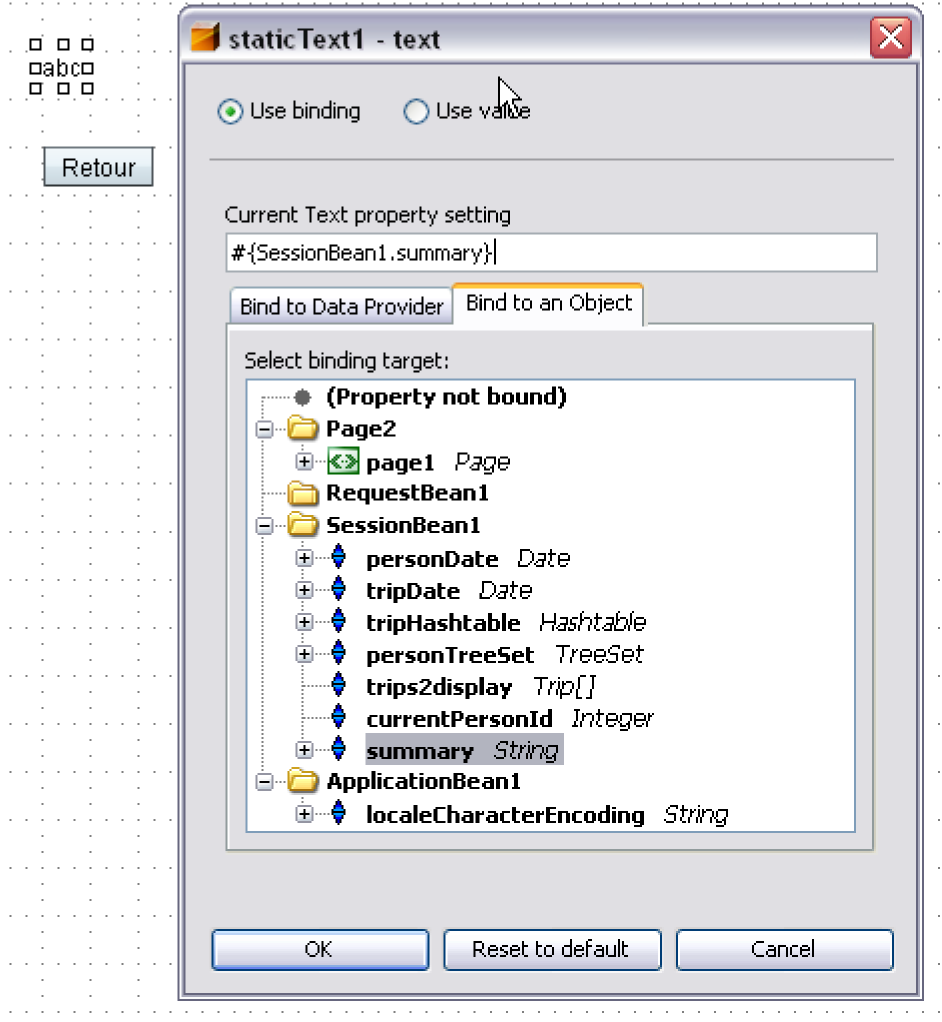
A partir de notre Page1, nous allons rajouter une seconde page. Ici, il faut faire exactement comme dans le cas traditionnel avec une base de données. Cette navigation peut parfois poser problème et on peut se retrouver avec des résultats surprenants (la Page2 affiche les données par défaut du ObjectArrayDataProvider, ou bien lorsqu'on revient à la page de départ la dernière valeur sélectionnée du dropdown a été oubliée). Cette Page2 est donc, si on veut, une preuve de la validité de l'architecture.

Commençons par créer cette Page2, de manière très simple: Menu File | New | Web Page | Page. Sur cette page je rajoute simplement un objet StaticText et un bouton auquel j'attribue la valeur "Retour". Puis retournons dans SessionBean1.java pour rajouter un champ texte:

```
private String summary = null;

   public String getSummary() {
       return summary;
   }

   public void setSummary(String summary) {
       this.summary = summary;
   }
```

Nous allons lier le texte affiché dans notre staticText à ce tout nouveau champ: clic droit sur le staticText | Bind to Data | Bind to an object | SessionBean1 | Summary.

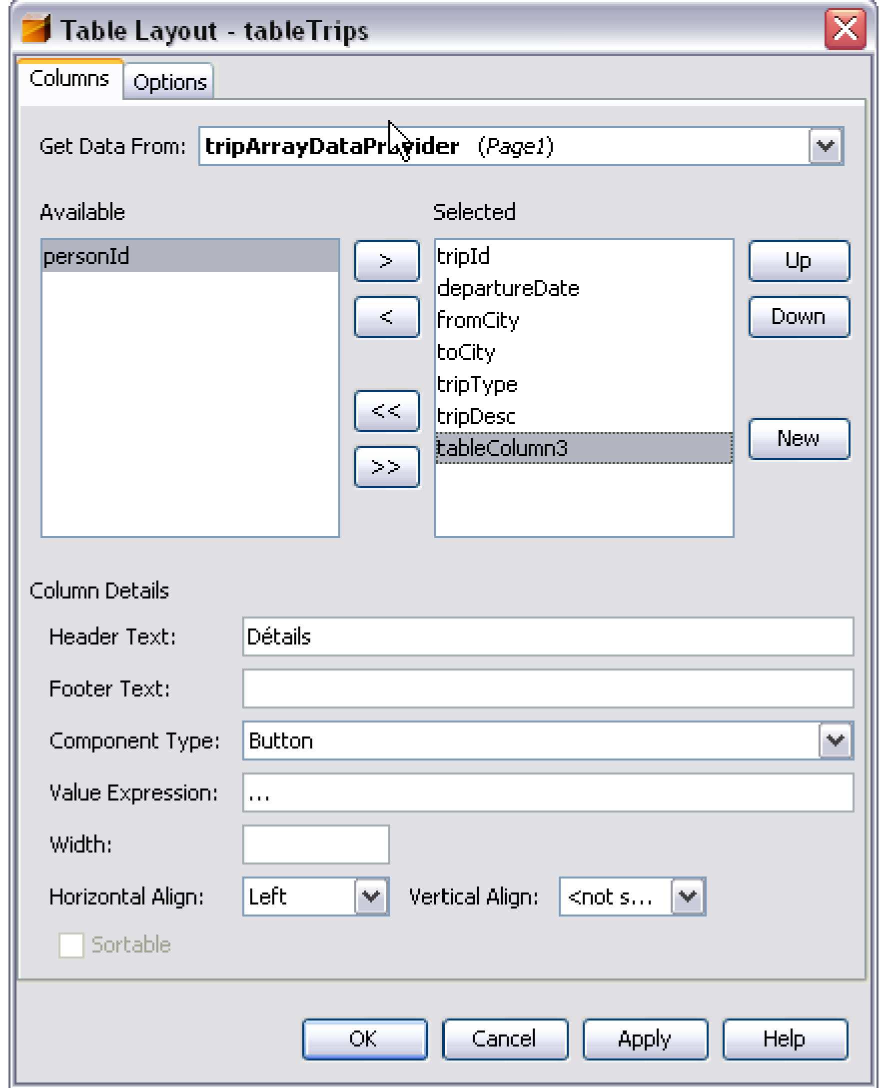
Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.



La Page 2

Cette page est finie, retournons dans Page1. En effet, si nous utilisons la méthode getSummary(), il faut bien que initialiser cette valeur avec un setSummary() quelque part. Ouvrons le Table Layout de notre tableau, rajoutons une colonne. Nous l'appelons Details, réglons son type à "Button" et son texte à "..." (par exemple).

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.



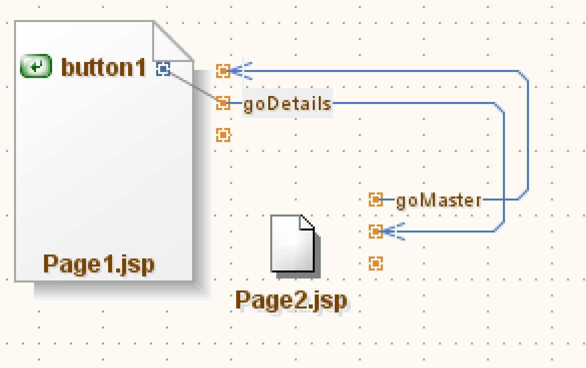
Ajoutons une colonne de boutons à notre table

Après avoir validé la fenêtre, la colonne de boutons apparaît bien. Sélectionnons le bouton (pas la colonne), clic droit | Edit Action. L'affichage passe en mode Java et la méthode button1_action() est créée. Nous la remplissons comme suit:

```
public String button1_action() {
    String from = (String) getValue("#{currentRow.value['fromCity']}");
    String to = (String) getValue("#{currentRow.value['toCity']}");
    this.getSessionBean1().setSummary("De " + from + " à " + to);
    return null;
}
```

Il ne reste plus qu'à valider la navigation. Dans l'onglet Projects, double clic sur Navigation. Cliquons sur Page1, et tirons un trait de button1 vers la Page2. Appelons cette règle de navigation goDetails. De même, créons la règle goMaster entre le bouton de la Page2 et la Page1. Nous pouvons remarquer que la méthode button1_action() de Page1 ne retourne plus null, mais "goDetails".

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.



La fenêtre Navigation

Et hop! un dernier "run" pour afficher le résultat final!

De San Jose à Grenoble

Retour

Le résultat final

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

VII - Conclusion

Cet article vous a montré plusieurs fonctionnalités de base du Visual Web Pack de Netbeans, telles que l'accès et la modification de données de la page JSP depuis la partie Java, l'usage d'un DateTimeConverter ou du TableLayout. Mais surtout, nous somme allés plus loin dans l'usage du VWP pour l'adapter véritablement à nos données: Créer un dropdown dynamiquement, et surtout utiliser les possibilités offertes par les dataProviders.

Cette approche manuelle nous a permis d'appréhender davantage le fonctionnement du VWP, et de mettre en évidence à la fois sa puissance et sa souplesse.

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

VIII - Sources et Crédits

De nombreuses ressources m'ont permis depuis plusieurs années de comprendre et maîtriser Java Studio Creator puis le VWP. Je citerais les **tutoriaux Creator**, le **forum Creator**, et depuis la sortie en Technology Preview du VWP **la mailing list nbusers** de Netbeans.

Winston Prakash publie un **blog** très intéressant sur les possibilités de Creator, et il explique d'ailleurs dans deux de ses billets l'utilisation des ObjectListDataProviders et ObjectArrayDataProvider, mais j'ai retrouvé ces deux billets (relativements vieux) *après* avoir rédigé le brouillon de cet article. Le declic qui m'a permis de réaliser ce tutorial est venu d'un message parmi d'autres sur nbusers expliquant comment construire ses dataproviders.

Merci à Ioan et Ricky81 pour leurs conseils et vbrabant pour la motivation!

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Copyright © 2006 - Valere Dejardin. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.