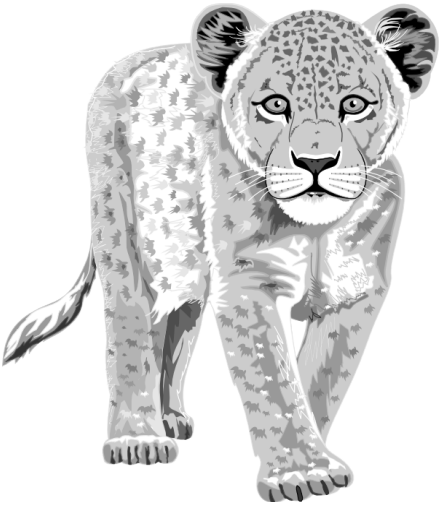
LAZARUS FREE PASCAL
Développement rapide
Version libre allégée

Programmation



ISBN 9782953125160

Éditions LIBERLOG
Éditeur n° 978-2-9531251
13 Rue Tanguy PRIGENT
35000 RENNES

Sommaire

A)A LIRE

Des mots que vous ne comprenez pas, avec leur première lettre en majuscule, peuvent être trouvés dans le glossaire à la fin du livre.
Les exemples peuvent se télécharger sur le site www.liberlog.fr.

Les mots en gras indiquent des chapitres.
Les mots en majuscules indiquent des groupements, des langages, des outils, des entités, des marques.
Les mots entre guillemets indiquent des mots de LAZARUS, des mots à chercher sur le Web, ou des mots de votre environnement.
Les mots entre crochets indiquent des mots à remplacer.

A)OBJECTIFS DU LIVRE

L'objectif du livre est d'apprendre facilement LAZARUS afin de créer rapidement un Logiciel. On met en avant la création de Composants, étape indispensable dans l'automatisation et l'optimisation. On donne la démarche pour créer très rapidement un Logiciel. Cette démarche consiste à éviter les copiés-collés.

Je mets en avant une nouvelle méthode dans ce livre. Le Développement Très Rapide d'Applications est l'aboutissement de tout savoir-faire automatisant la création de Logiciel. On prépare la création pour sauter une étape : La création du Logiciel à partir de l'analyse. C'est l'analyse qui crée le Logiciel grâce à votre moteur DTRA.

B) LICENCE

Le livre complet a été écrit par Matthieu GIROUX et Jean-Michel BERNABOTTO pour le langage PASCAL. Il est sous deux licences : CREATIVE COMMON BY SA et CREATIVE COMMON BY NC-ND. Vous disposez ici de quelques chapitres libres.

Vous pouvez utiliser ce livre en citant l'auteur.

C) DU MÊME AUTEUR

- L'astucieux LINUX d'après www.aides-informatique.com
- LAZARUS FREE PASCAL – Développement rapide
- Comment écrire des histoires d'après www.comment-ecrire.fr
- Nos Nouvelles Nos Vies
- Poèmes et Sketchs – De 2003 à 2008

Visibles sur GOOGLE BOOKS.

Disponibles sur www.comment-ecrire.fr, www.lazarus-components.org.
Site d'informations : www.france-analyse.com.

D) DU MÊME ÉDITEUR

- Alors vous voulez tout savoir sur l'économie ?
  Lyndon Larouche

Disponible sur www.france-analyse.com.

B)BIOGRAPHIE

Matthieu GIROUX aime l'écriture, cette recherche de la vérité en utilisant des moments de réflexion.

Il s'intéresse au Développement Rapide d'Applications afin de trouver de meilleures techniques de programmation. Sauter une étape la création du Logiciel est possible grâce à l'automatisation, maître mot de l'informatique dans l'entreprise.

LAZARUS n'est donc qu'une étape dans la réalisation d'un savoir-faire ou Framework. Il faudra ensuite automatiser à partir d'autres frameworks afin de mieux faire connaître le sien. Recherchez le partage pour vous enrichir et enrichir les autres.

C)LAZARUS FREE PASCAL

A)POURQUOI CHOISIR LAZARUS ?

Si vous voulez apprendre la programmation d'interfaces homme-machine LAZARUS est fait pour vous. LAZARUS c'est un EDI, un Environnement de Développement Intégré, disponible sur un maximum de machines, compatible DELPHI. DELPHI est beaucoup enseigné dans les écoles françaises. Nous souhaitons le même destin à LAZARUS qui ne possède pas de licence payante. LAZARUS c'est une utilisation des librairies Libres les plus fiables avec une interface facile à utiliser. LAZARUS est un Logiciel Libre.

Vous pouvez adapter des Composants DELPHI pour que vos Logiciels deviennent multiplate-forme. Ils seront donc utilisables sur beaucoup de machines. DELPHI a été créé en 1995. Le projet LAZARUS a commencé en 1999. Il est actuellement préférable d'utiliser LAZARUS plutôt que KYLIX, un DELPHI sous LINUX créé en 2001. En effet KYLIX a été abandonné dès sa troisième version.

Un logiciel Client/Serveur c'est en quelque sorte un logiciel qui n'a pas besoin de navigateur web pour fonctionner. Si vous voulez créer ce genre de Logiciel, vous pouvez compter sur des librairies de composants visuels, permettant de réaliser vos interfaces exécutables, sur vos plates-formes préférées.

Le multiplate-forme consiste à diffuser largement son Logiciel sur un maximum de machines. On utilise LAZARUS pour créer des Logiciels Client/Serveur multiplate-forme.

Si vous voulez créer votre jeu multiplate-forme, LAZARUS possède des librairies permettant de créer des jeux sur 2 Dimensions ou sur 3 Dimensions. On utilise LAZARUS pour créer des Logiciels graphiques multiplate-forme. Il existe des librairies permettant de lire des formats de fichiers 2D ou 3D, Libres ou propriétaires.

Vous pouvez rester de longues heures devant votre ordinateur, à programmer un logiciel répondant à une demande spécifique. Ne vous en faites pas, il existe des écrans n'émettant pas de lumière. Ils permettent de ne pas fatiguer ses yeux, afin de créer vos logiciels en toute sérénité.

B)ARCHITECTURES FREE PASCAL

Avec LAZARUS, en 2010, vous pouvez créer des exécutables fonctionnant sous WINDOWS, sous BSD, sous MAC OS X, sous LINUX, sous DEBIAN, sous UNIX, sous OS X, sous WIN CE, sous I PHONE, sous GPHONE, sous SMARTPHONE. Pour certaines plates-formes il est nécessaire de participer au projet LAZARUS.

Nous utilisons actuellement une architecture 32 bits voire 64 bits. On peut schématiser cela comme 32 ou 64 voies d'autoroutes.

Ainsi une adresse 32 bits peut adresser jusqu'à 2^{32} entiers de 32 bits. Une adresse 64 bits peut adresser jusqu'à 2^{64} entiers de 64 bits. Une adresse 64 bits ne fonctionne pas sur une adresse 32 bits. Par contre une adresse 32 bits peut fonctionner sur une architecture 64 bits, dans la limite de ses possibilités.

LAZARUS fonctionne sur les architectures 32 et 64 bits. Cependant vous devez vérifier que les composants complémentaires fonctionnent en 64 bits. En effet, en 2011, c'est toujours l'architecture 32 bits qui est prioritaire.

c)Applications Libres LAZARUS

Une Application Open Source n'est que partagée. Une Application Libre peut s'utiliser, s'étudier, se distribuer et se modifier grâce au partage.

Il existe des applications multiplate-forme faites avec LAZARUS. LAZARUS est choisi pour le multiplate-forme.

SKYCHART et VIRTUAL MOON sont des applications scientifiques en 3D et Libres.
Différents logiciels de gestion vous permettent de gérer votre café, entreprise ou restaurant.
PEA ZIP est un compresseur/décompresseur Libre concurrent de 7 ZIP.
Il existe des applications Libres pour les données pour PARADOX ou SQLITE.
ORTHONET est un Logiciel éducatif Libre. Des professeurs ont choisi LAZARUS.

Il existe aussi des utilitaires Libres pour la vidéo ou le son, d'autres projets scientifiques.

Des sites Web vous permettent de télécharger pour participer à ces Logiciels Libres.

Vous avez une liste de logiciels utilisant LAZARUS avec ce lien :
http://wiki.lazarus.freepascal.org/Projects_using_Lazarus/fr

d)Du PASCAL orienté Objet

LAZARUS est basé sur un langage nommé FREE PASCAL. FREE PASCAL est aussi un Compilateur PASCAL orienté Objet. Le langage FREE PASCAL possède une grammaire et des instructions PASCAL, basées sur l'algorithmique. Le PASCAL est donc enseigné pour cette raison particulière.

La programmation par Objets est la programmation la plus proche de l'humain. Cette programmation regroupe des parties du programme, créées dans des Objets. On peut ainsi facilement représenter son programme en entités.

Ces entités ont un comportement, une hiérarchie, des valeurs. Le Pascal Objet possède des Objets pouvant être enregistrés comme Composants. Les Composants PASCAL améliorent le Polymorphisme du PASCAL orienté Objet, cette capacité des Objets à accepter plusieurs formes.

Le langage PASCAL Objet est l'un des plus simple du marché. Il permet de trouver facilement les erreurs. Le Compilateur FREE PASCAL permet de créer des exécutables indépendants de toute librairie complémentaire.

LAZARUS a certes été créé en 1999. Mais il ne fait que reprendre les instructions de PASCAL Objet existantes, grâce au Compilateur multiplate-forme FREE PASCAL, point de départ de départ de LAZARUS. Le Compilateur FREE PASCAL évolue maintenant grâce à LAZARUS.

Comme tout outil Objet RAD, LAZARUS est accessible. Il peut cependant vous emmener, petit à petit, vers les Composants. Vous apprenez alors réellement l'Objet, permettant d'améliorer ce que vous utilisez.

Nous vous apprenons à connaître les instructions permettant de réaliser votre Logiciel, adapté à vos besoins. Pour aller plus loin, connaître

l'Objet et les différentes architectures logicielles permet d'améliorer LAZARUS.

E)LA COMMUNAUTÉ

LAZARUS dispose d'une communauté active. Elle ajoute au fur et à mesure de nouvelles possibilités. Des communautés LAZARUS s'ouvrent régulièrement dans d'autres pays.

Il existe un wiki LAZARUS, ainsi qu'un espace de documentation français. Un espace de traduction facilite la compréhension de LAZARUS. Vous pouvez participer à ces espaces, ou donner vos avis.

F)LAZARUS EST PARTAGÉ

Les sites de www.sourceforge.net et code.google.com ne diffusent que des Logiciels Open Source. SOURCEFORGE et GOOGLE CODE permettent en effet de partager vos Sources de Logiciel. Nous parlons de cet aspect dans le chapitre sur la **Collaboration**. Avec les sites web de Sources partagées, d'autres développeurs peuvent améliorer votre Logiciel, car les Sources sont la recette de votre Logiciel.

LAZARUS est donc Open Source. Vous pouvez le modifier. Ce site permet aussi de disposer d'une mise à jour des Sources, fichier par fichier, selon les modifications. Ces fichiers Sources ne sont pour certains pas testés.

G)LES VERSIONS DE LAZARUS

LAZARUS est un Logiciel Libre. Autrement dit vous avez accès aux

Sources, et vous avez le droit de les modifier librement. Vous pouvez donc intégrer le projet LAZARUS.

On a intérêt à mettre à jour LAZARUS, afin de disposer des dernières améliorations Libres. Les développeurs LAZARUS gèrent des numéros de version.

LAZARUS évolue continuellement. Évitez certaines versions.
Le Versioning LAZARUS se présente avec des chiffres, disposés de cette façon :

A.B.CC-D

Le premier nombre nommé "A" présente une version majeure de LAZARUS. Par exemple la version 1.0.00-0 de LAZARUS permettra de traduire entièrement tout Composant DELPHI 6 vers LAZARUS. Plus ce chiffre ou nombre est grand et récent, plus cette version est recommandée.

Le deuxième chiffre ou nombre présente une version mineure de LAZARUS. Plus ce nombre ou chiffre est grand, plus cette version est recommandée.

Le troisième nombre présente la maturité de la version mineure. Si le nombre est pair, la version est stable et utilisable, car elle a été testée.

Si le nombre est impair, il s'agit d'une version non testée, créée un jour donné. Les versions à troisième nombre impair ne veulent rien dire sans leur jour de création. Les versions à troisième nombre impair permettent de montrer un aperçu de la prochaine version paire. Ces versions ne sont jamais recommandées, exceptées quelques jours avant la finition. Avec on a la possibilité de créer des Composants de LAZARUS avec une grammaire FREE PASCAL récente.

Le quatrième chiffre ou nombre montre une création d'une version

stable de LAZARUS, à partir d'une base plus récente. En fait le quatrième chiffre ne vous apporte que la réparation de Bogues ou erreurs trouvées, ce qui est déjà intéressant, rien de nouveau sinon. Si le quatrième chiffre est impair la version de LAZARUS n'a pas été testée. Elle a cependant de grandes chances d'être suffisamment stable.

Vous avez la possibilité de télécharger un "snapshot LAZARUS" WINDOWS de la dernière version déboguée de LAZARUS. Pour les autres plates-formes, les Sources du "snapshot" permettent de créer un nouveau LAZARUS.

H) TÉLÉCHARGER LAZARUS

Vous pouvez donc maintenant télécharger la dernière version de LAZARUS pour votre environnement, sur la page de téléchargement du projet LAZARUS à http://sourceforge.net/projects/lazarus.

Vous pouvez par la même occasion voir l'avancement du projet à http://www.lazarus.freepascal.org, dans la page "Roadmap".
Si vous souhaitez connaître l'état de la prochaine version allez à http://bugs.freepascal.org.

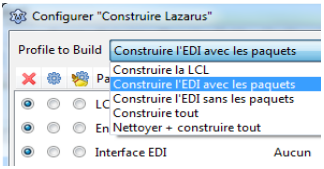
Il est possible de télécharger sa version LAZARUS du jour sur http://www.hu.freepascal.org/lazarus. On télécharge un Snapshot LAZARUS, c'est à dire une vue sur les modifications du jour apportées à LAZARUS. Un "snapshot" sert à travailler sur des Composants LAZARUS en cours de publication. Il compile rarement un exécutable.
Si on souhaite participer à LAZARUS il est possible de télécharger directement les Sources SVN sur le site Web de sourceforge à http://sourceforge.net/projects/lazarus, dans la partie "Develop".

1)Snapshot sur LINUX

Pour construire avec les sources d'un "snapshot" LAZARUS, remplacez les sources de LAZARUS par celles du "snapshot", puis démarrez LAZARUS comme ceci :

```
su root & startlazarus
```

Dans "Options" puis "Configurer Build LAZARUS" "Construire tout".



Options de construction de LAZARUS

Allez dans le menu "Outils" puis "Construire LAZARUS".

Si vous voulez travailler sur les Composants LAZARUS, le mieux est de changer le propriétaire du répertoire LAZARUS vers votre compte. Seul votre compte peut alors construire le noyau LAZARUS.

LAZARUS s'installe dans "/usr/lib/lazarus". Si vous souhaitez travailler souvent sur les Composants, il vous est possible d'accéder à ce répertoire, en devenant Propriétaire du répertoire, grâce au terminal LINUX. Utilisez cette commande en mode Administrateur :

```
chown -R mon_login.root /usr/lib/lazarus
```

I)INSTALLER LAZARUS SOUS WINDOWS

LAZARUS s'installe en vous demandant de relier les fichiers PASCAL ou LAZARUS à votre Environnement de Développement Intégré.

Notez le répertoire d'installation de LAZARUS.

J)INSTALLER LAZARUS SOUS LINUX

LINUX centralise ses paquets de logiciels. Ainsi un logiciel est très léger car il utilise des librairies dans d'autres paquets.
Sachez que LAZARUS utilise des librairies C. Vous devez disposer des librairies disponibles sur LINUX par défaut.

Une version moins récente est disponible dans votre gestionnaire de paquets. Pour disposer de la version plus récente ajoutez le serveur de tests à votre gestionnaire de paquets. Avant vérifiez si la version de www.sourceforge.net est plus récente que celle de votre gestionnaire.

Une version de LAZARUS est dans le gestionnaire de paquets. Il suffit d'installer le paquet "LAZARUS" graphiquement ou en tapant cette commande sur DEBIAN en mode "root" (Administrateur) :

```
apt-get install lazarus
```

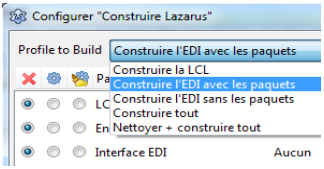
Sinon installez l'ensemble des paquets FREE PASCAL COMPILER débutant par "fpc" et "fp-" ainsi que les paquets LAZARUS débutant par "lazarus". FREE PASCAL COMPILER peut être trop récent. Faites attention à la version reconnue par LAZARUS.

K)CONFIGURER LAZARUS

LAZARUS se recompile presque entièrement par défaut. Allez dans le menu "Outils" puis "Configurer Build LAZARUS".

Choisissez "Construire l'EDI et les paquets". Ainsi vous reconstruisez l'exécutable de LAZARUS en permettant d'installer de nouveaux

Composants.



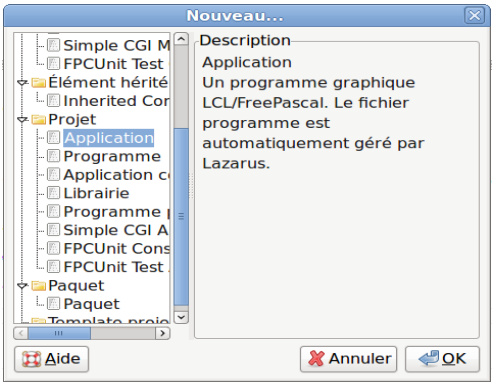
"Construire l'EDI avec les paquets"

"Construire la LCL", "Nettoyer et construire tout" ou "Construire tout" sont à choisir lorsque des Sources LAZARUS en version d'essai ont été installées. Le nettoyage consiste à supprimer les unités compilées. Pensez à modifier le répertoire de LAZARUS dans "Configuration" puis "Options".

D)PROGRAMMER FACILEMENT

A)CRÉER UN LOGICIEL

Après avoir créé un "Nouveau" projet "Application" allez dans l'"Éditeur de Source". Vous voyez une Source créée automatiquement ainsi qu'une fenêtre qui ne réagit pas comme une fenêtre habituelle. C'est votre base de travail permettant de créer un programme fenêtré.



Nouvelle "Application" LAZARUS

Vous voyez aussi une fenêtre, ou un formulaire, avec une grille en pointillés permettant de travailler graphiquement. Allez dans le menu "Projet" puis dans "Inspecteur de Projet".

Vous voyez les fichiers qui ont été créés automatiquement. Le fichier commençant par "Unit1" c'est le formulaire de votre nouveau projet en Source PASCAL Objet.

Vous voyez aussi un fichier ".lpr". Cette extension de fichier signifie

LAZARUS Project Resources, ou Ressources d'un Projet LAZARUS. Ce fichier, liant le projet vers les ressources LAZARUS, permet de compiler et d'exécuter votre formulaire.

Pour compiler et exécuter ce projet vide, cliquez sur ▶ dans la barre d'outils LAZARUS.
Vous voyez la même fenêtre vide sans les pointillés de présentation. C'est votre formulaire qui est exécuté. Vous pouvez le déplacer, l'agrandir, le diminuer et le fermer, contrairement au formulaire précédent.

Il s'agit bien d'un programme exécuté. Pour exécuter ce programme avec ce formulaire le Compilateur FREE PASCAL a d'abord compilé le formulaire grâce à ces Sources et celles de LAZARUS.

LAZARUS c'est un Environnement de Développement Intégré. Vous avez eu peu de programmation à effectuer pour créer ce logiciel.

1)Le nom d'unité

Allez dans l'"Éditeur de Sources". Placez-vous au début de l'unité du formulaire.

Du Code PASCAL Objet commence toujours par la clause "Unit" reprenant lettre pour lettre le nom du fichier PASCAL. Le nom d'unité se change dans "Fichier", puis "Enregistrer sous".

En général chaque instruction PASCAL est terminée par un ";". Il existe de rares exceptions pour lesquelles le Compilateur vous avertit.

Pour la compatibilité UNIX ou LINUX il est préférable de renommer les noms d'unités en minuscules.

2)La clause "uses"

Ensuite vous avez surement à réutiliser des existants. La clause "uses" est faite pour cela. La clause "uses" permet d'aller chercher les existants créés par l'équipe LAZARUS ou bien par un développeur indépendant de LAZARUS.

Lorsque vous ajoutez un Composant grâce à l'onglet des Composants cela ajoute aussi l'unité du Composant dans la clause "uses". Aussi un lien est créé dans le projet vers le regroupement d'unités nommé paquet.

Vous aurez sans doute à ajouter les unités communes LAZARUS comme "LCLType" ou "LCLIntf". Ces librairies remplacent certaines unités de DELPHI comme l'unité "windows".

Pourquoi n'a-t-on pas gardé certaines unités DELPHI ?

Les librairies LAZARUS possèdent une nomenclature. Une librairie possédant le mot "win" est une librairie système spécifique à la plate-forme WINDOWS. Elle ne doit pas être utilisée directement dans vos programmes. Les unités "gtk" servent à LINUX, "carbon" à MAC OS, "unix" à UNIX.

Vous avez un aperçu de ces mots clés dans les "Outils". La configuration de construction de LAZARUS vous montre l'ensemble des plates-formes indiquant les unités à éviter. Dans les Sources de LAZARUS, vous voyez des unités commençant pas ces plates-formes ou comportant ces noms de systèmes.

3)L'Exemple

L'exemple consiste à créer une fenêtre permettant d'afficher "Hello World !".

Nous allons éditer un nouveau projet.

Démarrez LAZARUS.

Allez dans "Fichier" puis "Nouveau". Choisissez "Application".

Tout d'abord nous allons rechercher l'unité "Dialogs" avec beaucoup de fainéantise.

Placez-vous après une virgule de la clause "uses". Comme chaque instruction, Cette dernière se termine par un ";".

En respectant la présentation LAZARUS, tapez uniquement "di".

Ce qui va suivre ne fonctionne que si votre Code Source est correct, du début de l'unité jusqu'à votre ligne éditée.

Appuyez en même temps sur "Ctrl" avec ensuite la barre d'espacement. Relâchez. Il s'affiche une liste d'unités LAZARUS. Choisissez "Dialogs" avec les flèches et la touche "Entrée". N'oubliez pas de placer ensuite une virgule pour ne pas créer d'erreur de compilation. Vous avez ajouté l'unité Dialogs.

Pour voir ce que vous avez ajouté, maintenez enfoncé "Ctrl" tout en cliquant sur l'unité "Dialogs". L'unité "Dialogs" s'affiche. Vous allez vers la Source liée en faisant de cette manière. LAZARUS va chercher ce qu'il connait grâce aux liens de vos projets et un Code correct.

On voit que la clause "uses" de "Dialogs" contient d'autres unités. Celles-ci sont elles aussi ajoutées à votre projet, si elles ne l'étaient pas déjà. En effet des unités obligatoires sont toujours utilisées.

Toute écriture de Code ajoute des informations à votre exécutable, avec en plus des unités qui sont intégrées. Votre programme compilé contient beaucoup de Code, mais il est indépendant.

Grâce aux onglets de l'"Éditeur de Sources", vous pouvez retourner sur votre Source en cliquant sur l'onglet "Unit1".

Ce nom "Unit1" n'est pas explicite. Nous allons le renommer en gardant une partie des informations importantes.

Cliquez dans le menu LAZARUS sur "Fichier" puis "Enregistrer". Créez votre répertoire de projet, puis sauvegardez votre unité en commençant par "u_". Cela indique comme avant que l'on sauvegarde une unité. Les fichiers sont ainsi regroupés au même endroit. Ensuite on met le thème de l'unité : "hello". Cela donne "u_hello". Sauvegardez.

Si vous avez mis des majuscules dans le nom d'unité, LAZARUS peut vous demander de renommer ce nom d'unité en minuscule. N'allez que rarement contre ses recommandations.

Votre nom d'unité est donc "u_hello". Elle peut être rajoutée dans une clauses "uses" d'une autre unité du projet.

La présentation

Nous allons rajouter un bouton à notre formulaire. Pour aller sur le formulaire de l'unité "u_hello" appuyez sur "F12", quand vous êtes sur l'unité dans l'"Éditeur de Source". Cliquez de nouveau sur "F12" permet de revenir sur l'"éditeur de Source". Revenez sur la fiche.

Allez sur la palette des Composants. Sélectionnez dans le premier

onglet "Standard" le bouton "TButton". Cliquez l'icône "TButton". Il s'enfonce.

Cliquez sur le formulaire pour placer votre "TButton". Vous voyez à l'exécution la même fenêtre avec son "TButton" sans les points noirs. Ces points noirs servent à se repérer à la conception.

Cliquez sur le bouton de votre formulaire puis sur "F11". Vous sélectionnez l'"Inspecteur d'Objets", qui permet de modifier votre "TButton".

Vous êtes sur l'onglet "Propriétés" de l'"Inspecteur d'Objets". Cet onglet classe les propriétés par ordre alphabétique.

Vous pouvez changer la propriété "Name" de votre "TButton", afin de renommer votre "TButton" en "BCliquez". Vous voyez alors le texte du "TButton" changer. C'est une automatisation de ce Composant qui change sa propriété "Caption", quand on change pour la première fois son nom.

Pour vous rendre compte, allez sur la propriété "Caption" et enlevez le "B". Le nom de votre Composant n'a pas changé. Pourtant la propriété "Caption" a bien changé le texte de votre "TButton".

Le Code Source

Allez sur l'onglet "Evènements" de l'"Inspecteur d'Objets". Double cliquez sur l'événement "OnClick".

Cela fait le même effet que d'appuyer sur les trois points : Vous créez un lien vers une procédure dans votre "Éditeur de Source".

Tapez entre le "Begin" et le "End;" de cette nouvelle procédure, deux

espaces puis "show". Puis appuyez en même temps sur "Ctrl" suivi de "Espace".

Si vous avez ajouté l'unité "Dialogs" vous voyez la procédure "ShowMessage". Cette procédure simple possède un seul paramètre chaîne.

Sélectionnez cette procédure avec les flèches et entrée. Vous pouvez aussi utiliser la souris.

Vous avez maintenant à ajouter le paramètre permettant d'afficher du texte. Les paramètres des procédures sont intégrés grâce aux parenthèses.

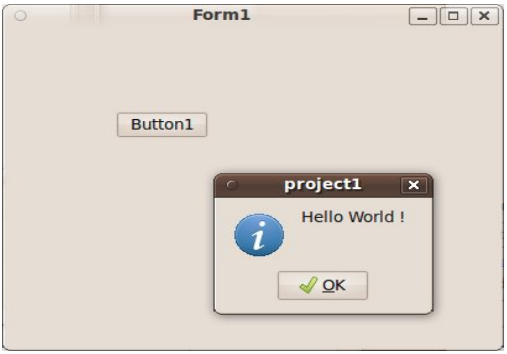
Ouvrez votre parenthèse. Les chaînes sont ajoutées en les entourant par une apostrophe. L'instruction se termine par un ";". Écrivez donc :

```
Begin
  ShowMessage ( 'Hello World' );
End;
```

Nous allons afficher un message "Hello World" quand on clique sur le bouton "BCliquez".

Appuyez simultanément sur "Ctrl" suivi de "F9". Cela compile votre exécutable, en vérifiant la syntaxe et en donnant des avertissements.

Cliquez sur ou appuyez sur "F9" pour exécuter votre Logiciel fenêtre. Après avoir appuyé sur le bouton voici ci-après ce que cela donne :



Un exemple mondial avec un événement

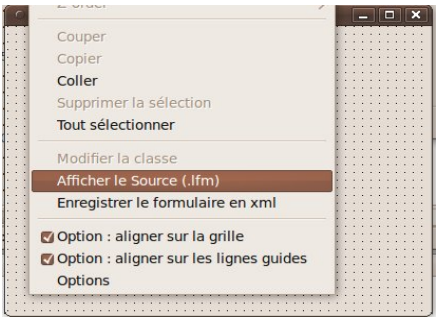
4)La Source du formulaire

Le type égal à "class (TForm)" décrit l'élément essentiel de l'EDI LAZARUS. Il s'agit de la fenêtre affichée contenant le bouton. On ne voit pas dans cette déclaration l'endroit où on a placé le bouton. L'endroit où on a placé le bouton se modifie dans l'"Inspecteur d'Objets".

L'"Inspecteur d'Objets" sauve les propriétés dans le fichier portant l'extension ".lfm". Vous pouvez voir ce fichier en affichant sur le formulaire modifiable un menu, grâce au bouton droit de la souris. Dans ce menu cliquez sur "Afficher le Source".

En scrutant le fichier ".lfm" vous voyez le bouton et ses coordonnées affichées.

Le type égal à "class (TForm)" permet d'afficher la fenêtre et son bouton en lisant ce fichier ".lfm".



"Afficher le Source" dans la conception du formulaire

Les modifications apportées au fichier permettent entre autre d'intervertir des Composants. Pour faire cela vous avez relié le paquet du Composant au projet grâce au "Gestionnaire de projet".

B) PAQUET POUR DÉBUTANTS

Dans la version 0.9.30 de LAZARS a été ajouté un paquet, une bibliothèque donc, permettant de faciliter le développement pour les débutants.

Ce paquet installe des fonctionnalités LAZARUS permettant de s'habituer au nommage, tout en facilitant la création de Code au sein de LAZARUS.

Allez dans "Paquets", puis "Configurer les paquets". Une fenêtre s'ouvre.
Sur la liste de droite, sélectionnez le paquet "educationlaz", pour l'"Installer". "Reconstruisez LAZARUS".

Après que LAZARUS ait redémarré, dans "Configuration", puis "Options", vous pouvez changer les paramètres pour débutants dans "Éducation".

Des les options d'"Education", dans "Général", activez le mode "Débutant".

Dans ces options toujours et dans la "Palette de Composants" "Montrez tout". Il est en effet possible d'y cacher les composants que l'on ne souhaite pas utiliser. Mais cette option, activée pour les professeurs, est surtout réservée aux démonstrations.

Vous pouvez afficher ou cacher d'autres outils, comme les boutons de référencement d'unités, de classes.

c)Indentation PASCAL

Vous voyez dans l'exemple que le Code Source est joliment présenté. Cela n'est pas une futilité puisque vous avez eu envie de le regarder. Vous avez peut-être eu envie de prendre exemple sur ce qui avait été fait. Voici la recette à suivre pour faire de même et mieux encore.

Un Code Source bien présenté ce sont des développeurs avec un comportement clair et précis. L'indentation c'est la présentation du Code Source pour qu'il devienne lisible. En effet vous créez du Code machine, alors que ce sont des hommes et femmes qui seuls peuvent le scruter, et le modifier correctement. Les erreurs sont toujours humaines.

L'indentation c'est donc du Code Source lisible. L'indentation permet de présenter le Code Source afin de le comprendre comme clair, sans avoir à fouiller pour trouver ce que l'on cherche.

L'indentation des éditeurs PASCAL est simple. En voici un résumé :

- Chaque instruction ou nœud (var, type, class, if, while, case, etc.) est à la même position que la ligne précédente
- Chaque imbrication d'un nœud doit être décalée de deux espaces

vers la droite
- Les instructions inclues entre les "Begin" et "End ;" sont décalées de deux espaces vers la droite

Cette présentation permet de revoir du Code Source plus facilement. Les développeurs qui regardent du Code Source avec cette présentation sont plus précis et plus sûrs d'eux.

Pour aider à indenter deux combinaisons de touches sont très importantes dans les éditeurs PASCAL :
- Ctrl + K + U permet de décaler votre sélection de lignes de deux espaces vers la gauche.
- Ctrl + K + I permet de décaler votre sélection de lignes de deux espaces vers la droite.

Vous pouvez aussi ajouter des règles de présentation du Code qui facilitent la maintenance. Nous vous disons pourquoi appliquer ces règles de présentation :
- Des commentaires sont obligatoirement avant chaque Source de fonctions ou procédures. Ainsi les commentaires avant chaque fonction ou procédure sont vus dans l'éditeur FREE PASCAL lorsque vous mettez la souris sur la fonction ou procédure.
- Des commentaires sont dans le Source si on y trouve un Bogue ou erreur. Si on corrige un Bogue dans le Source, il est possible que l'erreur revienne s'il n'y a pas de commentaires.
- Des commentaires sont dans le Source dès que l'on réfléchit pour écrire une Source. Si le Code demande réflexion pour être fait, c'est que vous y réfléchirez à nouveau si vous n'y mettez pas de commentaires. Notre mémoire oublie ce que l'on a écrit.
- Chaque fonction doit tenir sur la hauteur d'un écran. Une fonction qui ne tient pas sur la hauteur d'un écran est non structurée, illisible, difficile à maintenir, difficilement compilable si retravaillée.

Il existe d'autres méthodes de structuration, comme par exemple mettre des séries d'instructions alignées mot à mot. Cela permet de retrouver facilement les différences si le Code Source se répète.

Comme l'éditeur FREE PASCAL peut terminer les mots, il est possible d'affecter des longs noms de variables et procédures.

Toute nouveauté de l'éditeur est à saisir. Cela facilite le travail. Vous pouvez définir votre présentation et la tester.

Les exemples que nous vous montrons après sont indentés.

1)Paramétrer l'indentation

Vous pouvez modifier les "Options" d'"Indentation" dans "Configuration", puis "Options", puis "Outils de Code", puis "Général", puis "Indentation".

D)Structure du Code Source

Nous allons décrire le Code Source précédemment montré.

En général une unité faite en PASCAL se structure avec d'abord un entête, puis des déclarations publiques, puis le Code Source exécutable. Toute unité PASCAL se termine par "end.".

1)La déclaration Unit

La déclaration "Unit" est la première déclaration d'un fichier PASCAL. Elle est obligatoire et obligatoirement suivi du nom de fichier PASCAL, sans l'extension. On change cette déclaration en sauvegardant l'unité.

2)Les déclarations publiques

Les déclarations publiques comprennent obligatoirement le mot clé "interface" indiquant le début des déclarations publiques.

Vous pouvez ensuite y placer l'instruction "uses" permettant d'utiliser dans les déclarations publiques les unités déjà existantes.

Vous pouvez ensuite placer toute déclaration constante, type, variable, fonction ou procédure. Toute déclaration incluse dans la compilation ajoute du Code à l'exécutable.

Les variables allouent soit un pointeur soit un type simple.
Un pointeur est une adresse pointant vers un endroit de la mémoire. Il permet de définir et de stocker des types complexes.
Les constantes publiques peuvent être centralisées dans une ou plusieurs unités.

3)Le Code Source exécuté

Le Code Source commence par le mot clé "implementation".
En plus des déclarations déjà définies, on place dans cette partie le Code Source exécuté grâce aux fonctions et procédures.

Dans le Code Source exécuté on peut placer les mêmes déclarations que celles données précédemment. Seulement elles ne sont plus publiques donc pas utilisables ailleurs.

La procédure "ShowMessage" vous a permis d'afficher une boîte avec un message et un bouton. Si vous regardez le Code Source de "ShowMessage" grâce à "Ctrl" vous voyez qu'elle est déclarée avec le mot clé "procedure".

Une fonction s'appelle de la même façon mais renvoie une variable retour. En effet le rôle des fonctions et procédures est de modifier une partie de votre Logiciel. Elles transforment une entrée en sortie comme le ferait une équation mathématique.

L'événement que vous avez créé est une procédure particulière qui a permis d'afficher votre message à l'écran. La sortie ici c'est l'affichage de "Hello World" dans une fenêtre de dialogue.

Toute fonction ou procédure déclarée publiquement doit avoir une correspondance dans le Code Source exécuté. Le Compilateur vous le rappelle si vous oubliez. Le Code Source que vous avez ajouté était dans la partie "implementation".

4)Le Code Source exécuté au chargement

Il est possible d'exécuter du Code Source dès le chargement de l'unité afin d'automatiser la gestion de l'unité. Ces instructions sont à utiliser en dernier recours.

Le Code Source placé après le mot clé "initialization" permet de créer des Objets toujours en mémoire dès que l'on utilise l'unité.

Le Code Source placé après le mot clé "finalization" permet de détruire les Objets en mémoire à la fermeture du programme.

5)Fin de l'unité

La fin de l'unité est définie par "end."

E)Les fichiers ressources

A la fin d'un fichier PASCAL contenant votre formulaire, voici ce que l'on peut mettre :

```
{$IFDEF FPC}
  {$i Unite.lrs}
{$ENDIF}
```

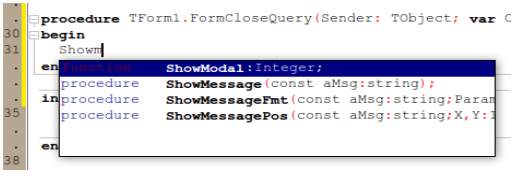
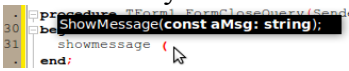
Ces trois lignes signifient : Si la directive FPC ou Compilateur FREE PASCAL est activée, alors inclure le fichier "Unite.lrs".

LAZARUS utilise d'autres fichiers ressources que Turbo PASCAL, ou DELPHI. On appelle les fichiers ressources LAZARUS les fichiers ".lrs" qui nécessitent idéalement d'avoir des fichiers images en format image compressée XPM. Les fichiers ".lrs" peuvent contenir aussi le fichier ".lfm" dans les unités de formulaires, pour les versions anciennes de LAZARUS.

Nous vous expliquons plus loin la création de votre premier Composant. Une partie de ce chapitre décrit la création et l'utilisation du fichier ".lrs".

F)Touches de raccourcis de complétion

La complétion permet d'écrire à votre place. C'est indispensable pour tout programmeur cherchant à gagner du temps.

Ctrl+Espace	Rechercher dans le Code lié : 
Ctrl+Maj +Espace	Afficher la syntaxe des méthodes : 
Ctrl+Maj+ flèche haut ou bas	Dans l'éditeur : Aller directement de la déclaration d'une méthode à son implémentation.

Ctrl+Maj+C	Complément de Code__: vous entrez **procedure** toto(s:**string**) dans :
	type
	TForm1 = class(TForm)
	private
	procedure toto(s:**string**);
	public
	Il vous écrit, dans la partie implémentation :
	procedure TForm1.toto(s:**string**);
	begin
	
	end;

Ctrl+Maj+i ou Ctrl+Maj+u	Indenter plusieurs lignes à la fois
	Passer de
	begin
	if X >0 **then** x=0;
	if X<0 **then** x=-1;A
	A:=truc+machintruc;
	end;
	begin
	if X >0 **then** x=0;
	if X<0 **then** x=-1;
	A:=truc+machintruc;
	end;
	Sans le faire ligne par ligne ?
	Sélectionner les lignes puis faire Ctrl+Maj+i pour déplacer les lignes vers la droite, ou Ctrl+Maj+u pour

	les déplacer vers la gauche.
Maj+touche flèchée et Ctrl+touche flèchée	Positionner ou dimensionner au pixel près un Composant

Ctrl+j	Faire appel aux modèles de Code.
	Nota : les modèles de Code permettent d'écrire du Code tout fait par exemple: en choisissant le modèle de Code if then else après avoir fait Crtl + j, on obtient
	
	if then
	begin
	
	end
	else
	begin
	
	end;
	
	Autre astuce : chaque modèle de Code possède une abréviation. Tapez cette abréviation puis Crtl + j. Tapez par exemple "if" puis Crtl + j. Les lignes correspondant à "if then else" s'écrivent à votre place. Les abréviations disponibles sont visibles en faisant Crtl + j.

Ctrl+Maj + 1 à 9 et Ctrl+ 1à9	Placer des marques (des signets) dans un Source pour pouvoir y revenir ultérieurement Vous êtes sur un bout de Source et vous vous aller voir ailleurs dans l'unité puis revenir rapidement : • Tapez :CTRL SHIFT 1 (ou un chiffre de 1 a 9 au dessus des lettres et non dans le pavé numérique) l'éditeur met un "1" dans la marge. • Pour revenir vous faites CTRL avec 1 • Pour annuler la marque, soit vous vous placez sur la ligne et vous refaites CTRL SHIFT 1, soit vous vous placez ailleurs et vous refaites CTRL SHIFT 1, pour déplacer la marque.
Ctrl+ flèche droite ou gauche	Se déplacer au mot suivant ou précédent.
Ctrl+Maj + droit ou gauche	Se déplacer au mot suivant ou précédent tout en sélectionnant.
Ctrl +haut ou bas	Revient au même que d'utiliser l'ascenseur

1)Touches de raccourci projet

Touche	Action du menu
Ctrl+F11	Fichier | Ouvrir un projet
Maj+F11	Projet | Ajouter au projet
F9	Exécuter | Exécuter
Ctrl + F9	Compiler le projet
Ctrl+S	Fichier | Enregistrer

Ctrl+F2	Réinitialiser le programme = arrête votre programme et revient en mode édition
Ctrl+F4	Ferme le fichier en cours

G) TOUCHES DE RACCOURCIS DE VISIBILITÉ

Touche	Action du menu
Ctrl+F3	Voir | Fenêtres de débogage | Pile d'appels
F1	Affiche l'aide contextuelle (si si je vous assure)
F11	Voir | Inspecteur d'Objets
F12	Voir | Basculer sur Formulaire/Unité
Ctrl+F12	Voir | Unités
Maj+F12	Voir | formulaires
Ctrl+Maj+E	Voir l'explorateur de Code
Ctrl+Maj+B	Voir l'explorateur de classe
Ctrl+ Maj+T	Ajouter un élément à faire
Alt+F10	Affiche un menu contextuel
Alt+F11	Fichier | Utiliser l'unité
Alt+F12	Bascule Form visuel / Source de la forme

H) TOUCHES DE RACCOURCIS DE DÉBOGAGE

Les raccourcis de débogage sont généralement utilisés dans l'éditeur de Source. Ils permettent de scruter le Code Source.

Il est possible de placer un point rouge sur une ligne de votre Source, dans la marge grisée à gauche de votre Source. C'est un point d'arrêt. Il va être appelé dès que l'on exécutera la ligne avec ce point d'arrêt, si l'exécutable le permet.

1)Exercice

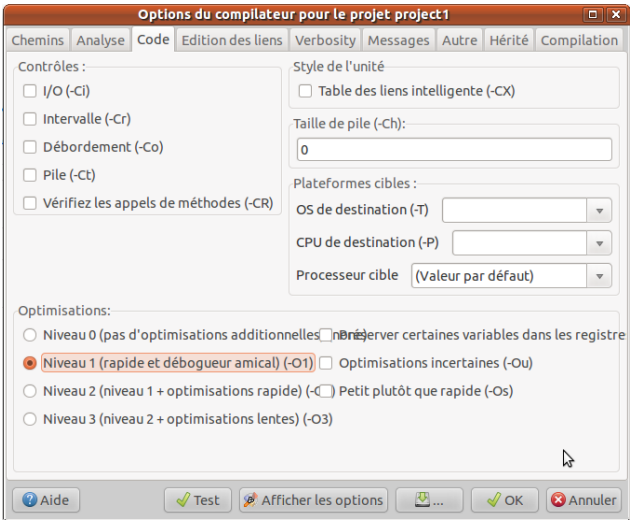
Placez un point d'arrêt à "ShowMessage" dans votre exemple.
Exécutez et cliquez sur le bouton. En général le programme se fige, puis LAZARUS se place sur le "Point d'arrêt" que vous avez mis.

2)Mes points d'arrêt sont inactifs

Pour que ces points d'arrêt fonctionnent, votre projet doit se compiler d'une certaine manière. Votre exécutable devient ainsi dix fois plus important qu'habituellement.
Si les points d'arrêt fonctionnent, votre exécutable est très lourd.

Allez dans le menu "Projet", puis "Options du Projet", puis "Options de compilation".
Allez dans l'onglet "Édition des liens".
Cochez "Afficher les numéros de lignes".
Ces numéros servent à ce que le débogueur se repère. Ces numéros de ligne alourdissent votre exécutable.

Allez dans l'onglet Code. En bas dans "Optimisations" choisissez l'option "Niveau 0" ou "Niveau 1».
Le niveau 0 crée un très gros exécutable mais scrute tous les points d'arrêt.



Le niveau 1 par défaut crée un gros exécutable

Touche	Action du menu
F4	Exécuter | Jusqu'au curseur
F5	Exécuter | Ajouter un point d'arrêt
F7	Exécuter | Pas à pas approfondi
Maj+F7	Exécuter | Jusqu'à la prochaine ligne
F8	Exécuter | Pas à pas
Ctrl+F5	Ajouter un point de suivi sous le curseur
Ctrl+F7	Evaluer/Modifier

I) TOUCHES DE RACCOURCIS DE L'ÉDITEUR

Touche	Action
Ctrl+K+B	Marque le début d'un bloc

Ctrl+K+C	Copie le bloc sélectionné
Ctrl+K+H	Affiche/cache le bloc sélectionné
Ctrl+K+K	Marque la fin d'un bloc
Ctrl+K+L	Marque la ligne en cours comme bloc
Ctrl+K+N	Fait passer un bloc en majuscules
Ctrl+K+O	Fait passer un bloc en minuscules
Ctrl+K+P	Imprime le bloc sélectionné
Ctrl+K+R	Lit un bloc de **procedure** puis un fichier
Ctrl+K+T	Marque un mot comme bloc
Ctrl+K+V	Déplace le bloc sélectionné
Ctrl+K+W	Écrit le bloc sélectionné dans un fichier
Ctrl+K+Y	Supprime le bloc sélectionné

J) TOUCHES DE RACCOURCIS DE L'ENVIRONNEMENT

Touche	Action du menu
Ctrl+C	Édition | Copier
Ctrl+V	Édition | Coller
Ctrl+X	Édition | Couper

E)MA PREMIÈRE APPLICATION

A)A FAIRE AVANT

Avant de créer sa première Application, sachez avant si c'est utile de créer le Logiciel voulu. Vérifiez que ce que vous voulez ne se fait pas déjà. A l'heure actuelle, seuls les Logiciels avec une personnalisation accrue sont à créer.

B)L'EXEMPLE

Dans notre premier exemple, nous allons tester la création de notre interface homme machine. On crée un utilitaire de recherche ou de remplacement, à personnaliser. Nous créons aussi notre propre savoir-faire centralisé.

C)CRÉATION DE L'INTERFACE

Un Logiciel doit être créé rapidement. Nous allons rapidement créer une interface Homme-Machine. LAZARUS est un outil permettant de créer rapidement un Logiciel, grâce à son interface graphique et du Code uniquement dédié au développement.

D)TESTER SES COMPOSANTS

Il est possible de créer rapidement son Logiciel.

Pour créer rapidement son Logiciel, il suffit de placer correctement les Composants, après avoir cliqué sur un Composant de la palette de Composants. La palette de Composants comporte un ensemble d'onglets identiques à des onglets de classeur. Lorsque vous cliquez sur un onglet portant un certain nom, un ensemble de Composants classés sous ce nom s'affichent.

La palette de Composants

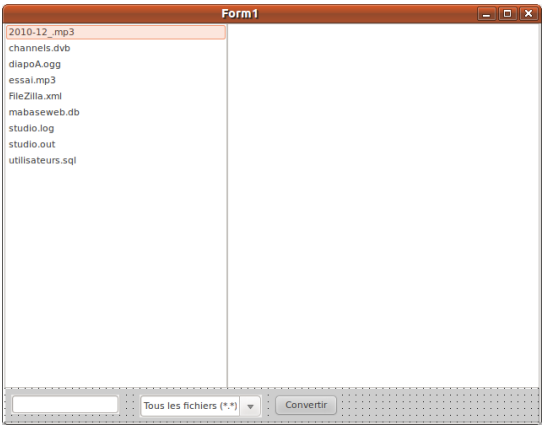
Vous voyez qu'il est possible de placer un certain nombres de Composants, sans limite de présentation.

Certains Composants ne sont représentés que par le même graphisme, de même taille que son icône situé dans la palette. Ces Composants n'ont en général aucune propriété graphique. Ce sont des Composants invisibles capables de réaliser certaines opérations visuelles ou pas.

D'autres Composants ajoutent un visuel à votre formulaire. Ce sont des Composants visuels. Avant de placer vos Composants visuels, n'oubliez pas d'utiliser un Composant "TPanel" pour une présentation soignée. Les Composants "TPanel" permettent à vos Composants visuels de se disposer sur l'ensemble de l'espace visuel disponible.

E) L'EXEMPLE

Nous allons créer cette interface ci-après. Elle s'adapte automatiquement à sa fenêtre :



Ma première Application

1)Une présentation soignée

Pour commencer notre Interface Homme-Machine, disposez un Composant "TPanel".
Les "TPanel" permettent à l'interface d'utiliser le maximum d'espace de travail.

Notre premier "TPanel" sert à disposer les boutons. En cliquant sur l'onglet "Standard" vous pouvez ajouter un panneau, en anglais un "panel". Cliquez dessus, puis cliquez sur le formulaire.

Cliquez sur F11 pour afficher l'inspecteur d'Objet. Votre "TPanel" y est sélectionné. Cliquez sur la boîte à option de la propriété "Align", et affectez "alBottom" à "Align".

Le deuxième "TPanel" sert à disposer le Composant de sélection du répertoire. Ajoutez le "TPanel" de la même manière mais ajoutez-le dans la zone vide.

Cliquez sur F11 pour afficher l'"Inspecteur d'Objet". Votre "TPanel" y est sélectionné. Cliquez sur la boîte à option de la propriété "Align", puis affectez "alTop" à "Align".

Le troisième "TPanel" affiche les fichiers à convertir.
Disposez un deuxième "TPanel" dans la zone où il n'y a pas de "TPanel".

Votre "TPanel" doit donc s'aplatir sur tout le formulaire. Affectez "alClient" à la propiété "Align" de ce "TPanel" afin de disposer d'un maximum d'espace. Vous pouvez enlever les bordures de vos "TPanel"...

2)L'Interface Homme Machine

Ensuite nous allons disposer les Composants de sélection de répertoire et de fichiers, dans les "TPanel". Nous allons aussi disposer les boutons.

Disposez un Composant "TDirectoryEdit" sur le panneau "TPanel" du haut. Le Composant "TDirectoryEdit" peut sembler insatisfaisant graphiquement. En effet, il n'affiche pas l'arborescence détaillée des répertoires. Vous pouvez consulter le chapitre suivant pour rechercher un meilleur savoir-faire.

Il est possible d'ancrer le Composant "TDirectoryEdit" sur la longueur du "TPanel". Ou alors choisissez un Composant de sélection de répertoire plus abouti. Les Composants fournis avec LAZARUS sont testés sur l'ensemble des plates-formes disponibles, en fonction de la "RoadMap" LAZARUS. Consultez-la sur www.lazarus.freepascal.org. Par exemple la "Roadmap" LAZARUS indique en 2011 que des

Composants ne marchent pas sur MAC OS.

Disposez un Composant de sélection de fichiers "TFileListBox" dans le panneau du milieu. Vous pouvez lui affecter la valeur "alClient" à sa propriété "Align". Vos fichiers sont disposés sur l'ensemble de l'interface. Ils sont donc visibles pour l'utilisateur.

Disposez les boutons "TButton" sur le panneau du bas. Les boutons n'ont pas besoin d'être alignés, car le texte qui leur est affecté a une longueur connue. Ils sont alignés à partir de la gauche.

Vous pouvez par exemple créer un bouton qui affiche la date du fichier, son chemin, etc.

Nous allons créer un Logiciel de conversion de fichiers LAZARUS.

Créer un bouton avec le nom "Convertir". Vous voyez que son libellé visuel change. Vous avez affecté aussi la propriété "Caption" du Composant. Changez le nom et son libellé visuel séparément.

Il est nécessaire de relier l'éditeur de répertoire avec la liste de fichier. Double-cliquez sur l'événement "OnChange" du "TDirectoryEdit". Placez-y cette Source :

```
procedure TForm1.DirectoryEditChange(Sender: TObject);
begin
  FileListBox.Directory := DirectoryEdit.Directory ;
end;
```

Double-cliquez sur l'événement "Onclick" du bouton.
Placez-y cette Source :

```
procedure TForm1.ConvertirClick(Sender: TObject);
var li_i : Integer ;
begin
  For Li_i := 0 To FileListBox.Items.Count - 1 do
```

```
  Begin
    ConvertitFichier ( DirectoryEdit.Directory + DirectorySeparator +
FileListBox.Items.Strings [ li_i ], FileListBox.Items.Strings [ li_i ] );
  End ;
end;
```

Ne compilez pas. Il manque la procédure "ConvertitFichier" ici :

```
procedure TForm1.ConvertitFichier(const FilePath, FileName :
String);
var li_i : Integer ;
   ls_Lignes : WideString ;
   lb_DFM ,
   lb_LFM : Boolean ;
   lst_Lignes : TStringList ;
begin
    lb_DFM := PosEx ( '.dfm', LowerCase ( FilePath ), length
( FilePath ) - 5 ) >= length ( FilePath ) - 5 ;
  lb_LFM := PosEx ( '.lfm', LowerCase ( FilePath ), length ( FilePath )
- 5 ) >= length ( FilePath ) - 5 ;
  lst_Lignes := TStringList.Create;
  try
    lst_Lignes.LoadFromFile ( FilePath );
    ls_Lignes := lst_Lignes.Text;

    Application.ProcessMessages;

    // Conversion d'une chaine du fichier
     ls_Lignes := StringReplace ( ls_Lignes, 'TADOQuery', 'TZQuery',
[rfReplaceAll,rfIgnoreCase] );

     if lb_DFM
     or lb_LFM Then
      Begin
```

```
      End;
     else
      Begin
      End;

    Application.ProcessMessages ;
    lst_Lignes.Text :=  ls_Lignes ;
    lst_Lignes.SaveToFile   ( FilePath );
  finally
    lst_Lignes.Free;
  end;
end;
```

Vous voyez que cette méthode prend beaucoup de place. Vous pouvez la couper, afin de mieux programmer.

Une fois que vous avez créé la procédure "ConvertitFichier", appuyez simultanément sur "Ctrl", "Shift", avec "C". Ainsi, si la source compile, la procédure "ConvertitFichier" est renseignée dans votre fiche en fonction de la déclaration.

Au début on teste si l'extension de fichier est "lfm", "dfm", ou "pas".

La fonction "ProcessMessages" de TApplication sert pendant les boucles. Elle permet à l'environnement de travailler, afin d'afficher la fenêtre.

La fonction "StringReplace" retourne une chaine, dont une sous-chaine en paramètre a éventuellement été remplacée. Elle possède des options, comme le remplacement sur toute la chaine, la distinction ou pas des majuscules et minuscules.

A la fin on affecte le "TStringlist" par sa propriété "Text", puis on remplace le fichier en utilisant la procédure "SaveToFile", avec le nom

de fichier passé en paramètre de la procédure "ConvertitFichier". Faites une copie du projet, avant de le convertir par cette moulinette.

Il n'y a plus qu'à filtrer les fichiers de la liste de fichiers en fonction du résultat demandé par l'utilisateur : Convertir tout ou convertir des fichiers "lfm", des fichiers "dfm" ou des fichiers "pas".

Placez une TFilterComboBox. Renseignez les types de fichiers voulus en renseignant "Filter". S'il n'y a pas d'éditeur de propriété avec le lien vers l'éditeur, placez ce genre de filtre :

```
Tous les fichiers (*.*)|*.*|Fichiers DELPHI et LAZARUS|*.dpr;*.pas;*.dfm;*.lfm|Fichiers projet|*.dpr|Unités DELPHI|*.pas|Propriétés DELPHI|*.dfm|Propriétés LAZARUS|*.lfm
```

On voit qu'avec la version 0.9.30 et sans éditeur de propriété il y a du temps perdu.

Puis renseignez l'événement "OnChange" avec cette Source :

```
procedure TForm1.FilterComboBoxChange(Sender: TObject);
begin
  FileListBox.Mask := FilterComboBox.Mask;
end;
```

Vous pouvez maintenant vous amuser à changer les Composants de projets LAZARUS, en série.

Notre interface est prête à être renseignée. Ce qui est visible par l'utilisateur a été créé rapidement. Nous n'avons pas eu besoin de tester l'interface. Pour voir le comportement de cet exemple, exécutez en appuyant sur "F9" ou en cliquant sur le bouton ▶.
Vous pouvez agrandir votre formulaire ou le diminuer pour tester votre interface.

F) CHERCHER DES PROJETS

Ce chapitre est une aide servant à améliorer vos projets grâce à INTERNET.

LAZARUS possède à son installation un nombre limité de Composants. Il est possible de trouver des savoir-faire sur INTERNET. Pour trouver un Composant, définissez ce que vous souhaitez, ainsi que les alternatives possibles. Ce n'est qu'avec l'expérience que l'on trouve des Composants adéquates.

Sur votre moteur de recherche tapez plutôt des mots anglais comme "component", "project" ou "visual" avec "LAZARUS" voire "DELPHI". Puis, avec ces mots, tapez votre spécification en anglais. Changez de mots ou de domaines, si vous ne trouvez pas.

G) INSTALLER DES COMPOSANTS

Avant de compiler LAZARUS, Pensez à dupliquer l'exécutable LAZARUS.

Pour installer des Composants décompressez votre archive, puis placez-la dans un dossier qui ne bougera pas.
Avec LAZARUS ouvrez le ou les paquets portant l'extension ".lpk".
Compilez et installez. Ne recompilez LAZARUS que si vous n'avez plus de Composants à ajouter.
Si le paquet ne compile pas cherchez dans le dossier ou sur Internet les paquets manquants et installez les.

H) LAZARUS NE DÉMARRE PLUS

1)Sur LINUX

Si LAZARUS ne redémarre pas alors allez dans votre "dossier personnel". Dans "Affichage" "Montrez les fichiers cachés". Allez dans le dossier ".LAZARUS" puis dans "bin". Effacez "LAZARUS".
Si vous ne trouvez pas l'exécutable LAZARUS, réinstallez le paquet "LAZARUS-ide".

2)Sur WINDOWS

Effacez le fichier "LAZARUS.exe". Il est soit dans "Documents and Settings" ou "Users", puis dans votre compte utilisateur, puis dans "Local Settings", "Application Data" et "LAZARUS".

Ou bien il est dans "C:\LAZARUS". Dans ce cas recopiez l'exécutable LAZARUS ou réinstallez LAZARUS.

I)VÉRIFIER LES LICENCES

Après avoir téléchargé vérifiez l'utilisation des licences. Chaque Composant possède des particularités de licence avec le mot "Modified" ou un autre mot générique.
Si vous avez téléchargé des Composants gratuits, voici les différents types de licences que vous pouvez trouver :

- Pas de Sources. Vos composants s'installent sur certains LAZARUS. Vous n'avez pas la possibilité de reprendre le projet s'il est abandonné.
- Aucun fichier de licence : En général vous faites ce que vous voulez. Vous devez contacter l'auteur pour définir une licence.

- GPL : Cette licence vous oblige à diffuser les Sources de votre Logiciel ou descendant, si vous diffusez le Composant, ceci en annonçant l'auteur.
- Creative Common by : Cette licence vous oblige à annoncer l'auteur du Composant.
- BSD : Cette licence entièrement Libre vous autorise à revendre le Composant, comme vous le voulez.
- Etc.

Une licence commerciale, pour laquelle vous avez acheté des Composants, peut disposer d'un ensemble de restrictions à lire.

J) COMPILATEUR FREE PASCAL

Le Compilateur qui a permis de créer LAZARUS est aussi celui utilisé pour créer les programmes. La seule différence est qu'il est semi-automatisé. Aussi il utilise les librairies LAZARUS. Pour compiler sur LAZARUS et vérifier son Code créé, appuyez sur "Ctrl" et "F9" en même temps. Pour exécuter sur LAZARUS cliquez sur le bouton Play ( ) ou sur "F9".

Vous exécutez ligne après ligne votre programme et appuyant sur "F7" ou "F8". Ces touches permettent de voir le Code s'exécuter, en regardant vos Sources. La touche "F8" passe certaines Sources.

L'utilisation des touches "F7" et "F8" est fastidieuse. Vous pouvez ajouter un voire plusieurs points d'arrêts, en cliquant dans votre "Éditeur de Source", sur la marge grisée d'une ou de plusieurs lignes à vérifier.

Le Compilateur FREE PASCAL crée des fichiers exécutables volumineux. Pourquoi ?

Parce que LAZARUS ne déporte pas l'utilisation des librairies de chaque environnement. Autrement dit votre exécutable peut ne pas utiliser les librairies que vous avez utilisées pour créer l'exécutable. Un exécutable LAZARUS a de très grandes chances de fonctionner seul.
Aussi LAZARUS, en version 0.9.30, peut ajouter des unités, liées par les paquets, que vous n'utilisez pas dans votre exécutable.

K) GESTION DES ERREURS

Dans votre Logiciel, le développeur et l'utilisateur doivent être guidés. Des erreurs peuvent se produire dans le Code servant au développement, ou dans celui servant à l'utilisateur. En général les erreurs de développement doivent être en anglais, tandis que les erreurs de l'utilisateur sont multilingues.

Anticipez ce qui va se passer dans votre programme. Imaginez un utilisateur tombant sur une erreur en anglais, alors qu'il ne connaît pas cette langue. En plus, est nécessaire un manuel de l'utilisateur, permettant de trouver les informations nécessaires à l'utilisation du Logiciel.

L) LES EXCEPTIONS

Les exceptions permettent de gérer les erreurs, ou bien de les rediriger correctement vers l'utilisateur. Méfiez-vous des Logiciels qui rapportent peu d'erreurs. En général on ne sait pas comment les utiliser.

```
try
  for i:=0 to 20 do
    chaine := chaine + chaine;
except
    Showmessage ( 'La chaine de mon formulaire est trop longue. Il n'y
```

```
a plus assez de mémoire.' );
end;
```

Cette gestion d'exception commence à "try" et finit à "end". La partie entre "try" et "except" rapporte vers la partie entre "except" et "end" l'erreur éventuelle.

Dans cette gestion d'exception on induit qu'une erreur produite est due au manque de mémoire.

Voici la même gestion d'exception mieux construite :

```
try
  for i:=0 to 20 do
    chaine := chaine + chaine;
except
  On E:EOutOfMemory do
       Showmessage ( 'La chaine de mon formulaire est trop longue. Il
n'y a plus assez de mémoire.' );
  On E:Exception do
       Showmessage ('Erreur avec la chaine. Le résultat peut être
tronqué.');
end;
```

L'instruction de l'exception " E:EOutOfMemory" est appelée quand il n'y a plus assez de mémoire. Cette exception se produit en général avec les manipulations de tableaux à longueur variable.

L'instruction de l'exception "On E:Exception do", suivie de sa gestion, récupère toutes les erreurs d'exception produites. Si elle est seule, elle peut être éludée. C'est pourquoi on place une exception, héritée au dessus de l'exception ancêtre, lorsqu'on utilise ce genre de gestion d'exception.

1)Renvoyer les exceptions

Au départ les erreurs peuvent s'adresser à soi. Mais à force d'utiliser son Logiciel on redéfinit les exceptions pour l'utilisateur.

Dans un Code réutilisé il peut être intéressant de renvoyer l'exception vers le programme, afin que lui seul gère le message à donner à l'utilisateur.

Le mot clé "raise" permet de renvoyer une erreur vers l'utilisateur.

```
type EStringTooLarge : EOutOfMemory;
begin
  if length ( chaine ) > 1000 Then raise (  EStringTooLarge );
  for i:=0 to 20 do
    chaine := chaine + chaine;
end;
```

Voici la même gestion d'erreur que précédemment, mais dans un Code Source réutilisé. On anticipe toujours sur le développeur, en réutilisant au mieux l'Héritage des erreurs. Il est possible d'utiliser les exceptions de LAZARUS existantes pour ce que l'on souhaite rediriger. C'est d'ailleurs recommandé.

Pratiquer la veille technologique permet de voir si son erreur n'a pas été ajoutée, avec une nouvelle version de son kit de développement.

F)L'OBJET

A)INTRODUCTION

La programmation orientée Objets permet, dans LAZARUS, de réaliser vos propres Composants, votre savoir-faire. Avec l'Objet, elle permet aussi de réaliser des Logiciels complexes. Ce chapitre et les deux chapitres suivants sont complémentaires. Ils doivent être bien compris afin de surpasser les capacités du programmeur.

Dans ce chapitre nous allons définir ce qu'est l'Objet. Cela va permettre de mieux comprendre LAZARUS, donc de sublimer cet outil par l'automatisation RAD. LAZARUS est un outil de Développement Rapide d'Applications, aboutissement de l'utilisation de la programmation orientée Objets.

Si vous ne connaissez pas l'Objet vos Composants sont mal faits. Il existe des règles simples et des astuces permettant de réaliser vos Composants.

Nous allons dans ce chapitre vous parler simplement de l'Objet.
Étoffez vos connaissances avec des livres sur l'analyse Objet, analyse proche de l'humain. Elle nécessite du travail et de la technique. Nous parlons succinctement des relations en Objet dans le chapitre sur les **Logiciels centralisés**.

Chaque Compilateur possède des spécificités ou améliorations, pour faciliter le travail du développeur. Nous vous les expliquons.

B) UN OBJET

Un Objet est une entité interagissant avec son environnement. Par exemple votre "clavier" d'ordinateur est un Objet. Ce "clavier" possède des "touches" qui peuvent aussi être des Objets "touche".

Vous voyez qu'il est possible de définir un Objet "Touche" ou un Objet "Touches". Préférez l'unicité en mettant en tableau votre Objet "Touche" au sein de votre Objet "Clavier". Vous créez un attribut au sein de votre Objet "Touche".

C) UNE CLASSE

Avec la notion de programmation par Objets, une méthode de programmation proche de l'humain, il convient d'énoncer la notion de classe.

Lors du tout premier exemple on avait déclaré une classe dans le premier chapitre, en créant un formulaire hérité de la classe "TForm". Cette notion de classe est présente dans le FREE PASCAL. Elle a été ajoutée au PASCAL standard.

Ce que l'on a pu nommer jusqu'à présent Objet est, pour FREE PASCAL, une classe d'Objet. Il s'agit donc d'un type FREE PASCAL. L'Objet FREE PASCAL est une instance de classe, plus simplement un exemplaire d'une classe, son utilisation en mémoire avec la possibilité d'être dupliqué.

On peut remarquer que FREE PASCAL définit une classe comme un Objet (type "classe") ou comme un enregistrement (type "record"). L'enregistrement record ne permet que de définir des méthodes. La classe est l'Objet analytique pouvant agir autour de lui.

Une classe accepte l'instruction "with", Vous pouvez utiliser "with" en prenant garde, toutefois, aux variables ou méthodes identiques, dans des entités différentes.

D)Une Instance d'Objet

Une instance d'Objet en UML est un Objet en mémoire en FREE PASCAL.
L'Objet en UML est une classe d'Objet en FREE PASCAL.

Le PASCAL Objet permet de créer des Instances d'Objets. Une Instance d'Objet c'est un Objet mis dans la mémoire de l'ordinateur, à l'exécution. Une Instance d'Objet peut être dupliquée. Elle est aussi personnalisée grâce à ses variables.

Si votre Classe d'Objet et ses Classes parentes ne contiennent aucune variable, il est nécessaire de se poser la question de leur utilité. Puis-je gagner du temps en créant une unité de fonctions à la place ?

Vous créez votre Objet, qui est instancié une ou n fois en exécution. Par exemple les formulaires prennent beaucoup de place en mémoire donc on les instancie en général une fois.

Les variables simples de votre Objet sont dupliquées et initialisées à zéro, pour chaque Instance créée.

1)Les attributs et propriétés

Deux Objets "Clavier" peuvent être faits de la même manière, avec une couleur différente. Ce sont toujours deux Objets "Clavier" avec un

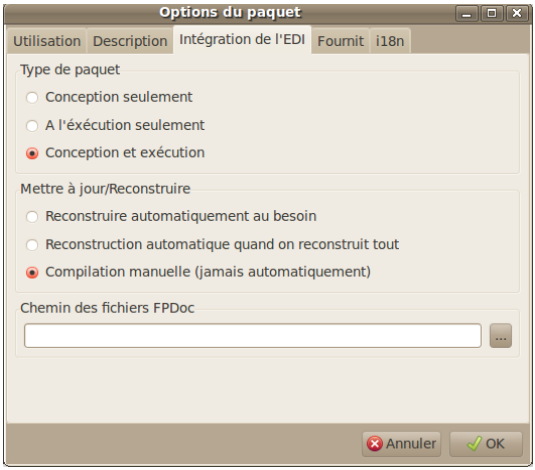
attribut différent. Un attribut peut être une couleur, un nombre, du texte, ou bien un autre Objet.

Seulement en FREE PASCAL les attributs, nommés variables, peuvent être enregistrés dans l'EDI comme des propriétés. Ces variables sont alors sauvées dans l'Objet "TForm", visible en exécution, ou le "TDataModule" invisible. Dès que vous modifiez un formulaire LAZARUS vous utilisez l'Objet. Seulement ces Objets n'ont pas besoin de l'EDI, pour charger leurs propriétés à l'exécution. Une partie du Code Source de développement est exécuté.

Au début du livre un formulaire possédait des propriétés permettant d'automatiser facilement le développement. Les propriétés manipulables dans l'"Inspecteur d'Objets" sont du Code Source servant uniquement au développement. Elles vous ont permis de gagner beaucoup de temps, dans la création de votre premier Logiciel LAZARUS.

Les développeurs LAZARUS ont soigneusement séparé les unités d'édition de propriétés, servant au développement du Code Source exécuté. Les Objets, éditant les propriétés dans l'"Inspecteur d'Objets", ne sont pas inclues dans votre exécutable. Lorsque vous concevrez votre premier Composant, il faudra aussi séparer le Code Source de développement du Code Source exécuté.

Vous ne devez jamais lier dans vos projets directement vers une unité d'enregistrement de Composants, puisqu'elle ne sert qu'à LAZARUS. C'est pour cette raison que des unités, spécifiques à l'enregistrement des Composants, sont à part des unités de Composants.



Le Code Source exécuté peut être séparé de la conception

2)Un attribut dans LAZARUS

Dans les classes LAZARUS, les attributs se déclarent avant les méthodes. Une méthode est une fonction ou procédure au sein d'une classe.

```
type
        TForm1 = class ( TForm )
          MonBouton : TButton ;
        public
         constructor Create ( AOwner : TComponent ) ;
        end;
```

L'attribut "MonBouton" crée un Objet bouton. L'attribut du type bouton sert à créer un bouton, dans le formulaire TForm1.

3)Une propriété dans LAZARUS

```
type
        TForm1 = class ( TForm )
          MonBouton : TButton ;
          procedure OnClickMonBouton ( Sender : TObject ) ;
        private
          FClickBouton : Boolean ;
        public
         constructor Create ( AOwner : TComponent ) ;
         property ClickBouton : Boolean read FClickBouton write
FClickBouton default false;
        end;
```

L'événement "OnClick" du bouton "MonBouton" fait appel à la méthode "OnClickMonBouton".

L'attribut privé "FClickBouton" ne peut être lu dans son descendant. Ainsi on accède à cet attribut par sa propriété publique. Ainsi on peut transformer la fiche en Composant plus facilement.

La propriété permet d'accéder aux attributs directement ou indirectement. Vous pourriez créer dans la propriété une fonction appelée getter, et une procédure appelée setter, permettant de préparer l'attribut avant de l'utiliser.

La propriété "ClickBouton" permet ici de ne pas avoir à définir de getter et setter afin d'accéder au clique de bouton. Ainsi un getter ou setter peuvent être ajoutés ensuite à la propriété. Le Code Source n'est que très peu changé.

On vous montre les différentes options de déclaration de propriétés. FClickBouton doit être initialisé à "false", dans le Constructeur de la

fiche. En effet, la déclaration "property" ne permet pas d'initialiser de valeur par défaut. Une déclaration n'agit pas en général.

4)Les méthodes et événements

Le clavier agit. Il possède alors des méthodes influençant ses attributs ou d'autres Objets. On pourrait créer une méthode d'appui sur une touche quelconque. La méthode est une procédure ou une fonction centralisée dans un Objet. Une méthode modifie les attributs ou variables du programme.

Seulement en FREE PASCAL il y a plus adaptable que la méthode. L'événement permet de renseigner rapidement tout Objet. Ainsi on ne crée pas l'Objet "Clavier".

LAZARUS va facilement propager des événements dans le formulaire de votre projet, puis dans chaque Objet de formulaire. Chaque Objet possède alors une méthode captant le clavier. Vous pouvez renseigner les différents événements clavier dans l'"Inspecteur d'Objets". Ces événements sont accessibles aussi, au sein des différents Objets, grâce à des méthodes.

L'événement consiste à transformer une méthode en variable, pour pouvoir, comme les propriétés, transférer facilement un événement à un autre Objet.

E)LES COMPORTEMENTS DE L'OBJET

L'Objet est, contrairement aux langages procéduraux, plus proche de l'homme que de la machine. Il permet de réaliser des systèmes humains

plus facilement. L'Objet étant plus proche des représentations humaines il complexifie au minimum le travail de la machine en l'organisant en Objets. Un Objet possède trois propriétés fondamentales :

- L'Héritage
- L'Encapsulation
- Le Polymorphisme

L'Objet peut être représenté schématiquement grâce à une boîte à outils nommée UML. Il existe des Logiciels Libres permettant de créer des schémas Objets, comme STAR UML ou TOP CASED.

F) L'HÉRITAGE

Les formulaires LAZARUS utilisent l'Objet. Vous voyez une définition d'un type Objet, dans chacune des unités de votre formulaire. Vous avez donc utilisé de l'Objet sans le savoir.

Un type Objet se déclare de cette manière :

```
type
        TForm1 = class ( TForm )
        end;
```

On crée ici une nouvelle classe qui hérite de la classe "TForm". Vous pouvez appuyer sur "Ctrl" ,puis cliquer sur "TForm", pour aller sur la déclaration du type "TForm". Vous ne voyez alors que des propriétés, car "TForm" hérite lui aussi de "TCustomForm", etc.

Notre formulaire se crée donc grâce à l'Objet "TForm". Cet Objet possède la particularité de gérer un fichier "lfm", contenant les Objets visibles dans la conception du formulaire.

Votre formulaire est compris par LAZARUS par le procédé de l'Héritage. Il n'utilise que rarement toutes les possibilités du type

formulaire surchargé.

Vous pourriez donc utiliser un autre type descendant de "TForm" pour votre formulaire. Cependant tout Composant LAZARUS est hérité du Composant le plus proche de son résultat demandé. On peut donc hériter votre formulaire du Composant "TCustomForm" si certaines propriétés gênent.

Vous pouvez regarder que ces deux Objets ne diffèrent que par la visibilité de leurs propriétés, car "TForm" hérite de "TCustomForm".
Vous voyez des déclarations de variables dans l'Objet "TForm". Certaines variables, comme les Objets, sont mises en mémoire par le Constructeur du formulaire.

Vous ne voyez en général pas de Constructeur dans un formulaire descendant de "TForm". En effet LAZARUS met automatiquement en mémoire vos variables de Composants, déclarées grâce à chaque fichier ".lfm" créé. Ce fichier ".lfm", c'est votre formulaire visuellement modifiable, grâce à la souris et à l'"Inspecteur d'Objets".

G) LA SURCHARGE

La surcharge consiste dans l'Héritage à modifier le comportement d'une méthode surchargée. Par exemple, on peut hériter de l'Objet "TForm", afin de surcharger la méthode "DoClose". Cette méthode surchargée contient le comportement spécifique de toutes les fiches que l'on utilise. Ainsi, dans un Composant héritant de "TForm", on peut automatiser la fermeture de ses formulaires. Il y a moins de Code puisque la fermeture est dans un seul Objet.

1)Le mot clé "virtual"

Par défaut toute méthode déclarée ne peut être surchargée. Une méthode est donc par défaut statique car il est impossible de la surcharger. En effet une méthode statique ne peut qu'être éludée, pas surchargée.

Pour qu'une méthode puisse être modifiée par ses héritières, ajoutez le mot clé "virtual". Si vous scrutez la toute première déclaration de la méthode DoClose vous trouvez ainsi cette Source dans la première déclaration de DoClose :

```
procedure DoClose(var CloseAction: TCloseAction); virtual;
```

On peut aussi utiliser cette déclaration :

```
procedure DoClose(var CloseAction: TCloseAction); dynamic;
```

Il est préférable d'utiliser la déclaration "virtual". En effet cette déclaration favorise la vitesse, plus importante que la taille en mémoire avec "dynamic".

Il existe d'autres moyens pour gagner de l'espace. Nous vous les citons dans le chapitre sur les Composants.

2)Le mot clé "override"

Pour surcharger une méthode virtuelle, ajoutez cette Source dans une classe descendante. Vous pouvez par exemple ajouter ce Code Source dans tout formulaire :

```
procedure DoClose(var CloseAction: TCloseAction); override;
```

Cette déclaration dans le formulaire est identique à l'affectation de l'événement "OnClose" de votre formulaire, hérité de "TForm".

Seulement créer cette méthode permet de créer un Composant personnalisé, hérité de "TForm".

Il suffit d'appuyer en même temps sur "Ctrl", puis "Shift", puis "C" pour créer le Code Source à automatiser. cette Source est ainsi créé :

```
procedure TForm1.DoClose(var CloseAction: TCloseAction);
begin
  inherited DoClose(CloseAction);
end;
```

Ainsi, lorsque tout Objet Propriétaire de "Doclose" appelle cette méthode, on appelle d'abord la méthode "Doclose" au dessus de toutes, qui peut éventuellement appeler les méthodes ascendantes "DoClose", par le mot clé "inherited".

Choisissez toujours d'utiliser le mot clé "inherited", lorsque vous surchargez votre méthode. Sinon votre programme boucle à l'infini.

Vous pourriez créer un Composant hérité de "TForm", avec vos options de fermeture de formulaires. Ainsi du Code Source ne serait plus dupliqué.

H) L'ENCAPSULATION

Nous venons de voir, par la surcharge, le procédé de l'Encapsulation. L'Encapsulation permet de cacher des comportements d'un Objet, afin d'en créer un nouveau.

L'Encapsulation existe déjà dans les langages procéduraux. Vous pouvez déjà réutiliser une fonction ou procédure, en affectant le même

nom à la nouvelle fonction ou procédure, dans une nouvelle unité, qui réutilise la fonction ou procédure. Seulement vous utilisez le nom de l'unité pour distinguer les deux fonctions ou procédures.

On pourrait ainsi appeler la fonction ou procédure imbriquée comme l'original. On pourrait donc se tromper facilement de fonction ou procédure identique, en éludant une unité. L'Encapsulation en PASCAL non Objet est donc approximative. En effet on évite de créer les mêmes noms de fonctions ou procédures, car cela crée des conflits sans forcément que l'on s'en aperçoive.

Au travers des unités et autres librairies, l'Encapsulation est améliorée avec FREE PASCAL en Objet.

L'intérêt de l'Encapsulation est subtil et humain. Si vous créez un descendant d'un bouton en surchargeant la méthode "Click", afin de créer un événement centralisé, et que vous utilisez ce bouton dans votre formulaire, vous vous apercevez que vous ne pourrez jamais appeler la méthode "Click" de ses ancêtres.

Autrement dit l'Encapsulation permet de modifier légèrement le comportement d'un Objet afin d'en créer un nouveau, répondant mieux à ce que l'on cherche. On peut hériter du Composant au-dessus, si l'on veut utiliser la nouvelle gestion, éludant l'ancienne, par le nouveau Composant.

Autrement dit, on s'aperçoit ici que la programmation Objet demande plus de mémoire que la programmation procédurale. En effet il est possible d'éluder le Code Source de la méthode encapsulée.

Cependant l'Objet permet de gagner du temps dans le développement, grâce à l'automatisation des systèmes humains, puis informatiques. L'Objet permet aussi de maintenir plus facilement le Code Source créé. Avec LAZARUS, tout Objet Composant Open Source d'un formulaire

est consultable, par simple clic de souris, grâce à l'inspecteur d'Objet.

Une variable ne peut pas être être héritée, mais peut être de nouveau déclarée. Cependant l'ancienne variable prend alors la place réservée à son pointeur à l'exécution.
Il est même possible que vous ne puissiez empêcher le Constructeur de la variable de mettre en mémoire ses autres variables, s'il s'agit d'une classe.
Quand trop de variables fantômes sont présentes dans l'ancien Composant hérité, demandez-vous s'il est utile de créer un nouveau Composant, héritant d'un des Composants ascendants. En effet les variables fantômes sont en mémoire, en partie, dans chaque instance de classe.

1) La déclaration "private"

Une méthode "private" ne peut pas être surchargée. Une variable privée n'est accessible dans aucun descendant. On ne peut accéder à une méthode, ou une variable "private", que dans l'Objet la possédant.

Allez sur la déclaration de "TForm" :

```
  TCustomForm = class(TScrollingWinControl)
  private
    FActiveControl: TWinControl;
    procedure SetActiveControl(AWinControl: TWinControl);
  published
    property ActiveControl: TWinControl read FActiveControl write
SetActiveControl;
  end;
```

Vous voyez, dans les Sources de LAZARUS, que beaucoup de

variables sont déclarées en privées. Elles sont tout de même accessibles dans l' "Inspecteur d'Objets", sous un autre nom. Souvent, il suffit d'enlever la première lettre de la variable, pour la retrouver dans l'"Inspecteur d'Objets". Des déclarations privées sont utilisées grâce aux getters et setters, eux aussi privés, afin d'être visibles dans l'"Inspecteur d'Objets".

La déclaration "private" permet de regrouper les variables, avec les "getters" et les "setters" des propriétés créées. Un "setter" est une affectation d'une variable, contenant généralement son nom, initialisant une partie du Composant. Le "getter" fait de même, en lisant cette fois la variable privée, contenue dans son libellé.

Ainsi le programmeur ne se trompe pas lorsqu'il réutilise la variable. Il utilise la propriété déclarée par "property", définissant d'éventuels "getters" et "setters", permettant d'initialiser le Composant en affectant la variable. Vous voyez des propriétés dans les Composants LAZARUS.

2)La déclaration "protected"

Une méthode "protected" ne peut être surchargée que dans l'unité de l'Objet, vers les Objets hérités. Surcharger un Composant permet donc d'accéder aux méthodes ou variables protégées, afin de modifier le comportement d'un Composant.

Le type "TForm1" peut accéder aux méthodes et variables "protected" de "TForm" et de ses ascendants. Par contre ce type ne peut pas accéder aux méthodes et variables protégées des Composants non hérités, en dehors de son unité.

On met dans la zone "protected" les méthodes de fonctionnement du

Composant, pouvant être surchargées, afin d'être modifiées dans un Héritage. Si on ne sait pas comment mettre une méthode dans la zone "private" ou "protected" il est préférable de placer sa méthode dans la zone "protected".

3)La déclaration "public"

Une méthode "public" peut être appelée dans tout le Logiciel qui l'utilise. Une variable et une méthode publiques sont toujours accessibles. Ainsi un Constructeur est toujours dans la zone "public" de sa classe.

Si vous ne savez pas mettre une méthode en "public" ou en "protected", cherchez si votre méthode est utilisée plutôt par le développeur utilisateur en "public", que par le développeur de Composants en "protected". Le développeur de Composants modifie les facultés de votre Composant.

4)La déclaration "published"

Une méthode "published" peut être appelée dans tout le Logiciel qui l'utilise. Une variable et une méthode publiées sont toujours accessibles.

Vous voyez dans vos Composants que les propriétés "published" sont visibles dans l'inspecteur d'Objet de LAZARUS. La déclaration "published", et les propriétés dénommées par "property", permettent d'améliorer le Polymorphisme du PASCAL Objet. Ce procédé permet de manipuler les propriétés et méthodes "published", sans avoir à

connaître leur propriétaire. Vous pouvez réaliser des procédés de lecture indépendants du type d'Objet, grâce à l'unité "PropEdits".

Ainsi une méthode publiée peut être appelée indépendamment de la classe Objet qui l'utilise. Une propriété publiée est sauvegardée dans le formulaire, accessible indépendamment de la classe Objet qui l'utilise.

Voici comment récupérer une méthode publiée sans connaître sa classe :

```
// récupère une propriété d'Objet
// aComp_ComponentToSet : Composant cible
// as_Name      : Propriété cible
// a_ValueToSet         : Valeur à affecter
function fmet_getComponentMethodProperty ( const
aComp_Component : TComponent ; const as_Name : String ) :
TMethod ;
Begin
  if   assigned ( GetPropInfo ( aComp_Component, as_Name ))
  and  PropIsType      ( aComp_Component, as_Name , tkMethod)
  then Result := GetMethodProp   ( aComp_Component, as_Name );
End ;
```

Cette méthode permet de récupérer toute méthode publiée " as_Name" de tout Composant. Il suffit ensuite de forcer le type de votre événement.

Par exemple :

```
{$mode DELPHI} // Directive de compilation permettant
d'enlever les ^ de pointeurs, Elle est à placer au début de l'unité

Var MonClick : TNotifyEvent ;
Begin
  MonClick :=TNotifyEvent ( fmet_getComponentMethodProperty
```

```
( AComponent , 'OnClick' ));
  if assigned ( MonClick ) Then
    MonClick (  Acomponent );
End;
```

Cette Source permet d'exécuter tout événement "OnClick" de tout Composant, s'il est publié.

Dans l'inspecteur d'Objet vous pouvez voir aussi des événements. Les événements sont un appel vers une méthode, grâce à une variable de méthode. Le type de la variable méthode permet d'affecter des paramètres à une méthode.

Nous avons déjà utilisé les événements dans un chapitre précédent.
Un événement est une définition en variable d'une méthode. Il est cependant possible d'accéder à une méthode publiée, sans avoir à utiliser l'événement, en utilisant le type "TMethod". Ce type permet d'accéder à une méthode publiée dans un Composant.

```
{$Mode DELPHI}
var lmet_MethodeDistribueeSearch: TMethod;
Begin
 lmet_MethodeDistribueeSearch.Data := Self ;
 lmet_MethodeDistribueeSearch.Code :=
MethodAddress('MaMethodePubliee') ;
 ( lmet_MethodeDistribueeSearch as MonTypeMethod )
( monparametre ) ;
End;
```

Si ce que vous avez défini n'existe pas déjà on définit un type méthode ainsi :

```
TMaMethode = procedure(const monparametre:TypeParametre)
  of object;
```

5)Les propriétés publiées

Lorsqu'on publie une propriété dans un Composant référencé celle-ci se retrouve dans l'inspecteur d'Objet.

Il est préférable de créer une propriété au plus proche de l'Objet, en surcharge, voire en participation Libre, si la modification est utile pour la communauté.

Voici un bouton amélioré pour une utilisation particulière :

```
type
        TMyButton = class ( TSpeedButton )
       private
         FClickBouton : Boolean ;
         function getClickBouton:Boolean;
       public
        procedure Click ; override ;
       published
        property ClickButton : Boolean read getClickBouton write
FClickBouton default false ;
        property Glyph stored false ;
       end;
```

La propriété "ClickButton" permet de savoir si le bouton a été cliqué.
Le getter getClickBouton permet de réinitialiser FClickBouton à la lecture du clique.

On empêche ici d'enregistrer l'image du bouton, en surchargeant la

déclaration publiée du "TSpeedButton". Cela permet d'empêcher d'enregistrer dans chaque fiche l'image qu'on aurait chargée. Nous vous expliquons cette optimisation dans le chapitre suivant.

L'option "stored" à "false" n'enregistre pas la propriété dans l'"Inspecteur d'Objets", donc dans l'exécutable. "stored", qui est à "true" par défaut, permet l'enregistrement dans l'"Inspecteur d'Objets", si votre Composant est dans la palette des Composants.

6) Les Constructeurs et Destructeurs

Les variables simples, comme on l'a vu, se libèrent automatiquement. FREE PASCAL ne libère pas automatiquement certaines variables, notamment les classes Objets utilisées.
Un descendant de TComponent détruit automatiquement les Objets de type TComponent, qui lui ont été affectés. Donc, si vous utilisez un descendant de TComponent dans votre classe, il suffit, pour le détruire automatiquement, d'affecter le Propriétaire du Composant, comme étant votre classe d'Objet.

Tout descendant de TComponent peut être enregistré dans l'"Inspecteur d'Objets".

Pour les autres classes d'Objet, il est nécessaire d'utiliser les Constructeurs et Destructeurs, pour d'abord les initialiser, enfin les libérer.

Voici un exemple de Constructeur et de Destructeur :

```
Interface
uses Classes ;
```

```
 TReadText = class(TComponent)
 MonFichierTexte : TStringlist;
 public
  constructor Create(TheOwner : TComponent);
  destructor Destroy;
 end ;

implementation

constructor TReadText.Create(TheOwner : TComponent);
begin
 Inherited Create(TheOwner);
 MonFichierTexte := nil;
end ;
destructor TReadText.Destroy;
begin
 Inherited Create(TheOwner);
 MonFichierTexte.Free;
end;
```

Tout Objet défini dans une classe doit être initialisé à "nil". Le Constructeur initialise le "TStringList". Il peut alors être créé dès qu'on l'utilise, grâce à une méthode centralisée.

Le Destructeur du Composant libère les Objets mis en mémoire. Ainsi la méthode statique "Free", de la classe "TObject", vérifie si la variable est à "nil". Puis, s'il n'est pas à "nil", elle appelle le Destructeur de l'Objet de type "TStringList".

I) LE POLYMORPHISME

Le terme Polymorphisme est certainement le plus redouté. Pour le comprendre, voici la sémantique du mot : "poly" signifie plusieurs et

"morphisme" signifie forme. Le Polymorphisme traite de la capacité de l'Objet à posséder plusieurs formes.

Cette capacité dérive directement du principe d'Héritage, vu précédemment. En effet, un Objet hérite des champs et méthodes de ses ancêtres. On peut redéfinir une méthode, afin de la réécrire ou de la compléter.

Le concept de Polymorphisme permet de choisir, en fonction des besoins, quelle méthode ancêtre appeler. Le comportement d'un Objet polymorphe devient donc modifiable à volonté.

Le Polymorphisme est donc la capacité du système à choisir dynamiquement la méthode de l'Objet en cours.

On considère un Objet Véhicule et ses descendants Bateau, Avion, Voiture. Ces Objets possèdent tous une méthode Avancer, le système appelle la fonction Avancer spécifique, suivant que le véhicule est un Bateau, un Avion ou bien une Voiture.

Attention !

Le concept de Polymorphisme en orienté Objet ne doit pas être confondu avec celui d'Héritage multiple en Objet pur. L'Héritage multiple n'est pas supporté par le FREE PASCAL. Il permet à un Objet d'hériter des champs et méthodes de plusieurs Objets à la fois. Le Polymorphisme permet de modifier le comportement d'un Objet et celui de ses descendants au cours de l'exécution.

L'Héritage multiple possède lui aussi ses défauts. Il est encore plus lourd en mémoire que le Polymorphisme en orienté Objet.

1)L'Abstraction

L'Abstraction ne sert, en FREE PASCAL, qu'à centraliser plusieurs Objets, en héritant d'un Objet abstrait donc pas utilisable. Cet Objet abstrait possède en fait des méthodes dites "abstraites" ou "abstract", non implémentées dans l'Objet abstrait.

Voici à quoi l'Abstraction ressemble dans une classe d'Objet :

```
procedure Nager; abstract;
```

Cette déclaration est la seule à ne pas demander d'implémentation. Elle crée une erreur malgré tout, à l'exécution, si son descendant n'implémente pas la méthode "Nager".

Ainsi l'Objet abstrait qui nage n'a pas besoin de savoir comment il nage. Les différents descendants nagent en fonction du type classe créé. Une classe "Poisson" nage en oscillations horizontale. Une classe "Humain" nage en brasse, en crawle, en papillon, etc. On crée donc dans la classe Humain des méthodes secondaires.

Il reste à définir comment on choisit le type de nage en fonction de l'Objet créé. On peut définir le type de nage grâce à un nombre en paramètre.

2)Le type "interface"

Le type "interface" est le Polymorphisme en FREE PASCAL en son essence.

Il est impossible d'hériter de deux classes voire plus. Utilisez le type "interface", pour remédier à ce problème. Une classe FREE PASCAL

hérite d'une classe, avec, éventuellement, un ensemble d'interfaces.

Le type "interface" permet de créer des Objets polymorphes. Il peut s'ajouter à un Héritage. On crée le type interface comme si on crée une classe. Cependant il n'est possible que d'y ajouter des méthodes abstraites.

Nous parlons du type "interface" dans le chapitre suivant.

3)Polymorphe avec les propriétés publiées

Nous avons vu, précédemment, qu'il est possible d'appeler n'importe quel événement "Onclick", de n'importe quel Composant.

Le type classe ou Objet "TForm", c'est un Composant particulier. Le Composant et Objet "TForm" permettent de créer d'autres Objets Composants, à l'aide d'un fichier avec l'extension ".lfm". Vous pouvez donc changer de Composants dans vos formulaires, en changeant le type de votre Composant Source vers votre type destination. Changez alors le type de votre Composant, à la fois dans le fichier ".pas" de votre ficher, puis dans le fichier ".lfm".

Tout d'abord allez sur le formulaire à transformer.
Ajoutez un Composant "TLabel". Redimensionnez-le.

Allez sur la déclaration classe de sa formulaire dans le fichier ".pas".
Changez le type de son Composant vers le nouveau type de Composant, sans vous tromper. Vous pouvez par exemple lui affecter le type "TButton".

Allez sur le formulaire visuel et cliquez sur le bouton droit dessus. Choisissez "Afficher le Source".

Retrouvez votre Composant en faisant une recherche. Puis changez le type par le même type "TButton".

Fermez et rouvrez le formulaire. Des propriétés n'existent plus. LAZARUS demande de les effacer.

Votre nouveau Composant "TButton" a alors remplacé l'ancien, avec les propriétés identiques de ce dernier.
C'est la nomenclature LAZARUS qui a permis d'adapter les propriétés anciennes, vers le nouveau Composant.
Autrement dit, avant de créer une nouvelle propriété LAZARUS, il est nécessaire de connaître les propriétés standards de tout Objet LAZARUS. Une propriété LAZARUS est en anglais.

Les types Objets "interface" ne permettent pas de déclarer de véritables Objets. Elles ne font que centraliser la déclaration de méthodes. Les unités de fonctions permettent de centraliser le Code.

La déclaration "published" et les propriétés permettent d'accéder à d'autres Composants, sans avoir à connaître leur type "classe" ou "interface".
La déclaration "published" et les unités de fonctions résolvent partiellement le Polymorphisme. Elles permettent cependant une hiérarchisation de vos Sources, grâce aux paquets.

J)LES PROPRIÉTÉS

FREE PASCAL possède une syntaxe permettant de créer des Composants, qui sont des Objets avec des propriétés. Les propriétés permettent, si elles sont correctement créées, de mieux gérer le Polymorphisme. Les propriétés publiées permettent la rapidité de conception, en étant présentes dans l'"Inspecteur d'Objets".

Il ne faut pas être créatif pour nommer ses propriétés. Reprenez ce qui existe déjà au sein de LAZARUS, afin de mieux gérer le Polymorphisme. Une propriété doit être nommée du même nom que les propriétés de même nature.
Il est donc nécessaire de connaître les noms des propriétés des Composants de même nature, afin de créer des propriétés identiques.

K)L'UML POUR PROGRAMMER EN OBJETS

Des outils UML Libres existent pour automatiser la création de vos Logiciels orientés Objets.
UML en anglais signifie Unified Modeling Language, pour Langage de Modélisation Unifié. Ainsi tout outil UML peut être compatible avec d'autres outils UML. Vérifiez cependant si c'est bien le cas.

1)STAR UML

STAR UML est un outil Libre donc partagé en licence GNU/GPL.

Il a été fait sous DELPHI. Vous pouvez donc tenter de le traduire vers LAZARUS après avoir vérifié que quelqu'un ne le fait pas déjà. Dans ce cas, participez au projet de traduction, qui peut être intégré au projet STAR UML.

G)CRÉER SON SAVOIR-FAIRE

A)INTRODUCTION

Pour créer un savoir-faire respectez un seul principe : Éviter le copier-coller.
On crée donc avec cette technique un savoir-faire :

- Au début il n'y a que des unités de fonctions
- Puis on utilise et surcharge des Composants
- On crée des paquets de Composants
- On ouvre alors sa librairie aux autres API
- On automatise les paquets en une librairie
- La librairie nécessite peu de Code à créer ou aucun

B)CRÉER DES UNITÉS DE FONCTIONS

Vous pouvez vous aussi créer votre savoir-faire en créant d'abord des projets. Pour créer ces projets, nous vous apprenons à centraliser ce qui aurait été redondant. Évitez le copier-coller. Vous créez des unités de fonctions en centralisant un Code redondant.

Une unité de fonction c'est un regroupement de fonctions autour d'un même thème. Si la fonction que vous ajoutez n'est pas immédiatement associée à ce thème, n'hésitez pas à en trouver un nouveau. On crée alors une nouvelle unité.

C)LES COMPOSANTS

LAZARUS sans les Composants ne serait pas utile. LAZARUS permet

de créer rapidement des Logiciels grâce aux Composants. La programmation par Composants est rapide sur LAZARUS grâce à l'"Inspecteur d'Objets". Vous pouvez décupler votre vitesse de création avec LAZARUS.

Il est recommandé d'améliorer ou de créer des Composants, qui sont facilement mis en place. Ainsi votre savoir-faire ou vos Composants sont centralisés dans des paquets de Composants, puis dans une voire plusieurs librairies utilisant vos Composants. La librairie contient en plus des outils préparant le travail.

Un paquet est un regroupement de Composants. Les paquets sont installés rapidement sur LAZARUS. Les Composants sont visibles dans LAZARUS. Les Composants visuels sont directement modifiables dans LAZARUS.

Vos unités de fonctions peuvent ensuite améliorer des Composants par le procédé de l'Héritage, ou par la participation Open Source. Vous créez alors votre premier paquet, en apprenant l'Objet. Vos unités de fonctions sont utilisées et améliorées.

Vous participez alors à un projet Open Source après avoir vérifié la licence Open Source. Si la participation au projet est refusée il est nécessaire de s’interpeller sur l'utilité de cette participation. L'entreprise qui refuse la participation peut avoir d'autres objectifs que les vôtres. Vous pouvez alors surcharger le Composant.

LAZARUS c'est une interface facilitant la programmation. Un Composant LAZARUS est mis en place rapidement grâce à l'"Inspecteur d'Objets". Cet inspecteur permet d'affecter les propriétés de vos Composants, comme si vous étiez un simple utilisateur. Les propriétés permettant de développer sont un supplément au formulaire ouvert. Elles sont et doivent donc être situées en dehors du Code Source d'exécution.

Votre Composant peut être amélioré grâce à la communauté. Cela ne vous empêche pas par contre de toujours être compétitif en recherchant d'autres projets à créer, pour que vos ou d'autres Composants en héritent.

d) Développement Très Rapide d'Applications

Le Développement Très Rapide d'Applications c'est l'aboutissement de tout savoir-faire RAD. C'est l'utilisation de la Programmation Objet afin de séparer, ce qui est demandé par le client, du Logiciel en lui-même.

On supprime alors une étape de programmation, à savoir la transcription de l'analyse en Logiciel, car celle-ci est automatisée par la Programmation Objet, et les fichiers passifs. Les fichiers passifs contenant une analyse du Logiciel sont lus par le moteur DTRA, afin de créer le Logiciel.

Un Logiciel créé en Développement Très Rapide est toujours conforme à l'analyse, car l'analyse crée le Logiciel.

On crée des unités de fonctions, puis des Composants regroupés en paquets LAZARUS, puis une voire plusieurs librairies. Ces librairies sont indépendantes de LAZARUS, en étant entièrement automatisées grâce à des fichiers.

L'aboutissement est la prise en compte de ce qui est demandé par les clients, en créant des fichiers automatisant la librairie. Ces fichiers sont dédiés à ce qui est demandé. Ce sont les fichiers métiers. Ces fichiers peuvent être automatiquement créés par l'analyste.

Il n'y a alors plus d'inadéquation entre l'analyse et le Logiciel. En effet

il est difficile d'avoir une analyse identique au Logiciel demandé, sans service qualité ou sans automatisation.

E)Intérêts du DTRA

Mis à part le gain de temps, supprimer une étape de programmation permet d'améliorer aussi la qualité des Logiciels créés. L'analyse correspond toujours au Logiciel, puisque l'analyse crée le Logiciel. Les jeux de tests sont mis en place pour le moteur DTRA uniquement.

Même si un moteur DTRA nécessite au moins un ingénieur développeur, ce dernier pense fonctionnalités et pérennité du système.

Les fichiers passifs peuvent être lus par d'autres moteurs. Il est possible de créer des Forks de LEONARDI sur d'autres langages, grâce à des Frameworks de gestion, avec fiches et relations.

F)Créer un Framework DTRA

LEONARDI est un Framework ou savoir-faire DTRA JAVA, permettant de créer des Logiciels de gestion avec fiabilité.

LEONARDI utilise des fichiers de description de ses méta-données, ces données servant à définir comment on utilise une quelconque donnée, afin d'homogénéiser les logiciels, pour pouvoir les créer plus vite.

Il existe aussi les modèles de méta-données MICROSOFT, permettant de décrire encore plus d'entités de programmation.

Il existe un Framework ou savoir-faire DTRA LAZARUS Libre nommé XML Frames. C'est un savoir-faire Client/Serveur, qui crée des applications de gestion sur WINDOWS, GNOME, ou KDE.

Il est possible de créer son Framework Web de Développement Très Rapide, si la création de son Logiciel est répétitive. Ainsi l'automatisation permet de gagner du temps, ensuite.

Un Logiciel de gestion c'est un Logiciel gérant des processus. Il est composé d'un lien vers les données, de fiches, de relations, de filtres, de statistiques, de feuilles de calcul, etc. Il est simple à modéliser donc simple à automatiser.

Les Composants ou plugins permettent de créer une partie du Logiciel selon les spécificités demandées. Ces Composants ou plugins permettent d'étendre les capacités du moteur. Ils nécessitent une approche à la fois élémentaire, pour répondre à une partie infime de la demande, puis globale pour être inclus dans toute analyse. Ils sont utilisés dans l'EDI DTRA.

Il suffit alors d'utiliser une architecture pilotée par modèle, afin de centraliser ce genre de Logiciel dans un moteur contenant les Composants. Les Composants sont renseignés par les modèles analytiques, pouvant lire toute analyse. Les Logiciels, créés sans le moteur DTRA, peuvent uniquement renseigner le Composant formulaire.

Ainsi il y a au moins deux couches dans un moteur DTRA. Une fois cette étape passée les Logiciels peuvent être en partie traduits, par le reverse engeenering traduisant les données en un début de Logiciel. Vous créez alors la partie dynamique, non incluse dans les données.

On peut créer une troisième couche, qui va permettre d'analyser les Logiciels à mettre en place.

Il est donc possible de créer des plugins, dans les Frameworks Rails ou tout Framework de gestion, afin de permettre leur automatisation.

G) Créer un Composant

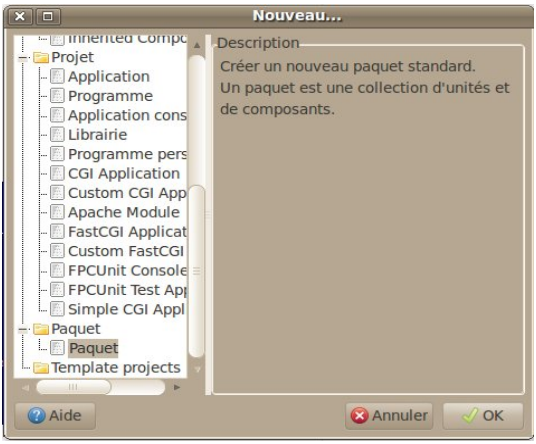
Nous allons changer la couleur d'un bouton de fermeture, et lui affecter une image.

Pour tout bouton créé, sera affectée une seule image. Ainsi l'exécutable est plus léger. Ce bouton permet de fermer un formulaire.

Le Composant qui va être créé va donc répondre à une demande technique : La taille de l'exécutable et la centralisation des Sources.

La centralisation des Sources va permettre de répondre à l'automatisation de son savoir-faire. Cela va donc permettre de répondre plus rapidement à la demande du client.

Pour créer votre paquet personnalisé Faire "Fichier" puis "Nouveau" puis "Paquet".



Créer un nouveau paquet dans l'"EDI LAZARUS"

Nommez le paquet "LightButtons".

"Ajoutez" la condition "LCL". Une condition est un paquet utilisé. Le paquet LCL contient tous les Composants standards.

"Ajoutez" un "Nouveau fichier" "Unité", puis sauvegardez votre unité sous le nom "u_buttons_appli".

Créez dans le répertoire du paquet un fichier vide "u_buttons_appli.lrs".

Enregistrez votre paquet en cliquant sur "Enregistrer", dans le projet de paquet. Affectez lui le nom "LazBoutonsPersonnalises".

Ajoutez ses deux types dans la partie interface de votre unité :

```
{$mode DELPHI}
interface
type
  IMyButton = interface
  ['{620FE27F-98C1-4A6D-E54F-FE57A06207D5}']
  End ;
```

On utilise le mode DELPHI afin de ne pas avoir à gérer les adresses de pointeurs. Le mode DELPHI est expliqué dans le chapitre suivant.

On déclare un type interface permettant d'indiquer que nos boutons supportent tous le type "IMyButton".

La chaine hexadécimale permet d'utiliser la méthode "support" dans toute classe afin de reconnaître le type "IMyButton". Ainsi nous retrouvons plus facilement nos boutons. Nous utilisons le Polymorphisme pour reconnaître nos propres boutons.

Voici le Code Source permettant de reconnaître nos boutons :

```
If UneClasse.Support ( IMyButton ) Then
  Begin
    ShowMessage ( 'C'est mon bouton !' );
  End;
```

En FREE PASCAL on définit au maximum ce que l'on fait. Les définitions permettent d'utiliser l'Objet, afin d'améliorer l'utilisation de nos Composants.

Juste après la définition de l'interface "IMyButton" placez ce Code :

```
  TMyClose = class ( TBitBtn,IMyButton )
    private
    public
     constructor Create(TheOwner: TComponent); override;
     procedure Click; override;
    published
     property Glyph stored False;
   End;
```

TMyClose est le descendant de TBitBtn. Il supporte l'interface IMyButton.

Créer la clause uses dans la partie "interface". Ajoutez ensuite les noms d'unités "StdCtrls, Classes, lresources".

L'unité "StdCtrls" contient les boutons. L'unité "Classes" contient le type composant. L'unité "lresources" vous permettra d'ajouter les images à vos boutons.

Le Constructeur Create et la méthode Click sont présents dans le Composant TBitButton par l'Héritage. Nous les surchargeons par le mot clé "override" suivi d'un ";".

Grâce au Polymorphisme LAZARUS vous pouvez changer l'ancêtre

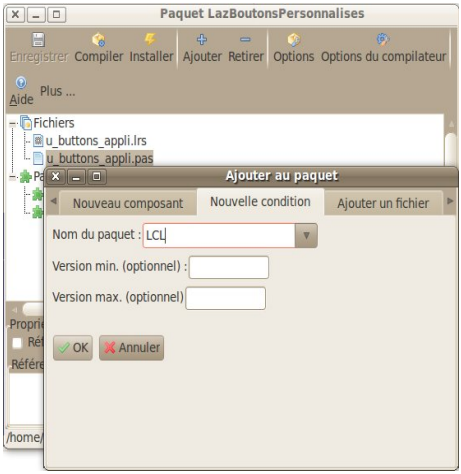
"TBitBtn" par un ancêtre bouton possédant un "Glyph". D'autres boutons avec "Glyph" existent en dehors du projet LAZARUS. Vous pouvez donc facilement changer de type Bouton par la classe que vous créez.

Tous les Composants LAZARUS possèdent le Constructeur Create avec comme paramètre le Propriétaire du Composant. Notre Composant est donc automatiquement détruit lorsque son parent se détruit.

Aussi tous les Composants descendants de la classe TLCLComponent possèdent la méthode "Click" sans aucun paramètre. Une méthode qui peut être surchargée possède le mot clé "virtual".

Le Composant ancêtre définissant la méthode utilise donc le mot clé "virtual" sur la méthode créée. Vous pouvez donc facilement vérifier si c'est bien TLCLComponent qui définit "Click" en maintenant "Ctrl" enfoncée puis en cliquant sur chaque type ascendant utilisé.

Pour chercher l'unité du bouton "TBitBtn" cherchez dans tous les répertoires du Code Source de LAZARUS la bonne unité. Sinon cette unité s'appelle "StdCtrls". Cette unité est incluse dans le paquet LCL. Dans le paquet "Ajoutez" la "Condition" "LCL".



Ajouter la condition "LCL" au paquet

Vous pouvez "Compiler" le paquet pour trouver des erreurs. Si votre Code Source est bon vous pouvez appuyez sur "Ctrl"+ "Shift" + "C". Cela va créer les deux méthodes.

Maintenant nous allons remplir les deux méthodes surchargées dans la partie "implémentation" :

```
implementation

uses Forms;

 // Créer des constantes permet d'éviter de se tromper  pour leur réutilisation
const
   CST_FWCLOSE='TMyClose';  // Nom du fichier Image

{ TMyClose }
procedure TMyClose.Click;
begin
  if not assigned ( OnClick )
```

```
  and ( Owner is TCustomForm ) then
    Begin
      ( Owner as TCustomForm ).Close; // Fermeture du formulaire
      Exit;
    End;
  inherited;
end;

constructor TMyClose.Create(TheOwner: TComponent);
begin
  inherited Create ( TheOwner );
  if Glyph.Empty then
    Begin
 // Charge le fichier Image
      Glyph.LoadFromLazarusResource ( CST_MYCLOSE);
    End;
end;
```

Ajout la constante CST_MYCLOSE à la partie interface de votre unité de composant. Affectez lui le nom de votre classe créée, à savoir "TMyClose".

Voilà l'essentiel de notre Composant "TMyClose". La procédure surchargée "Click" ferme le formulaire, s'il n'y a pas d'événement de "Click" sur le Bouton dans le formulaire. Il suffit que cette Source soit exécutée deux fois dans l'Application pour que cette Source ait remplit un objectif d'automatisation.

On passe la méthode "Click" héritée par la méthode "Exit". On a vérifié, avant d'éluder la méthode "Click" de l'ancêtre, qu'il n'y avait que la gestion de l'événement "OnClick" que l'on évitait. Nous n'éludons effectivement pas l'événement "OnClick" en vérifiant s'il existe. Ainsi nous permettons d'utiliser le bouton de fermeture uniquement pour l'image centralisée.

Le Code spécifique est automatisé avec le maximum de Composants possédant différents objectifs. On se rend compte que, bien que le Composant permette de centraliser, celui-ci augmente aussi la taille de l'exécutable.

Si la fonction de fermeture était sur chaque événement de chaque bouton, cela alourdirait l'exécutable. Aussi, si les conditions techniques de fermeture du formulaire changeaient, il faudrait changer le Code Source de chaque bouton ne descendant pas de "TMyClose".

1)Choix de l'image

Le Code Source du Constructeur va créer une erreur à l'exécution car il n'y a pas de fichier image.

Le Constructeur de notre bouton charge l'image du bouton grâce au fichier "lrs".

Choisissez votre image de fermeture sur un site Web contenant des images Libres avec une licence "Art Libre", ou "Creative Common" avec ou sans "by" avec ou sans "SA".

Vérifiez la licence. Si vous vous posez des questions sur les licences vous pouvez contacter un Groupe d'Utilisateurs LINUX local.

Les licences Libres possèdent plus de facilités quant aux modifications. Vous avez une définition de Libre dans le glossaire.

Le fichier image doit être un fichier "XPM", de 24 pixels sur 24 pixels, avec comme nom le type de la classe de notre bouton. Cela permet de charger l'image du bouton dans la palette de composants.

Convertissez avec GIMP votre image au format "XPM". Pour faire cela sauvegardez l'image avec GIMP, en remplaçant l'ancienne extension de fichier par "XPM" sans enlever le point. L'extension ce sont les dernières lettres après le dernier point du fichier.

Allez dans votre dossier contenant "LAZARUS". Allez dans le répertoire "tools". Compilez le projet "lazres", si ce n'est pas déjà fait.

Ouvrez un "Terminal", ou allez dans votre accessoire "Ligne de commande".

Copiez-y le lien vers l'exécutable "lazres". Copiez le lien vers le fichier ressources "lrs" qui va être créé. Puis copiez le lien vers le fichier XPM.

Vous pouvez éventuellement ajouter d'autres images, si vous voulez créer d'autres boutons.

Nous allons ajouter le fichier ressource à l'unité.

Allez tout à la fin de l'unité de votre Composant. Insérez cette Source avant le "end." final :

```
initialization
  {$I u_buttons_appli.lrs}
end.
```

L'unité "ressources" est nécessaire.
La directive {$I} ajoute le fichier ressource au Code Source.

2)Enregistrement du Composant

A l'enregistrement du Composant et en fonction de la surcharge effectuée, celui-ci voit certaines de ses propriétés reconnues par LAZARUS, afin d'éditer rapidement le Composant.

Il est possible d'utiliser des propriétés, voire de les surcharger, dans son unité d'enregistrement de Composants.

Un Composant est un Objet descendant de l'Objet TComponent. Tout Composant LAZARUS, descendant de TComponent, peut être accessible dans la palette de Composant.

Pour qu'un Composant soit enregistré dans une palette, voici le Code Source à affecter à une unité d'enregistrement du Composant.

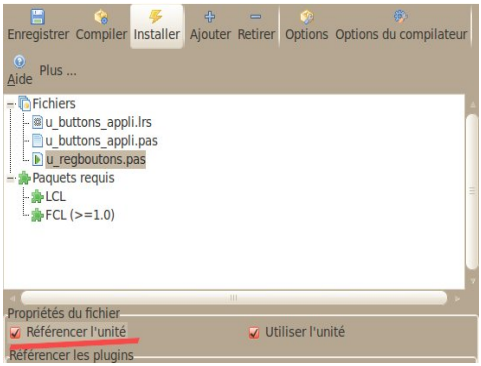
Créez cette nouvelle unité dans votre nouveau paquet :

```
Unit u_regboutons;

interface

uses Classes;

procedure Register;

implementation

uses u_buttons_appli;

procedure Register;
begin
  RegisterComponents('MesBoutons', [TMyClose]);
end;
```

```
end.
```

La procédure "Register" doit être exécutée par le paquet qui installe le Composant. Vérifiez pour cette unité si la case à cocher "Référencer l'unité" est cochée.

L'unité d'enregistrement ne doit jamais être utilisée dans votre programme. En effet le Code Source de développement est inutile pour le programme exécutable.



"Référencer l'unité" est coché pour l'unité

3)Modifier ou Surcharger ?

Si votre Composant ajoute un objectif au Composant initial, il est préférable de créer une autre unité surchargeant le Composant. Cette dernière permet de ne pas avoir de Code inutile, pour le premier objectif du Composant.

La modification d'un Composant se fait en deux étapes. Pour pouvoir modifier un Composant, l'accord de l'auteur est nécessaire. Vous avez

alors accès au gestionnaire de version permettant de le modifier.

Sinon que le Composant peut être abandonné, et sa licence permet de le modifier. S'il n'y a aucune licence, contactez l'auteur.

Sinon vous ne faites qu'ajouter des options mineures. Seulement, sans l'accord de participation de l'auteur, vous ne pouvez pas correctement mettre à jour le Composant, Vous êtes alors seul responsable de ses évolutions. Il suffit souvent de contacter l'auteur, qui sera en général content de vous aider à améliorer son savoir-faire.

H) Surcharger un Composant

N'hésitez pas à surcharger tout Composant, surtout si vous disposez des Sources qui permettent de trouver plus facilement les contournements pour les erreurs d'Héritage. Lorsque vous surchargez un Composant, vous avez accès aux méthodes protégées. Ces méthodes sont inaccessibles sans la surcharge.

Il faut quelquefois être astucieux, afin de surcharger correctement certaines méthodes, de certains Composants. Vous pouvez indiquer à l'auteur du Composant les modifications nécessaires à un Héritage facilement effectué. Évitez d'utiliser des contournements, car ils peuvent provoquer de nouvelles erreurs à la mise à jour.

I) Créer une librairie homogène

Un paquet de Composants est mis en place rapidement. En voulant être encore plus rapide vos paquets deviennent une librairie complète paramétrable au minimum voulu. Vous sublimez alors votre capacité d'adaptation en ne créant que l'utile.

Ce qui sera personnalisé sera paramétrable. Le reste se configurera automatiquement.

Une librairie fournit des outils et des procédés pour créer son Logiciel sans perdre de temps. Il faudra améliorer l'IDE LAZARUS ou bien éviter d'utiliser l'IDE avec des fichiers de configuration. Ces fichiers pourront être idéalement des fichiers analytiques UML ou MERISE. Ces fichiers vous permettront d'être indépendant de LAZARUS, en utilisant les mêmes savoir-faire sur d'autres outils de programmation.

J) Le Libre et l'entreprise

LAZARUS contredit les professeurs ou institutions, indiquant que les Composants nécessitent plus de temps en maintenance, qu'en gain de temps à l'utilisation. Un Composant LAZARUS fait gagner beaucoup de temps, grâce à l'"Inspecteur d'Objets" et au Code Source de développement.

Votre Composant s'il est bien documenté peut devenir un atout fiable, faisant connaître votre société. Il peut aussi être sauvé de l'abandon, grâce à une communauté toujours plus avare de Composants Libres. Un Composant qui devient Libre veut soit être populaire, soit se faire connaître pour survivre. Cela n'empêche pas forcément une activité commerciale.

Créer un Composant Libre permet de, certes faire connaître votre Composant, mais aussi de trouver le moyen de savoir si on reste le meilleur dans son domaine, en créant une communauté sondant les utilisateurs et contributeurs.

N'espérez pas des contributeurs qu'ils collaborent sans être rémunérés. Vous serez par contre peut-être rémunéré pour des modifications

importantes. Attendez de votre communauté une expertise sur votre savoir, des aides pour vous promouvoir. Ces aides ne sont peut-être pas celles que vous souhaitez. Elles permettent donc de trouver de nouveaux marchés.

K) TESTER SON SAVOIR-FAIRE

Un composant est au cœur de votre savoir-faire. Si vous modifiez votre composant cela implique forcément des réactions en chaine dans votre logiciel. Il est donc très intéressant de mettre en place des jeux de tests en surchargeant vos paquets de composants.

L) L'EXEMPLE

FPCUnit vous permet de tester les logiciels FREE PASCAL. Nous allons mettre en place des jeux de tests pour notre bouton de fermeture.

Dans le répertoire de votre composants créez un dossier "tests". Dans le menu "Fichier", créez une "Nouvelle" "Application de tests FPCUnit",en remplaçant le nom du jeu de tests "TestCase" par "TestButtons". "Enregistrez" le projet en le nommant "fpcunitbuttons", dans le dossier créé.

Ajoutez-y dans la clause "uses" du projet l'unité "Interfaces". Cette unité est nécessaire pour gérer les formulaires, dont nous avons besoin pour tester le bouton de fermeture du formulaire.

Dans le menu "Projet", puis "Inspecteur de Projet", ajoutez avec "+" votre paquet en "Nouvelle Condition". Ou bien ajoutez le dossier "../" dans les "Options du projet", puis "Options de compilation", puis le dossier des librairies. Cela permet de tester le composant sans avoir à

l'installer.

1)Le formulaire

Ajoutez un nouveau formulaire nommé "testform". Dans la partie "interface", ajoutez l'unité "u_buttons_appli".

Dans le formulaire, nous allons mettre en place le bouton que nous avons créé.
Le formulaire possède ces méthodes et variables, dont certaines existaient déjà dans l'ascendant de la fiche :

```
{ TTestForm }

TTestForm = class(TForm)
public
  Bouton : TMyClose;
  procedure DoClose ( var AAction : TCloseAction ); override;
  procedure RunBouton;
  destructor Destroy; override;
end;
```

Nous n'avons pas besoin de tester le formulaire, mais le bouton. Nous créons donc le bouton manuellement, sans utiliser l'IDE :

```
procedure TTestForm.RunBouton;
begin
  Bouton := TMyClose.Create(Self);
end;
```

Il est intéressant d'utiliser d'autres moyens de création, que ceux utilisés par le développeur, afin de vérifier le composant autrement. Ne négligez pas cependant les jeux de tests à effectuer.

Nous créons l'auto-destruction du formulaire, afin de vérifier plus tard s'il est correctement détruit :

```
procedure TTestForm.DoClose(var AAction: TCloseAction);
begin
  inherited DoClose(AAction);
  AAction := caFree;
end;

destructor TTestForm.Destroy;
begin
  inherited Destroy;
  TestForm := nil;
end;
```

Nous testons le bouton et ne testons pas le formulaire. Nous nous assurons donc que le formulaire est correctement affecté à "nil", même si dans les sources il est censé être correctement affecté à "nil".

2) La classe de tests unitaires

Voici à quoi ressemble notre classe de tests :

```
{ TTestButtons }

 TTestButtons= class(TTestCase)
 protected
   procedure SetUp; override;
   procedure TearDown; override;
 published
   Procedure Tests;
   procedure TestHookUp;
 end;
```

Les deux méthodes publiées sont affectées dans les jeux des tests par l'initialisation d'unité suivante, située tout à la fin de votre unité de classe de tests :

```
initialization
  RegisterTest(TTestButtons);
end.
```

Les tests sont récupérés par le projet de tests avec cette méthode.

Voici la méthode de création des Objets testés :

```
procedure TTestButtons.SetUp;
begin
  // Initialize
  Application.CreateForm( TTestForm, TestForm );
  TestForm.RunBouton;
end;
```

Dans votre savoir-faire il est préférable de mettre les commentaires en anglais, afin d'internationaliser vos composants.

Le formulaire a été créé. Il est préférable de le détruire par nous même :

```
procedure TTestButtons.TearDown;
```

```
begin
  // Freeing
  TestForm.Free;
end;
```

Passons aux jeux de tests :

```
procedure TTestButtons.Tests;
begin
  SetUp;
  // testing
  AssertTrue('Testing Glyph of
TMyClose',assigned(TestForm.Bouton.Glyph));
  TestForm.Bouton.Click;
  AssertTrue('TestForm of TMyClose not
visible',Assigned( TestForm ));
  TearDown;
end;
```

L'implémentation précédente est une des deux méthodes publiées qui sera appelée pour tester le bouton. Il est donc nécessaire de créer les composants avant les jeux de tests avec "SetUp", pour les détruire à la fin avec "TearDown".

La classe TTestCase possède elle-même des méthodes permettant d'effectuer les tests en les enregistrant dans le projet.

Un jeu de test FPCUnit commence par "Assert", suivi du type de jeu de tests.

"AssertTrue" va donc tester si le deuxième paramètre est bien à "True". Quant au premier paramètre il permet de savoir quel jeu de test nous réalisons. Si vous voulez tester une valeur utilisez "AssertEquals".

Nous testons bien le fait que le "Glyph" soit affecté à la création, sans

avoir besoin de formulaire. Nous testons aussi la fermeture du formulaire par le bouton "TMyClose". Nous ne testons pas cependant le fait que le Glyph ne soit pas enregistré dans le formulaire.

Cette méthode publiée permet de tester si les jeux de tests sont effectivement exécutés :

```
procedure TTestButtons.TestHookUp;
begin
  Fail('Test of button finished');
end;
```

La méthode "Fail" envoie un message d'erreur. Elle devra donc être vue à l'exécution. Elle pourra être enlevée en utilisant d'autres utilitaires de tests plus évolués.

3)Le projet de tests

Dans le fichier "lpr" créez cette classe :

```
  { TMyTestRunner }

  TMyTestRunner = class(TTestRunner)
   // override the protected methods of TTestRunner to customize its behavior
  public
    procedure RunTests;
  end;
```

Cette méthode permet de démarrer tous les jeux de tests :

```
{ TMyTestRunner }

procedure TMyTestRunner.RunTests;
begin
  DoTestRun(GetTestRegistry);
end;
```

Ensuite nous créons l'exécution du projet :

```
var
  Application: TMyTestRunner;

begin
  Application := TMyTestRunner.Create(nil);
  Application.Initialize;
  Application.Title := 'FPCUnit Console test runner';
  Application.RunTests;
  Application.Free;
end.
```

Comme dans un projet standard, nous créeons un Objet central, qui gère les Objets de tests cette fois ci.

Après avoir initialisé, puis nommé notre logiciel, nous démarrons les tests. Enfin nous libérons notre logiciel.

M) EXERCICE

Créez un rapport de tests, grâce à la documentation de FPCUnit.

N) CONCLUSION

Les composants doivent être testés. Mais ils permettent de gagner

beaucoup de temps grâce à votre IDE RAD.

Vos composants permettent de gagner beaucoup de temps, tout en créant des logiciels personnalisés et intuitifs.

A la fin vous disposez d'un moteur créant vos logiciels grâce à une analyse uniforme. Les gains de temps sont minimes avec le moteur, mais ce dernier permet de fiabiliser votre création de logiciels.

H)DE PASCAL VERS FREE PASCAL

A)INTRODUCTION

FREE PASCAL permet de créer des savoir-faire multiplate-forme à partir de savoir-faire WINDOWS comme DELPHI ou TURBO PASCAL.
A partir de ces savoir-faire ou Logiciels, on crée des librairies FREE PASCAL. Cependant cela demande l'adaptation d'une partie des Sources.

B)DE TURBO PASCAL VERS FREE PASCAL

Pour transférer un Logiciel TURBO PASCAL textuel il faut juste réutiliser les bonnes unités. Elles se trouvent nécessairement dans LAZARUS, ou FREE PASCAL.
Pour transférer un savoir-faire TURBO PASCAL graphique :

- Insérez la partie graphique dans une fenêtre et adaptez la taille graphique à la fenêtre
- Recréez la partie graphique avec les Composants visuels

C)GESTIONNAIRE DE DONNÉES

Si vous souhaitez beaucoup d'utilisateurs ou de données créez un lien vers un Système de Gestion de Base de Données, comme FIREBIRD, SQLITE, MY SQL, voire ORACLE.

FIREBIRD et SQLITE sont des S.G.B.D. mi-lourds suffisants pour tout

Logiciel installé facilement. Ils s'utilisent en Embarqué.

MY SQL permet lui de faire évoluer votre serveur de données en serveurs de données répartis nommés "clusters". ORACLE lui permet de créer un seul serveur de données très lourd.

Il est possible d'utiliser des gestionnaires de données mi-lourds, si votre entreprise est répartie. On crée un réseau maillé permettant de dupliquer les informations. Le siège prend le pas sur les filiales.

D)DE DELPHI VERS LAZARUS

Lorsque son Logiciel est créé en DELPHI, le transfert consiste à utiliser des Composants LAZARUS à la place des Composants DELPHI. Il est possible que son Logiciel reste compatible DELPHI, grâce aux directives de compilation. Cela permet d'alléger l'exécutable WINDOWS.

Vous pouvez trouver les Composants qui vous manquent, grâce à une recherche sur un site Web de Composants comme www.LAZARUS-components.org.

E)LES DIRECTIVES DE COMPILATION

Les instructions de compilation permettent de porter son programme vers les différentes plates-formes. Elles définissent une façon de compiler l'unité. Elles sont donc à placer au début de l'unité.

1)Compatibilité DELPHI

Voici une instruction de compilation FREE PASCAL :

```
{$mode DELPHI} // Compilation compatible DELPHI
```

Il existe des DELPHI gratuits, permettant de réaliser des applications non commerciales. Si quelqu'un vous demande d'être compatible DELPHI vous pouvez le faire grâce à une vieille licence de DELPHI 7, voire DELPHI 6.

Les directives de compilations permettent de compiler uniformément l'unité, en éludant des Sources incompatibles avec certaines plates-formes. Ainsi, les Sources d'une plate-forme sont éludées lorsqu'on compile sur une autre plate-forme.

L'instruction de compilation, qui permet à FREE PASCAL de rendre le Code Source compatible DELPHI, n'est pas lisible par DELPHI.
On peut y ajouter une directive de compilation pour DELPHI, si on l'utilise.

Nous avons vu la notion de pointeur dans le chapitre sur la **Programmation Procédurale avancée**.
Le mode DELPHI permet de supprimer la notion d'adresse de pointeur. Cela n'est pas gênant, au contraire, car DELPHI protège les pointeurs.

Les pointeurs sont des adresses en mémoire, redirigeant vers un autre endroit de la mémoire. Ils permettent de manipuler les Objets. En mettant en mode DELPHI vous pouvez oublier les pointeurs en partie, car le compilateur FREE PASCAL n'est pas entièrement compatible avec ce mode, surtout avec les dernières grammaires FREE PASCAL.

Le développeur a besoin de manipuler des Objets, pas de gérer des pointeurs. Le pointeur peut cependant être utile pour scruter les tableaux. Mais PASCAL Objet permet toujours cela.

Il est possible en Pascal d'éluder totalement les affectations de pointeurs, grâce à cette directive à ajouter à tout formulaire :

```
{$IFDEF FPC}
{$mode DELPHI}
{$R *.lfm}
{$ELSE}
{$R *.dfm}
{$ENDIF}
```

Ces directives peuvent être mises en tout début d'unité et une seule fois. Elles définissent comment va être compilée l'unité.

Ici l'instruction de compilation "Mode DELPHI" est exécutée si on utilise le Compilateur FREE PASCAL. Aussi on charge le fichier "lfm". Sinon on charge les Composants contenus dans le fichier "dfm". Si votre unité n'est ni un formulaire, ni un module de données, vous pouvez enlever les directives de chargements de fichiers "lfm" et "dfm".

Un fichier "dfm" est une correspondance DELPHI de fichier "lfm" avec LAZARUS. Les fichiers "dfm" et "lfm" contiennent les informations des Composants chargés dans un module de donnée, ou un formulaire. Ces Composants sont visibles dans l'"Inspecteur d'Objets".

Cette directive de compilation permet de placer une instruction DELPHI dans le Code :

```
{$IFNDEF FPC}
var DirectorySeparator : Char = '\';
{$ENDIF}
```

Ici on définit le caractère de séparation de répertoire quand on est sur DELPHI.

2)Compatibilités avec les plates-formes

Cette Source permet de déclarer sur WINDOWS le séparateur de répertoires des chemins :

```
{$IFDEF WINDOWS}
DirectorySeparator := '\';
{$ENDIF}
```

Cette Source permet de déclarer sur les systèmes UNIX le séparateur de répertoires :

```
{$IFDEF UNIX}
DirectorySeparator := '/';
{$ENDIF}
{$IFDEF LINUX}
DirectorySeparator := '/';
{$ENDIF}
```

En effet lorsqu'on crée un Logiciel multiplate-forme, il est nécessaire de penser aux différences entre les différentes plates-formes. Aussi penser aux erreurs pouvant se produire permet de sécuriser son Logiciel.

Nous ne sommes pas des machines. Pourtant nous créons des Logiciels répondant à une logique. Si cette logique n'est pas respectée notre programme n'est pas sûr.

F)Traduction de Composants

Pour traduire un Composant DELPHI vers LAZARUS, il est préférable

que ce Composant reste compatible DELPHI, grâce aux directives de compilation. Cela permet de participer au projet existant, et de centraliser les Sources.

Remplacez le Code graphique, et les liens vers les librairies WINDOWS, par du Code Source LAZARUS. LAZARUS possède une large bibliothèque de Codes Sources. Des Composants Libres DELPHI peuvent être traduits vers LAZARUS.

Le Code graphique LAZARUS utilise les différentes librairies Libres de chaque plate-forme. Ces différentes librairies sont homogénéisées en des librairies génériques.

Les unités à ne pas utiliser sont les unités spécifiques à une seule plate-forme, comme les unités :

- win pour WINDOWS
- gtk pour LINUX
- carbon pour MAC OS
- unix pour UNIX

Si vous êtes obligé d'utiliser une de ces unités, sachez que vous aurez peut-être une unité générique dans la prochaine version de LAZARUS. Votre Composant sera compatible avec la seule plate-forme de l'unité non générique utilisée.

Si vous ne trouvez pas ce qu'il vous faut, recherchez dans les Sources LAZARUS ou FREE PASCAL. Contactez sinon un développeur LAZARUS. Il vous indique alors ce que vous pouvez faire.

G) CARACTÈRES ACCENTUÉS

Si vous observez que les caractères accentués ne sont pas gardés, c'est à cause d'un problème de traduction en général. Il est nécessaire d'avoir

des fiches en UTF8. Pour faire cela vous pouvez écrire un programme utilisant la méthode "AnsiToUTF8", ou bien vous pouvez utiliser un éditeur de texte ouvrant en Ansi, et enregistrant en UTF8.

Sinon c'est à cause des caractères ANSI ou UTF8. En effet il existe des caractères deux fois plus longs, les caractères UTF16. Ces caractères permettent une lecture presque internationale, pour les pays possédant un alphabet de lettres.

Vérifiez si le "TStringlist" utilise des "WideStrings". Ce sont des chaines avec des caractères deux fois plus volumineux.

I)CRÉATION DU LIVRE

A)HISTORIQUE

En 2008 il n'existait pas de livre français sur LAZARUS. Matthieu GIROUX a donc créé des essais, puis des articles sur LAZARUS. Puis des images de LAZARUS ont été ajoutées. Le livre s'est ordonné en chapitres.

Il a fait connaître son livre sur www.developpez.com et www.framasoft.net, en diffusant des articles devenus Libres.

Le livre évolue toujours en 2011, avec les améliorations de LAZARUS et les nouveaux paquets LAZARUS.

Les chapitres sur la programmation procédurale en FREE PASCAL sont un article retravaillé de Jean-Michel BERNABOTTO.

Les images d'écrans ont été créées à partir de la version 0.9.28 ou 0.9.30 de LAZARUS. Les dessins font soit partie de LAZARUS, ou bien du projet OPEN CLIPART.

J)CHAPITRES

A)RETROUVER UN CHAPITRE

K)GLOSSAIRE

A)RETROUVER UN MOT

Abstraction	L'Abstraction permet le Polymorphisme en orienté Objet en créant une classe abstraite qui sera renseignée avec ses classes filles.
Application	Logiciel permettant de réaliser une ou plusieurs tâches.
Bogue ou Bug	Anciennement, ce terme anglais désigne la punaise, qui mange les circuits électroniques. Cette punaise créait des erreurs. Les programmeurs se sont déchargés alors des erreurs dans le Code, pour les attribuer à cet animal.J'ai mis en place le paiement par carte avec paypal api

Client/Serveur	Interface logicielle connectée à un serveur de données. Un serveur peut fournir l'interface. On appelle cette hiérarchie le Trois Tiers : – Navigateur Web – Serveur d'interface Web – Serveur de données
Code	Ensemble de 0 et de 1 créés par le compilateur. Le Code sert à agir sur l'ordinateur.
Compilateur	Programme reprenant les différentes Sources de Code pour les compiler en un langage machine, des 0 et des 1 exécutables par le processeur.
Composant	Partie réutilisable de ses Logiciels.
Composant RAD	Composant mis en place facilement.
Constructeur	Méthode paramétrée permettant de créer les Objets d'un Objet et d'initialiser l'Objet. On déclare cette méthode particulière en commençant par "constructor".
Destructeur	Méthode paramétrée permettant de détruire les Objets d'un Objet. En PASCAL Objet on déclare cette méthode particulière en commençant par "destructor".
Embarqué ou Embedded	L'Embarqué c'est placer un Logiciel dans du matériel électronique.
Encapsulation	Déclarations permettant de réutiliser une partie d'un Objet.
Fonction	Procédure ou Méthode paramétrée retournant une variable.
Fork	Copie d'un projet Libre voire Open Source

Framework	Savoir-faire Logiciel réutilisable permettant de réaliser un certain nombres de tâches.
Héritage	Procédés permettant à un Objet d'hériter d'un autre Objet.
IDE ou EDI	Integrated Development Environment ou Environnement de Développement Intégré. LAZARUS est un IDE. Il permet de créer un Logiciel grâce à un ensemble d'outils homogènes.
Librairie	Ensemble de Composants et d'outils permettant des fonctionnalités ou un gain de temps.
Libre	Une création ou un Logiciel Libre c'est une création pour laquelle la licence Libre permet de : – Étudier la Source. – Utiliser la création librement. – Modifier le Logiciel ou la création. – Dupliquer tout ce qui a été fait. – Diffuser commercialement ou pas ce qui a été créé. Une licence Libre peut être : – Virale : La communauté grandit facilement car on est obligé de diffuser la Source modifiée ou le Logiciel l'utilisant. La licence Libre virale la plus connue est la licence GPL qui est utilisée pour partie avec LAZARUS. – Entièrement Libre. La création peut être modifiée commercialement sans avoir aucune contrainte de diffusion forcée de la Source. La licence BSD est une licence entièrement Libre. – Du domaine public car les auteurs ont

	abandonné les droits commerciaux ou l'œuvre a vu sa période de fin de droits d'auteur terminée. En général il est possible de s'approprier le projet du domaine public.
Logiciel	Ensemble de traitements permettant la résolution de tâches.
Méthode	Procédure ou fonction dans un Objet. Une méthode peut être encapsulée ou surchargée.
Objet (Analyse ou Programmation Objet)	L'analyse Objet est une analyse des systèmes d'informations qui est proche de l'humain, tout en offrant une architecture compréhensible par la machine. On représente toute entité abstraite ou concrète par un Objet qui dispose de propriétés Objet. Les quatre genres de propriétés d'un Objet sont l'Héritage, le Polymorphisme, l'Encapsulation, l'Abstraction. En respectant ces propriétés, on crée un programme d'Objets disposant de spécificités propres au langage utilisé.
Open Source	Projet à Sources partagées. Il peut être possible de participer au projet en fonction de la licence et de la disponibilité de l'entreprise.
Paquet	Projet LAZARUS permettant de réunir et d'installer des Composants.
Patch	Modification à ajouter à un exécutable. Le patch augmente ou diminue la taille de l'exécutable.
Pointeur	Un pointeur permet de stocker les variables pour les retrouver. Un pointeur est une adresse pointant vers un endroit de la mémoire.
Polymorphisme	Procédé permettant par différentes manières de réutiliser plusieurs Objets en même temps. Le Polymorphisme est résolu par le type Objet

	"interface" permettant de dénommer les méthodes communes de futurs Objets.
Procédure	Bout de Code commençant par la déclaration implémentée et paramétrable "procedure", délimitée par un "Begin" et un "End".
Procédural (Langage)	Les langages procéduraux sont des langages non Objets. Il sont architecturés selon des unités de procédures et fonctions s'appelant entre elles. Le langage PASCAL était au début un langage procédural.
Propriétaire ou Privateur	Un Logiciel Propriétaire ne diffuse aucune source. Il est restreint. WINDOWS est Privateur dans le sens où c'est un Système d'Exploitation en location. En effet, vous êtes restreint par la nécessité de rester compatible avec les dernières Applications payantes.
RAD ou DRA	Rapid Application Development. Développement Rapide d'Applications grâce à un IDE ou une librairie.
Source	Ce qui permet de créer un Logiciel ou un Composant. Il est intéressant d'utiliser des projets Open Source ou Libres afin de ne pas travailler dans le flou.
Type	Définition d'une variable permettant de la manipuler facilement grâce au compilateur.
UML	Unified Modeling Language ou Langage Unifié de Modélisation. Boîte à outils de modèles d'analyse servant à analyser un projet basé sur la programmation Objet.
Variable	Partie de la mémoire, définie par un type, pouvant varier à l'exécution. Une variable stocke des

	chiffres, lettres, adresses, Objets.
Web	Réseau maillé pouvant contenir un nombre presque illimité d'ordinateurs, grâce au protocole IPV6. Les ordinateurs utilisent un navigateur Web, qui peut accéder aux applications participatives, dites Web 2.0.

ISBN 978295312516

Éditions LIBERLOG
Éditeur n° 978-2-9531251

Dépôt Légal RENNES 2010

Imprimé en France en Avril 2011 par :
JOUVE
1, rue du Docteur-Sauvé
BP 3
53101 Mayenne cedex