Annexe 1 : Extrait d'un dossier de demande de branchement

EDF

DEMANDE DE BRANCHEMENT ÉLECTRIQUE

Date de la demande : .. / .. / ....

Cadre réservé à nos services

Référence Dossier :		**ZEI de rattachement :**	

NOM : ..... **PRÉNOM :** .....

COORDONNÉES ACTUELLES

ADRESSE : .....
.....
TÉL : .....

ADRESSE DES TRAVAUX

RUE : .....
NOM LOTISSEMENT : .....
COMMUNE : .....

TYPE DE LA DEMANDE

☐ Neuf	☐ Provisoire	☐ Modification
Date emménagement .. / .. /....	**Date début :** .. / .. / .... **Date fin** : .. / .. /	

POUR UN BRANCHEMENT NEUF :

COORDONNEES DES PROFESSIONNELS QUI VOUS ACCOMPAGNENT DANS VOTRE PROJET DE CONSTRUCTION

VOTRE ÉLECTRICIEN

NOM : .....
ADRESSE : .....
TEL : .....

VOTRE MAÎTRE D'ŒUVRE (éventuellement)

NOM : .....
ADRESSE : .....
TEL : .....

SIGNATURE

Annexe 2 : Extrait de la description des différentes opérations

EDF **NOMENCLATURE DES OPÉRATIONS (Mise à jour au 25/01/2007)**

AD	OPÉRATIONS ADMINISTRATIVES	Poids
AD1	**Front-Office**	
AD11	Demande de dossier d'un client	10
AD12	Demande d'informations au client	10
AD13	Réclamation d'un client	15
AD14	Demande RDV du client	15
AD2	**Back-Office**	
AD21	Envoi dossier vierge	10
AD22	Demande complément d'information	10
AD23	Traitement relance client	15
AD24	Confirmation d'enregistrement	10
AD3	**Traitement dossier**	
AD31	Étude branchement	45
AD32	Planification des travaux	45
...	...	...
IN	**OPÉRATIONS TECHNIQUES**	Poids
IN1	**Branchement électrique**	
IN11	Récupération ligne	45
IN12	Pose ligne	45
IN13	Dépose ligne	25
...	...	
IN2	**Contrôles techniques**	
IN21	Test branchement	45
IN22	Vérification compteur	30
...	...	...

La sous-catégorie « Traitement dossier », codifiée **AD3** est la troisième sous-catégorie de la catégorie « Opérations administratives » codifiée **AD**.

L'opération « Planification des travaux », codifiée **AD32** est la deuxième opération de la sous-catégorie « Traitement dossier ».

Annexe 3 : Liste des différentes ZEI (zones élémentaires d'intervention)

Le département est découpé en **ZEI**. Une ZEI correspond à un secteur autour d'une ville principale.

Code ZEI	NOM VILLE
BA	BAyeux
CA	CAen
CG	CabourG
CH	CHeux
CO	COnde sur Noireau
DO	DOuvres
DV	DeauVille
FA	FAlaise
FM	Fontenay le Marmion

Code ZEI	NOM VILLE
HO	HOnfleur
IS	ISigny
LI	LIsieux
MO	MOult
OR	ORbec
SP	St Pierre sur Dives
TR	TRoarn
VI	VIre

Annexe 4 : Extrait du schéma relationnel de la base « Gestion des rendez-vous »

Les tables décrites ci-dessous ne permettent pas de répondre à tous les besoins de l'application, mais elles suffisent pour répondre aux questions posées dans le dossier concerné.

SOUS_TRAITANT (code, nom)
code : clé primaire

Représente tous les sous-traitants.

CONTRAT (numero, codeSousTraitant, nbTotRDVPris)
numero : clé primaire
codeSousTraitant : clé étrangère en référence à code de SOUS_TRAITANT

Remarque :
- *nbTotRDVPris* : nombre total de rendez-vous pour l'année en cours

Représente tous les contrats passés avec les sous-traitants pour l'année en cours.

PLANNING (dateJournee, numeroContrat, chargeMAT, chargeAPM)
dateJournée, numeroContrat : clé primaire
numeroContrat : clé étrangère en référence à numero de CONTRAT

Remarques :
- *chargeMAT* correspond à la charge de travail affectée le matin ; elle est initialisée à zéro au moment de la création et ne peut dépasser 240 minutes.
- *chargeAPM* correspond à la charge de travail affectée l'après-midi ; elle est initialisée à zéro au moment de la création et ne peut dépasser 240 minutes.

Représente toutes les journées d'intervention planifiées à l'avance dans le cadre des contrats passés pour l'année en cours.

Exemple : Si le mardi et le mercredi sont les jours d'intervention possibles dans le cadre du contrat N, la table PLANNING contient, pour ce contrat, autant de lignes que de mardis et de mercredis dans l'année en cours.

AFFECTER (codeZEI, numeroContrat)
codeZEI, numeroContrat : clé primaire
numeroContrat : clé étrangère en référence à numero de CONTRAT

Représente toutes les affectations de ZEI à chaque contrat de l'année en cours.

Annexe 5 : Classe Champs, structure de la table RDV et classe GèreRDV

> Classe Champs

La classe *Champs* est destinée à la gestion d'un ensemble de champs d'une ligne de table relationnelle (nom, valeur). Elle fonctionne globalement comme une collection de champs.

```
Classe Champs
Privé  ...
        nbChamps : entier
                // Nombre de champs mémorisés
Public ...
        procédure ajouter(unNom : chaîne, uneValeur : chaîne)
                // Ajoute un champ de nom unNom et de valeur uneValeur
        fonction getNbChamps() : entier
                // Renvoie le nombre de champs mémorisés.
        fonction getNom(unIndex : entier) : chaîne
                // Retourne le nom du champ mémorisé à l'index unIndex.
                // Le premier champ est à l'index 0.
        fonction getValeur(unIndex : entier) : chaîne
                //Retourne la valeur du champ mémorisé à l'index unIndex.
                // Le premier champ est à l'index 0.
        procédure vider()
                // Enlève l'ensemble des champs mémorisés.
Fin Classe
```

Exemple d'utilisation :

```
lesChamps : Champs
lesChamps ← new Champs()
lesChamps.ajouter("ref","P01")
lesChamps.ajouter("désignation","souris")
lesChamps.ajouter("prix","12.5")
afficher (lesChamps.getNbChamps())              // affiche 3
nomDuChamp, valeurDuChamp : chaîne
nomDuChamp ← lesChamps.getNom(1)
valeurDuChamp ← lesChamps.getValeur(1)
afficher (nomDuChamp, " : ", valeurDuChamp)     // affiche désignation : souris
```

> Structure de la table RDV

numRdv :	entier
nomClient :	chaîne
adresseClient :	chaîne
telClient :	chaîne
codeZei :	chaîne
numeroContrat :	entier
dateRdv :	date
chargeRdv :	entier
plageHoraire :	chaîne

> **Classe GèreRDV**

Cette classe permet de mettre à jour la base de données du centre de Douvres. Elle opère les ajouts, modifications et suppressions sur la table RDV.

```
Classe GèreRDV
Privé ...
        fonction getType(nomChamp : chaîne) : caractère
                // Retourne un caractère indiquant le type du champ (C,N ou D).
        fonction valeurFormatée(nomChamp : chaîne, valeurChamp : chaîne) : chaîne
                // Retourne la valeur correctement formatée en fonction du type de champ.
        procédure execSql(sql : chaîne)
                // Exécute l'instruction SQL insert, update ou delete passée en paramètre.
Public
        gèreRDV(chaineConnexion : chaîne)
                // constructeur, permet entre autres de se connecter au SGDB en utilisant la
                // chaîne de connexion passée en paramètre.
        procédure supprimer(numéro : chaîne)
                // Supprime dans la table RDV le RDV dont le numéro est passé en paramètre.
        procédure ajouter(numéro : chaîne, lesChamps : Champs)
                // Ajoute une ligne dans la table RDV. Le paramètre lesChamps regroupe, dans l'ordre,
                // l'ensemble des champs de la table RDV, à l'exception du numéro passé dans le
                // 1er paramètre.
        procédure modifier(numéro : chaîne, lesChamps : Champs)
                // Modifie une ligne dans la table RDV. Le paramètre lesChamps contient uniquement les
                // champs qui doivent être modifiés pour le rendez-vous dont le numéro est passé dans le
                // premier paramètre.
Fin Classe
```

Description de la méthode *supprimer* de la classe GèreRDV

```
procédure supprimer(numéro : chaîne)
                // Supprime le RDV dont le numéro est passé en paramètre.
Début
        requête : chaîne
        requête ← "delete from RDV where numRdv="
        requête ← requête + valeurFormatee("numRdv", numéro)  // + : concaténation
        execSql(requête)
fin
```

Exemple d'utilisation :

```
        gRdv : GèreRDV
        gRdv ← new GèreRDV("Provider=interbase;BD=planning")
        gRdv.supprimer("1215")
                        // Supprime le RDV n° 1215 de la base du centre de Douvres.
        lesChamps : Champs
        lesChamps ← new Champs()
        lesChamps.ajouter("dateRdv", "10/04/2007")
        lesChamps.ajouter ("chargeRdv", "45")
        gRdv.modifier("1230", lesChamps)
                // Utilise les informations contenues dans le paramètre lesChamps pour mettre à jour le
                // RDV n° 1230. Cette instruction modifie donc les champs dateRdv et chargeRdv.
```

Annexe 6 : Terminologie XML, classe NoeudXml et classe DocXml

> Terminologie XML utilisée

Soit le fichier XML suivant :

```
<catalogue>
	<produit ref="P01">
		<désignation>souris</désignation>
		<prix>12.5</prix>
	</produit>
</catalogue>
```

- Un nœud XML est un élément ou un attribut.
- La racine du document XML ci-dessus est le nœud *<catalogue>*. Il s'agit d'un élément, son nom est *catalogue*, sa valeur est *vide*. Cet élément possède un élément fils (le noeud *<produit>*).
- Le nœud *<produit>* est un élément. Son nom est *produit*, sa valeur est vide. Il possède un attribut (*ref="P01"*) et deux éléments fils (*<désignation>* et *<prix>*).
- Le noeud *ref="P01"* est un attribut qui a pour nom *ref* et pour valeur *P01*.
- Le nœud *<désignation>* est un élément de nom *désignation* et de valeur *souris*.
- Le nœud <prix> est un élément de nom *prix* et de valeur *12.5*.

> Classe NoeudXml

Cette classe représente un élément ou un attribut XML.

Classe NoeudXml
Privé
	nom, valeur : chaîne *// nom et valeur du nœud.*
	...
Public
	fonction getNom() : chaîne *// retourne le nom du nœud XML.*
	fonction getValeur() : chaîne *// retourne la valeur du nœud XML.*
	fonction nbFils() : entier
		// Retourne le nombre d'éléments fils du nœud courant s'il s'agit d'un élément XML.
		// Retourne -1 s'il s'agit d'un attribut XML.
	fonction getFils(unIndex : entier) : NoeudXml
		// Retourne l'élément fils d'index unIndex si le nœud courant est un élément XML.
		// Le premier élément fils est à l'index 0.
		// Retourne null si le nœud courant est un attribut XML.
	fonction nbAttributs() : entier
		// Retourne le nombre d'attributs XML du nœud courant s'il s'agit d'un élément XML.
		// Retourne -1 s'il s'agit d'un attribut XML.
	fonction getAttribut(unIndex : entier) : NoeudXml
		// Retourne l'attribut XML d'index unIndex si le nœud courant est un élément XML.
		// Le premier attribut est à l'index 0.
		// Retourne null si le nœud courant est un attribut XML.
	...
Fin Classe

> **Classe DocXml**

Cette classe permet de parcourir un document XML après l'avoir chargé en mémoire.

Classe DocXml
Privé
    ...
Public
    DocXml()
        // constructeur
    procédure charger(nomFichier : chaîne)
        // charge l'objet DocXml à partir du fichier nomFichier.
    fonction racine() : NoeudXml
        // Retourne la racine du document XML.
    ...
Fin Classe

Exemple d'utilisation pour le parcours d'un document XML (pdts.xml) :

```
<catalogue>
    <produit ref="P01">
        <désignation>souris</désignation>
        <prix>12.5</prix>
    </produit>
    <produit ref="P02">
        <désignation>clavier</désignation>
        <prix>20</prix>
    </produit>
</catalogue>
```

Le tableau ci-dessous montre la trace de l'exécution d'une série d'instructions.

Instruction	**pdt.nom**	**pdt.valeur**	**att.nom**	**att.valeur**	**nd.nom**	**nd.valeur**
rac,pdt,att,n : NoeudXml						
doc : DocXml						
...						
doc ← new DocXml()						
doc.charger("pdts.xml")						
rac ← doc.Racine()						
pdt ← rac.getFils(0)	**produit**					
att ← pdt.getAttribut(0)	produit		**ref**	**P01**		
nd ← pdt.getFils(0)	produit		ref	P01	**désignation**	**souris**
nd ← pdt.getFils(1)	produit		ref	P01	**prix**	**12.5**

Annexe 7 : Exemple de fichier modifsRdv.xml et début du programme majTableRdv

> Fichier *modifsRdv.xml*

```
<infosRdv>
	<rdv action="ajout">
		<numRdv>1245</numRdv>
		<nomClient>Dubois</nomClient>
		<adresseClient>12 rue des fleurs 14000 Caen</adresseClient>
		<telClient>0429784529</telClient>
		<codeZei>CA</codeZei>
		<numeroContrat>287</numeroContrat>
		<dateRdv>12/04/2007</dateRdv>
		<chargeRdv>50</chargeRdv>
		<plageHoraire>mat</plageHoraire>
	</rdv>
	<rdv action="modif">
		<numRdv>1230</numRdv>
		<dateRdv>10/04/2007</dateRdv>
		<chargeRdv>45</chargeRdv>
		<plageHoraire>apm</plageHoraire>
	</rdv>
	<rdv action="supp">
		<numRdv>1215</numRdv>
	</rdv>
</infosRdv>
```

L'exécution du programme majTableRDV avec ce fichier aura pour effet :
- d'insérer le RDV 1245 (action="ajout"),
- de modifier le RDV 1230 (action="modif"),
- de supprimer le RDV 1215 (action="supp").

Remarques :
- Dans un élément *<rdv action="ajout">* ou *<rdv action="modif">*, les éléments fils apparaissent toujours dans le même ordre que les champs de la table RDV.
- L'élément fils *numRdv* est présent dans tous les éléments *<rdv>*, quelle que soit l'action.
- Le contenu du fichier *modifsRdv.xml* est automatiquement contrôlé lors de sa création et peut donc être considéré comme fiable : il contient toujours au moins une action et chaque action, selon son type, comporte au moins la valeur d'un champ.

> Programme *majTableRdv*

```
Programme majTableRdv
début
	doc: DocXml
	doc ← new DocXml ()
	gRdv : GèreRDV
	gRdv ← new GèreRDV("Provider=interbase;BD=planning")
	racine : NoeudXml
	doc.charger("modifsRdv.xml")
	racine ← doc.racine()
	...
fin
```