Lazarus - Premier exemple : développement rapide

par Matthieu Giroux (LAZARUS Components)

Date de publication : 18/07/2010

Dernière mise à jour : 19/03/2011

LAZARUS vous permet de créer facilement vos logiciels. Voici un exemple permettant de comprendre l'intérêt des composants LAZARUS.

1 - A lire

Les mots que vous ne comprenez pas peuvent être trouvés dans le glossaire à la fin du livre.
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 crochets indiquent des mots volés à LAZARUS ou à votre environnement.

1.A - Objectifs

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 donnera la démarche pour créer très rapidement un logiciel.

1.B - Licence

2 - Biographie

Matthieu GIROUX aime l'écriture, cette recherche de la vérité à force d'utiliser des moments de réflexion.
Il s'intéresse au Développement Rapide d'Applications afin de trouver de meilleures techniques de programmation. Sauter une ou des étapes dans 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. Il faut rechercher le partage pour s'enrichir et ainsi enrichir les autres.

3 - Du même auteur

- L'astucieux LINUX
- Nos Nouvelles Nos Vies
- Poèmes et Sketchs - De 2003 à 2008
- France - Fonctionnement de notre société
- Créer et référencer son Site Web
- Comment écrire des histoires

Disponibles sur www.comment-ecrire.fr

4 - Débuter avec Lazarus

4.A - Pourquoi choisir Lazarus

Si vous voulez apprendre la programmation d'interfaces homme-machine, LAZARUS est fait pour vous. LAZARUS c'est une utilisation des librairies libres les plus fiables avec une interface facile à utiliser. DELPHI est beaucoup enseigné dans les écoles françaises. Nous souhaitons le même destin à LAZARUS qui ne possède pas de licence payante. LAZARUS est un logiciel libre.
LAZARUS c'est un EDI, un Environnement de Développement Intégré, qui est multi-plateformes et compatible DELPHI. Vous pourrez donc avec adapter des composants DELPHI pour que vos logiciels deviennent multi-plateformes. DELPHI a été créé en 1995. Le projet LAZARUS commence en 1999. Il est actuellement préférable d'utiliser LAZARUS plutôt que les composants KYLIX, un DELPHI sous LINUX créé en 2001. En effet KYLIX, a été abandonné dès sa deuxième version.

On utilise LAZARUS pour créer des logiciels Client/Serveur multi-plateformes. Il faut cependant participer au projet LAZARUS afin, par exemple, de permettre les tests détaillés. Si vous voulez créer ce genre de logiciel, vous pouvez compter sur les librairies permettant de réaliser vos interfaces exécutables sur vos plateformes préférées. Mais il existe aussi des composants libres ou payants sur le web.
Si vous voulez créer votre jeu multi-plateforme, LAZARUS possède des librairies OPEN GL permettant de créer des jeux 2D ou 3D. On utilisera LAZARUS pour créer des logiciels graphiques multi-plateformes. Il existe des librairies permettant de lire des formats de fichiers 3D libres ou propriétaires.

4.B - JAVA vs LAZARUS

Indéniablement, LAZARUS manque encore de librairies. Cependant, il existe toutes sortes de composants libres ou gratuits sur DELPHI facilement traduisibles.
LAZARUS, ce sont les atouts de JAVA sans des contraintes de JAVA. On s'aperçoit que les destructeurs permettent une meilleure optimisation de la mémoire. Au lieu d'installer un serveur avec TOMCAT qui devra être optimisé, il est plus judicieux de détecter les fuites mémoires au sein des destructeurs afin d'optimiser les variables créées. En effet, libérer la mémoire dès qu'elle est inutilisée permet déjà d'optimiser. LAZARUS permettra bientôt de détecter les fuites mémoires.
Il n'y a pas de JVM sous LAZARUS. Votre exécutable LAZARUS est aussi suffisamment complet pour être indépendant de librairies. Les librairies utilisées par LAZARUS ne sont en général pas nécessaires à votre exécutable.
Vous ne connaissez peut-être pas les BEANS sous JAVA. La partie composants des BEANS JAVA ne permet pas de gagner autant de temps que les composants RAD. On appelle cela les LCL sur LAZARUS, pour Librairie de Composants Lazarus (Lazarus Component Library). Dans LAZARUS même les composants non visuels sont dans cette librairie. Un EDI JAVA nécessite plus de temps de maintenance et de développement qu'un framework LAZARUS, à cause d'une gestion rendue complexe par la programmation JAVA nécessitant d'utiliser toujours des classes, ainsi que par la réinvention d'un nouvel EDI.
Dans LAZARUS ou DELPHI, tout composant est facilement remplaçable par un autre. Il suffit pour cela de changer le type de composant dans le fichier PASCAL et dans le code du formulaire.
Comme vous le savez sans doute, l'inspecteur d'objet permet de mettre en place rapidement tout composant. Vous gagnerez un temps précieux à créer des composants en Développement Rapide d'Applications, contrairement à JAVA.
Tout comme JAVA, vous pouvez utiliser l'orienté objet pour créer des classes qui seront les objets instanciés de votre logiciel. Mais vous avez des alternatives comme les unités de fonctions. A quoi bon créer une classe statique. En PASCAL Objet, vous pouvez créer des unités avec des fonctions et procédures sans avoir à créer une classe statique. Il est par ailleurs recommandé d'utiliser un maximum d'unités pour que l'exécutable prenne moins de place.

4.C - Du PASCAL Objet

LAZARUS est basé sur un langage nommé PASCAL objet. Ce langage possède une grammaire et des instructions PASCAL basées sur l'algorithmique. Le PASCAL est donc enseigné pour cette raison particulière.
Le langage PASCAL Objet est l'un des plus simples 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 multi-plateformes FREE PASCAL, point de départ de départ de LAZARUS. Le compilateur FREE PASCAL évolue maintenant grâce à LAZARUS.
Nous vous apprendrons à connaître les instructions permettant de réaliser votre logiciel adapté à vos besoins. Il faudra pour aller loin connaître l'objet et les différentes architectures logicielles.

4.D - Destruction de variables

Le langage PASCAL Objet permet de créer des objets grâce à ses variables typées. Pour créer du code source propre, vous devez respecter la syntaxe ainsi que les règles de destruction de vos variables en mémoire.
Les variables simples comme les nombres, les chaînes de caractères, les tableaux typés sont détruits dès que leur objet ou unité parente est détruite.
Vous ne trouverez pas chez LAZARUS de destruction automatique des variables d'Objets. Seulement les composants Objets se détruisent dès que leur fenêtre ou module est détruit.

On voit que l'auto-destruction, si besoin, n'est pas aussi fiable qu'elle le prétendait. LAZARUS permet donc de créer des interfaces hommes machines fiables si vous informez correctement vos constructeurs et destructeurs d'Objets.

4.E - La communauté

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

4.F - Installer LAZARUS

LAZARUS évolue continuellement. Il faudra éviter certaines versions.
Le site de www.sourceforge.net ne diffuse que des logiciels Open Source. 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.

4.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 aura intérêt à mettre à jour **LAZARUS** afin de disposer des dernières améliorations libres. LAZARUS gère des numéros de version.
Le Versioning **LAZARUS** se présente avec des chiffres disposés de cette façon : **A.B.CC-D**
Le premier chiffre ou nombre « A » présente une version majeure de **LAZARUS**. Par exemple, la version 1.0.00-0 de LAZARUS permettra de traduire tout composant DELPHI 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.
Les deux troisièmes chiffres présentent la maturité de la version mineure. Si le chiffre est pair, la version est utilisable car elle a été testée. Si le chiffre 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. Elle n'est donc pas testée. Les versions à troisième nombre impair ne sont jamais recommandées. Elles servent à créer les versions testées.
Le quatrième chiffre ou nombre montre une création de la version normalement paire à partir d'une base plus récente. En fait, le quatrième chiffre ne vous apportera presque rien exceptée la possibilité de créer des composants sur chaque plateforme plus facilement, ce qui est un haut niveau de programmation.

4.H - Télécharger LAZARUS

Vous pouvez donc maintenant télécharger 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 la 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 sur le site web de sourceforge à **http://sourceforge.net/projects/lazarus**.

4.I - Installer LAZARUS sous WINDOWS

LAZARUS s'installe en vous demandant de relier les fichiers PASCAL ou LAZARUS au logiciel LAZARUS. Notez le répertoire d'installation de LAZARUS.

4.J - Installer LAZARUS sous LINUX

Une version moins récente est disponible dans votre gestionnaire de paquets. Pour disposer de la version plus récente, il faut ajouter le serveur de tests à votre gestionnaire de paquets. Il faut avant vérifier si la version de www.sourceforge.net est plus récente que celle de votre gestionnaire.
Si vous possédez un LINUX DEBIAN ou un LINUX UBUNTU votre version de LAZARUS est dans le gestionnaire de paquets SYNAPTIC. Il suffit d'installer le paquet « lazarus » graphiquement ou en tapant cette commande en mode « root »(Administrateur) :

```
apt-get install lazarus
```

Sinon il faut installer 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. Il faut faire attention à la version reconnue par LAZARUS.
Pour reconstruire LAZARUS, il faut démarrer LAZARUS comme ceci :

```
su root  startlazarus
```

Si vous voulez travailler sur les composants LAZARUS, le mieux est de rendre le répertoire LAZARUS à votre compte. Seul votre compte peut alors construire 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
```

4.K - Apprendre LAZARUS en s'amusant

Après avoir créé un nouveau projet, 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.
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 Resource ou Ressource d'un Projet LAZARUS. Cette ressource centralisant le projet permettra 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. Le formulaire dans son ensemble appartient par contre au projet LAZARUS. LAZARUS c'est un Environnement de Développement Intégré regroupant des composants tous visuels.

4.K.a - Le nom d'unité

Du code PASCAL Objet commence toujours par la clause « Unit » reprenant lettre pour lettre le nom du fichier PASCAL. Chaque instruction PASCAL est terminée par un « ; ». Il existe de rares exceptions pour lesquelles le compilateur vous avertira.

4.K.b - La clause « uses »

Ensuite, vous aurez sûrement à réutiliser des existants. La clause « uses » est faite pour cela. La clause « uses » permet d'aller chercher les existants créé par l'équipe LAZARUS ou bien par un développeur indépendant de LAZARUS. Lorsque vous ajoutez un composant grâce à l'onglet de composants cela ajoutera automatiquement l'unité du composant dans la clause « uses ». Vous aurez sans doute à ajouter les unités communes LAZARUS comme LCLType ou LCLTInf. Ces librairies remplacent certaines unités de DELPHI comme Windows.
Pourquoi n'a-t-on pas gardé certaines unités DELPHI ?
Les librairies LAZARUS possèdent une nomenclature sur les librairies PASCAL. Une librairie possédant le mot « win » est une librairie système spécifique à la plateforme 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 plateformes indiquant les unités à éviter. Dans les sources de LAZARUS vous voyez des unités commençant pas ces plateformes.

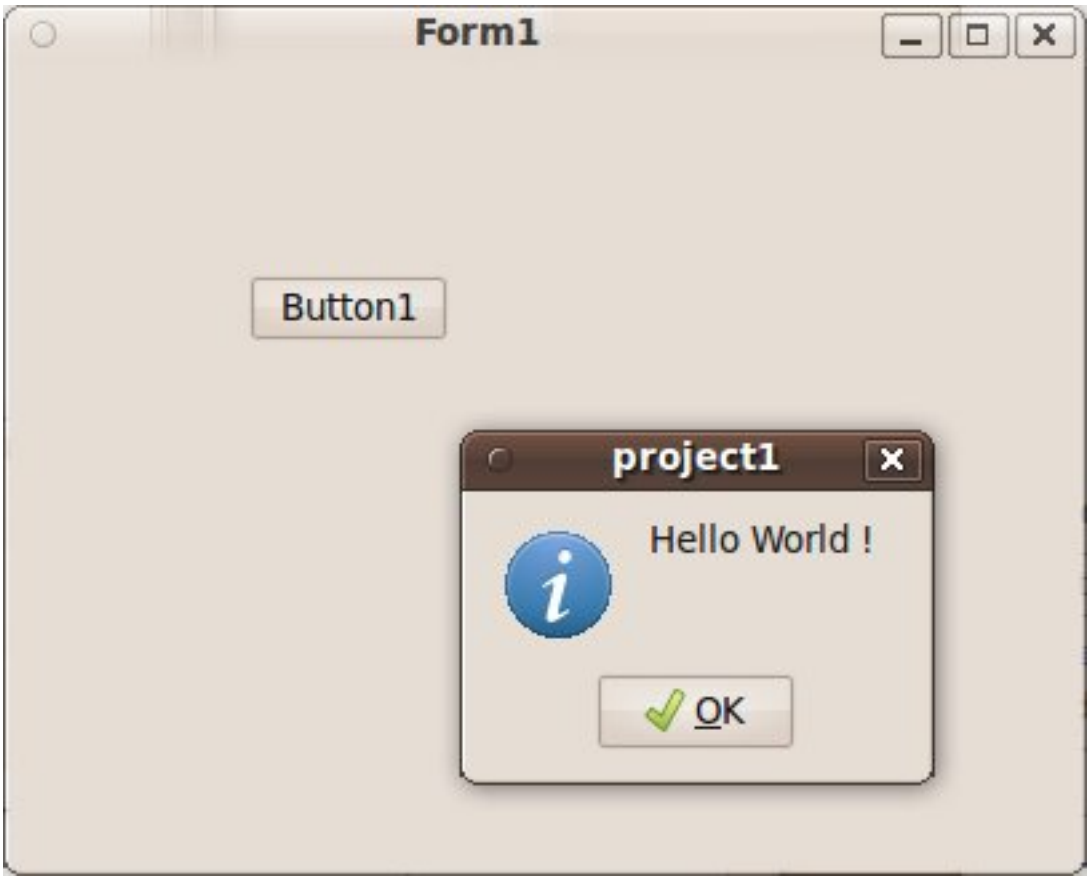
4.K.c - L'Exemple

L'exemple consiste à créer une fenêtre permettant d'afficher « Hello World ! ».
Nous allons éditer un nouveau projet. 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 ». Elle se termine comme chaque instruction par un « ; ».
En respectant la présentation LAZARUS, tapez uniquement « di ».
Ce qui va suivre ne fonctionne que si votre code source est correct des unités utilisées jusqu'à votre ligne éditée.
Appuyez en même temps d'abord 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 irez toujours vers la source première en faisant de cette manière. LAZARUS ira chercher ce qu'il pourra aller chercher grâce aux liens de vos projets et un code correct.
Vous pouvez voir que la clause « uses » de « Dialogs » contient d'autres unités. Celles-ci seront elles aussi ajoutées à votre projet si elles ne l'étaient pas déjà. En effet, des unités obligatoires seront toujours utilisées.
Toute écriture de code ajoute des informations à votre exécutable avec en plus des unités qui seront intégrées. Elles permettent le maximum d'indépendance. Votre programme compilé contiendra beaucoup de code mais sera indépendant.
Grâce aux onglets de l'éditeur de source, 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 et sauvegardez votre unité en commençant par « u_ ». Cela indique comme avant que l'on sauvegarde une unité. Les fichiers seront ainsi regroupés au même endroit. Ensuite, on mettra le thème de l'unité comme « 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. Acceptez.
Votre nom d'unité est donc « u_hello ». Elle peut être rajoutée dans une autre unité du projet dans une clauses « uses » d'une autre unité du projet. 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 ». Cliquer de nouveau sur « F12 » permet de revenir sur l'« éditeur de source ».
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 verrez 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 va permettre 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 » et 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 et pourtant la proprété « Caption » a bien changé le texte de votre « TButton ».
Allez sur l'onglet « Evènements » de l'« Inspecteur d'Objets ». Double cliquez sur l'évènement « OnClick ». Cela fait le même effet que d'appuyer sur les trois points : Vous créez un lien vers une procédure dans votre « Editeur 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 devez voir la procédure « ShowMessage ». Vous voyez que cette procédure simple possède un seul paramètre chaîne. Sélectionnez cette procédure avec les flèches et entrée ou la souris.
Vous avez maintenant à ajouter le paramètre. Les paramètres des procédures sont intégrés grâce aux parenthèses. Ouvrez votre parenthèse. Les chaînes sont ajoutées grâce en les entourant par un caractère « ' ». L'instruction se terminera par un « ; ». Il faut donc écrire :

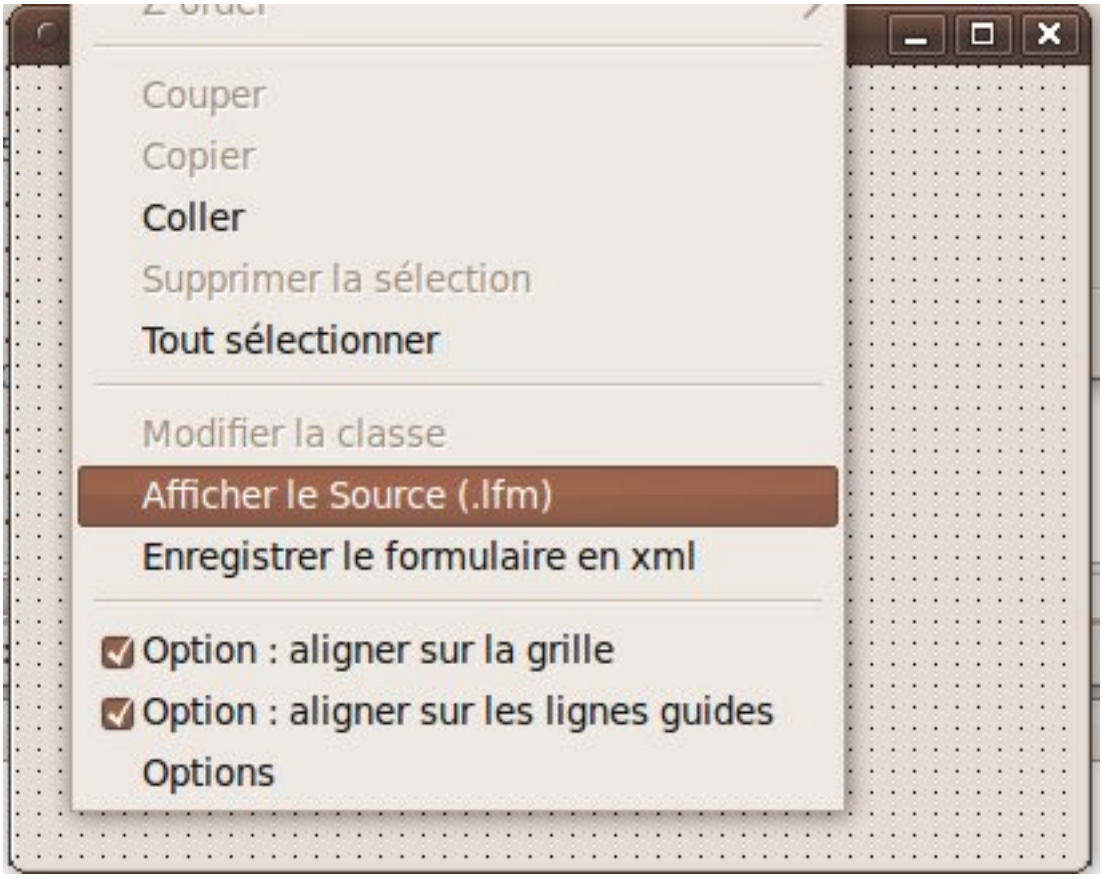
```
Begin
  ShowMessage ( 'Hello World' );
End;
```

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, ce que cela donne ci-après :



4.K.d - 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'objet 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 ».



Les modifications apportées au fichier permettent entre autre d'intervertir des composants.
Pour cela il faut avoir relié le paquet du composant au projet grâce au gestionnaire de projet.

4.L - 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 clairs 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 c'est présenter le code source afin de le comprendre comme clair sans avoir à le regarder de près.
L'indentation des éditeurs PASCAL est simple. En voici un résumé :

- Chaque instruction ou noeud (var, type, class, if, while, case, etc.) commence au début de la ligne
- Chaque imbrication d'un noeud 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 passer du code source plus facilement. Les développeurs qui regardent du code source avec cette présentation seront 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 deux espaces vers la gauche.
- Ctrl + K + I permet de décaler votre sélection 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 obligatoirement avant chaque code source de fonctions ou procédures
  - Ainsi les commentaires avant chaque fonction ou procédure seront vus dans l'éditeur FREE PASCAL.
- Des commentaires dans le code source si on y trouve un bug
  - Si on trouve un bug dans le code source, il est possible que ce bug revienne s'il n'y a pas de commentaires.
- Des commentaires dans le code 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 oubliera 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 de mettre 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.

4.L.a - 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 en-tête, puis des déclarations publiques, puis le code source exécutable. Toute unité PASCAL se termine par « end. ».

4.L.a.1 - La déclaration Unit

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

4.L.a.2 - Les instructions de compilation

Les instructions de compilation permettent de porter son programme sur les différentes plateformes. Elles définissent une façon de compiler l'unité. Elles sont donc à placer au début de l'unité.
Voici une instruction de compilation FREE PASCAL :

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

4.L.a.3 - Les directives de compilation

Les directives de compilations permettent de compiler uniformément l'unité en éludant des sources incompatibles. Ainsi les sources d'une plateforme seront éludées lorsqu'on compilera sur une autre plateforme.

4.L.a.3.a - Compatibilité DELPHI

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. En effet DELPHI, compile avec une erreur sinon.

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

L'instruction de compilation « Mode Delphi » est exécutée si on utilise le compilateur FREE PASCAL. Sinon on charge les composants contenus dans le fichier « dfm ». Un fichier « dfm » est une correspondance de fichier « lfm » en 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 :

```
{$IFNDEF FPC}
instruction_delphi;
{$ENDIF}
```

4.L.a.4 - Compatibilités avec les plateformes

Cette source permet de déclarer le séparateur de répertoires dans les chemins :

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

Cette source permet de déclarer le séparateur de répertoires dans les chemins :

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

4.M - Les fichiers ressources

A la fin du fichier PASCAL 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 images compressées permettent de réduire la taille de l'exécutable. Il faut éviter de répéter les mêmes images à différents endroits de votre logiciel. C'est pour cela que les fichiers ressources et les composants LAZARUS existent.

Vous avez peut-être des ressources bitmap non compressées; alors, l'idéal serait de les traduire grâce à GIMP en fichier XPM. Pour faire cela, ouvrez votre fichier BITMAP avec GIMP puis enregistrez le fichier en remplaçant l'extension BMP par XPM.
Dans C:\LAZARUS sous WINDOWS ou /usr/lib/lazarus sur LINUX, il y a l'outil lazres dans le répertoire tools de LAZARUS. Après avoir compilé le projet, cet outil permet de créer le fichier ressource lrs avec comme premier paramètre le fichier ressources. Puis les images peuvent être indéfiniment ajoutées en séparant chaque paramètre par un espace.

4.M.a - Les déclarations publiques

Les déclarations publiques comprennent obligatoirement le mot clé « interface » indiquant le début des déclarations publiques.
Avant le prochain mot clé, vous pouvez y placer l'instruction « uses » permettant d'utiliser dans les déclarations publiques les unités déjà existantes.
Vous pourrez ensuite placer toute déclaration constante, type, variable, fonction ou procédure. Toute déclaration ajoute du code exécutable.
Les variables alloueront soit un pointeur soit un type simple.
Les constantes publiques pourront être centralisées dans une ou plusieurs unités.

4.M.b - Le code source exécuté

Le code source commence par le mot clé « implementation ». Dans le code source exécuté, on peut placer les mêmes déclarations que celles données précédemment. Seulement elles ne seront plus publiques donc pas utilisables ailleurs.
En plus des déclarations déjà définies, on place dans cette partie le code source exécuté grâce aux fonctions et procédures.
La procédure ShowMessage vous a permis d'afficher une boîte avec un message et un bouton. Si vous regardez le code source de ShowMessage, 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 indique cela. Le code source que vous avez facilement ajouté était dans la partie « implementation ».

4.M.c - 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.
Le code source placé après le mot clé « initialization » permet de créer des objets toujours en mémoire.
Le code source placé après le mot clé « finalization » permet détruire les objets en mémoire à la fermeture du programme.

4.M.d - Fin de l'unité

La fin de l'unité est définie par « end. »

4.N - Touches de raccourcis de complétion

La complétion permet d'écrire à votre place. C'est indispensable pour tout programmeur voulant être aidé à coder.

Ctrl+Maj+Espace	Afficher la syntaxe des méthodes :

	`procedure TForm1.ButtonClick(Sender: TObject);`
	`begin`
	`  ShowMessage(const Msg:String);`
	`end;`

Ctrl+Maj+flèche haut ou bas	Dans l'éditeur : allez 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:=truc +machintruc;`
	`end;`
	A
	`begin`
	
	`  if X >0 then x=0;`
	
	`  if X<0 then x=-1;`
	
	`  A:=truc +machintruc;`
	`end;`
	
	Sans le faire ligne par ligne ?
	Sélectionnez les lignes puis faites 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 fléchée	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 «

	ifeb » puis Crtl + j. Les lignes correspondant à if then else s'écrivent à votre place. Les abréviation disponibles sont visibles en faisant Crtl + j.
Ctrl+Maj+ 1 à 9 et Ctrl+ 1 à 9	Placer des marques (des signets) dans un source pour pouvoir y revenir ultérieurement. Vous êtes sur un bout de source et vous voulez 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 (= déplace 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

4.N.a - 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

4.O - 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 **procedure** 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

4.P - Touches de raccourcis de débogage

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

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

4.Q - 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+I	Indente un bloc de la valeur spécifiée par la boîte à options Indentation de bloc dans la page Editeur de la boîte de dialogue Options d'environnement
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 **procedure** depuis un fichier
Ctrl+K+U	Désindente un bloc de la valeur spécifiée par la boîte à options Indentation de bloc dans la page Editeur de la boîte de dialogue Options d'environnement
Ctrl+K+V	Déplace le bloc sélectionné
Ctrl+K+W	Ecrit le bloc sélectionné dans un fichier
Ctrl+K+Y	Supprime le bloc sélectionné

4.R - Touches de raccourcis de l'environnement

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