

LES COMPOSANTS DE LAZARUS

Objectifs : dans ce chapitre, vous allez apprendre à étendre les possibilités de **Lazarus** en créant et installant un paquet et les composants qu'il comprend.

Sommaire : Définitions – La création d'un paquet – La création d'un composant pour un paquet – Le composant **TGVURLLabel** – Le test d'un composant hors installation dans l'EDI – L'installation d'un paquet – L'exploitation du composant créé – Créer des composants de qualité – La modification d'un paquet

Ressources : les programmes de test sont présents dans le sous-répertoire *components* du répertoire *exemples*.

DEFINITIONS

Un *composant* est une classe qui descend de la classe **TComponent** et qui peut être installée dans la palette de **Lazarus**. Grâce à la gestion des flux, les propriétés de cette classe peuvent être sauvegardées et récupérées.

Un *paquet* est une collection d'unités et de composants, contenant les informations nécessaires à leur compilation et à leur emploi par des projets, d'autres paquets ou **Lazarus** lui-même. L'intérêt essentiel des paquets est par conséquent de rassembler en leur sein des éléments qui sont liés. C'est ainsi que sont proposées des bibliothèques chargées d'une tâche particulière : accès à Internet, composants visuels, outils de programmation, etc.

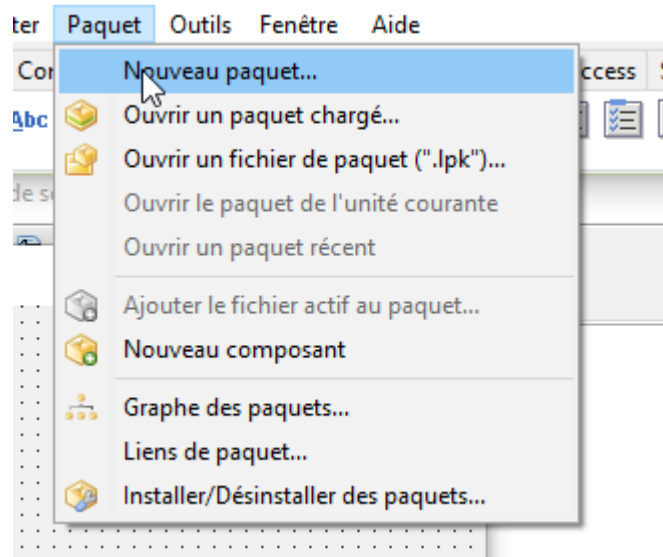
LA CREATION D'UN PAQUET

L'apprentissage intégrera la création d'un composant réellement utilisable : **TGVURLLabel**. Ce nouveau composant est un descendant de **TLabel** muni de l'aptitude à lancer le navigateur par défaut pour afficher une page spécifique du Web.

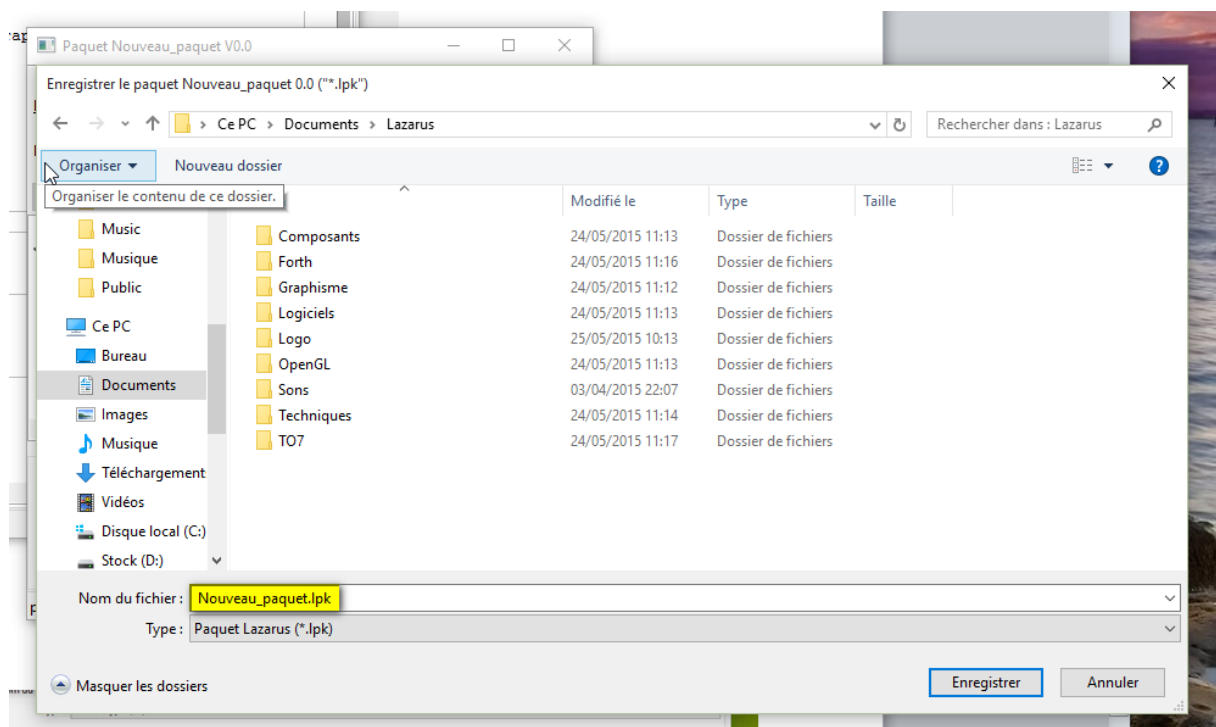
Dans un premier temps, il prendra une forme peu satisfaisante : l'essentiel est de comprendre comment créer un composant et l'intégrer à la palette de **Lazarus**. Ensuite, il s'agira de critiquer cette ébauche pour mettre en lumière quelques bonnes pratiques relatives à la création de nouveaux composants.

[Exemple Comp-01]

Afin de créer un nouveau paquet, à partir du menu principal de **Lazarus**, choisissez tout d'abord « *Paquet → Nouveau paquet...* » :

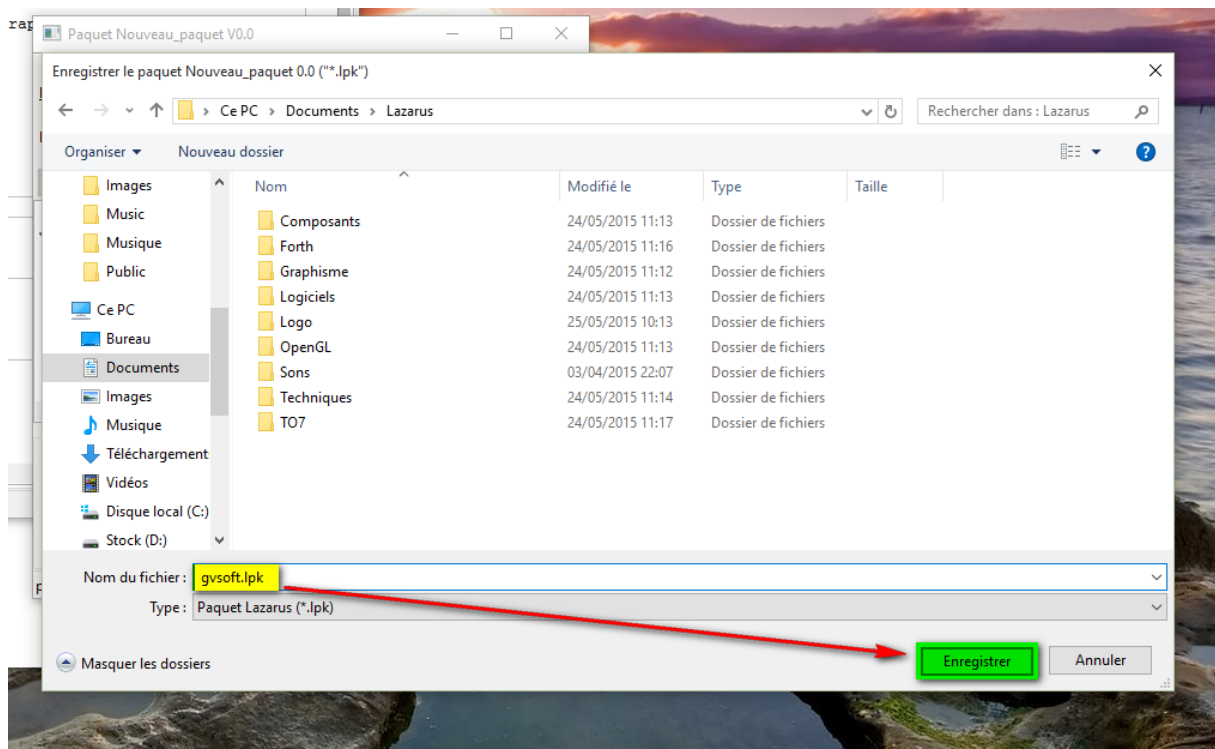


Lazarus vous demande le nom du paquet que vous comptez créer. Par défaut, il propose le nom de fichier *NouveauPaquet.lpk* qu'il est conseillé de modifier pour rendre plus parlant son utilisation future :

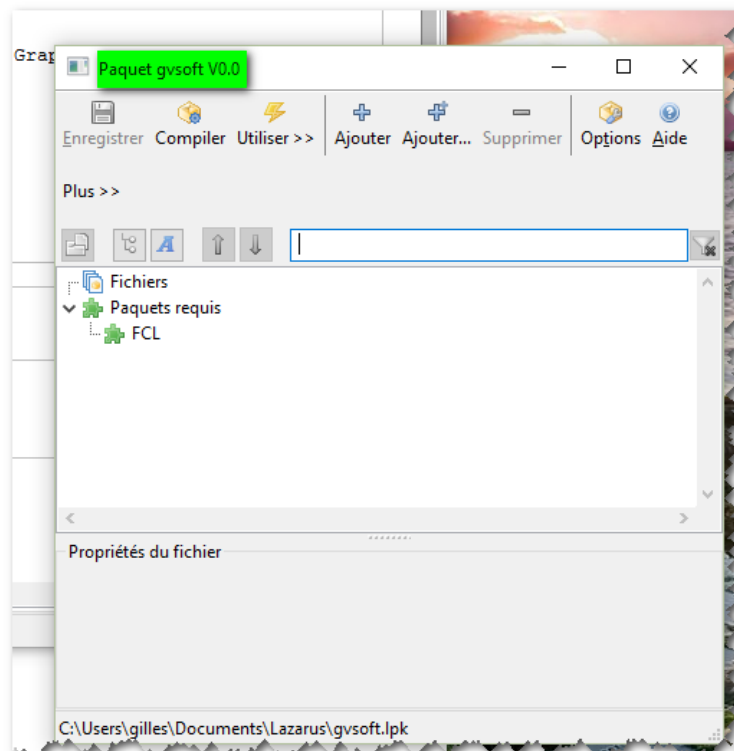


Votre travail consistant à créer un paquet comprenant l'ensemble des productions de la société GVSoft, changez donc le nom par défaut en tapant *gvsoft.lpk*.

Cliquez alors sur le bouton « Enregistrer » :

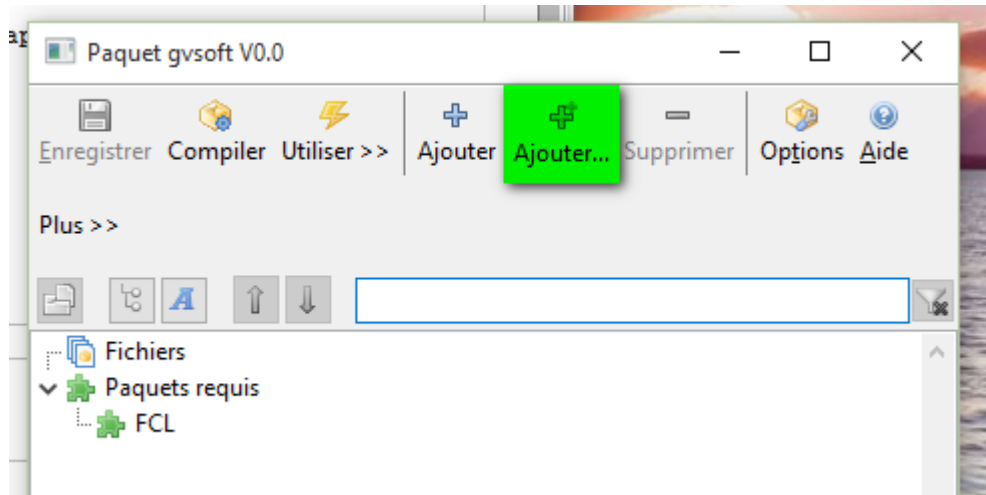


Lazarus active une nouvelle fenêtre dont la barre de titre contient le nom du paquet nouvellement créé :



Le paquet ne comprend pas de fichier, mais déjà un autre paquet nommé *FCL* qui sert de base de travail.

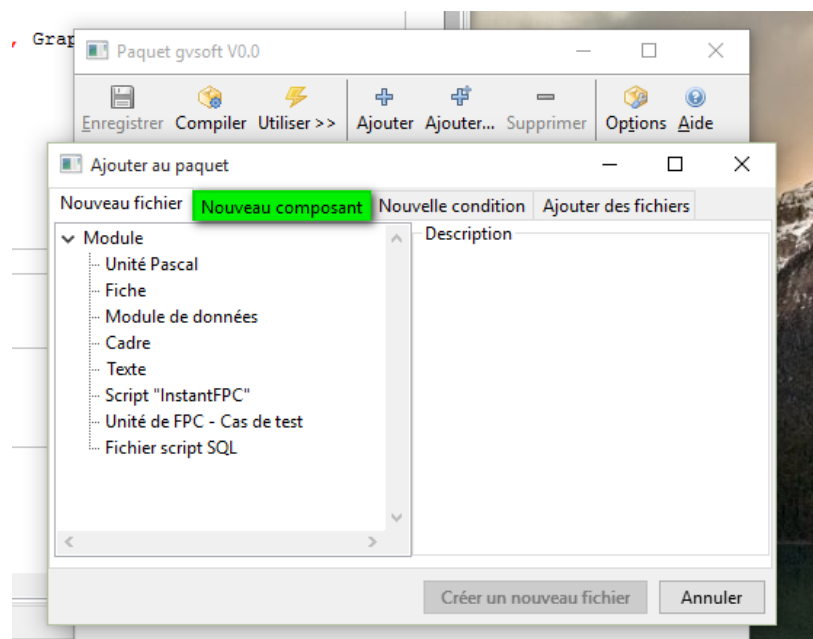
Afin d'ajouter des fichiers à ce paquet, il suffit de cliquer sur le bouton marqué « + Ajouter... » :



Le bouton situé juste à gauche de celui à cliquer et qui ne comprend que le signe + n'est utilisable qu'avec des fichiers déjà créés. Celui choisi ne se distingue que par les points de suspension qui signifient qu'un choix va être proposé parmi les différents types de fichiers susceptibles d'être créés.

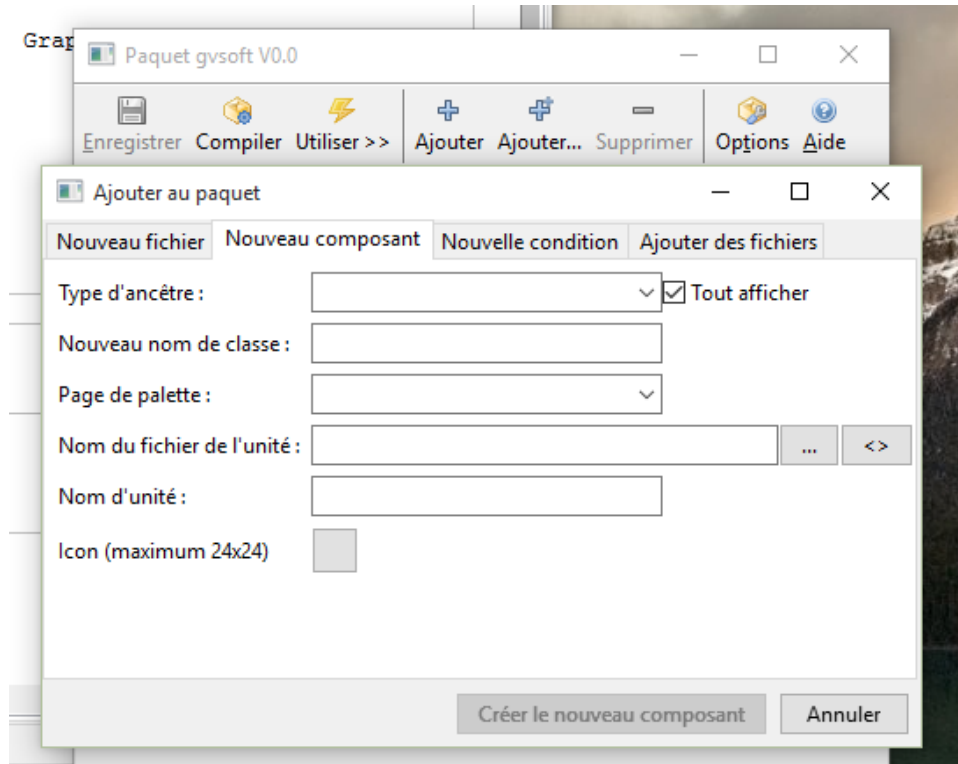
LA CREATION D'UN COMPOSANT POUR LE PAQUET : TGVURLLABEL

Dans la fenêtre qui s'affiche, comme votre intention est de créer un composant, activez l'onglet marqué « *Nouveau composant* » :

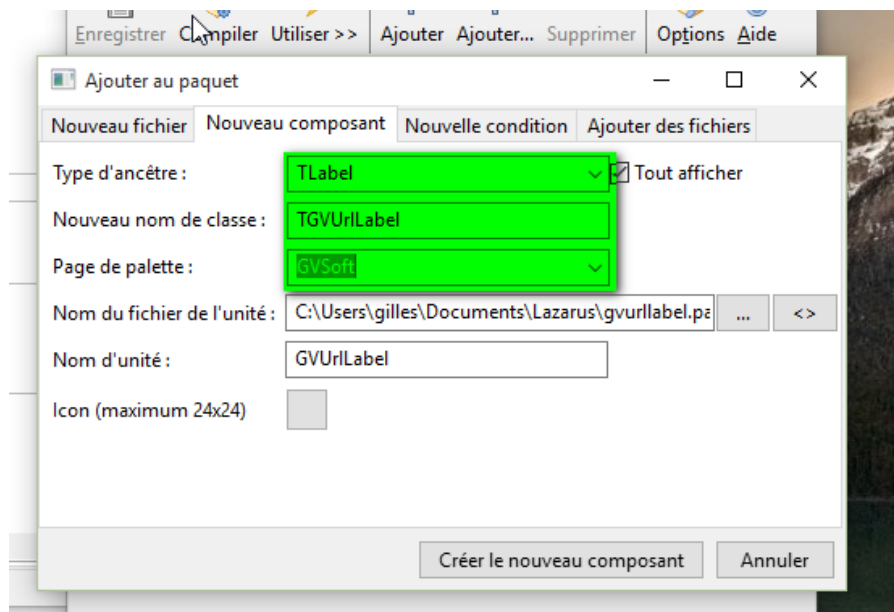


Vous remarquerez qu'il est tout à fait possible d'inclure de nouveaux fichiers dans le paquet ou encore d'ajouter des fichiers existants. L'onglet « Nouvelle condition » permet d'indiquer les conditions requises pour le paquet quant à la version minimale/maximale des autres paquets utilisés.

L'activation de l'onglet « *Nouveau composant* » produit l'affichage suivant :



Pour parfaire votre travail, remplissez les trois premières lignes d'édition (celles surlignées en vert) selon le modèle suivant :



Le « *type d'ancêtre* » détermine sur quel composant déjà enregistré s'appuiera votre nouveau composant. Vous pouvez faire défiler le nom des composants disponibles afin d'en choisir un qui limitera d'ores et déjà votre travail si vous souhaitez en récupérer les propriétés et le comportement par héritage. Au minimum, pour un comportement sans lien avec un ancêtre connu, il faudrait choisir **TComponent** qui est l'ancêtre commun à tout composant.

Le « *nom de classe* » est tout simplement le nom du composant à créer. Comme il s'agit d'un type à définir, la tradition veut que ce nom commence par la lettre majuscule T.

La « *page de palette* » est le nom de la page où figurera ce nouveau composant. Afin de le repérer facilement, vous créez une nouvelle page « GVSoft », mais vous auriez pu décider de choisir une page déjà en cours d'utilisation en faisant défiler les éléments de la zone de liste modifiable.

Les éléments restants sont générés automatiquement. À moins d'une bonne raison, il n'y a pas lieu de les modifier.


De manière optionnelle, il reste à choisir une image de type PNG dont le format doit être de 24x24 pixels. Cette image est celle qui s'affichera dans la palette de **Lazarus**. Si vous ne la précisez pas, une icône par défaut sera fournie automatiquement.

Pour changer d'icône, cliquez sur le bouton en face du mot « *icône* ». Après que vous aurez choisi l'icône adaptée, cette dernière est affichée à la place du carré vide :

Page de palette :

Nom du fichier de l'unité :

Nom d'unité :

Icon (maximum 24x24) 

Vous pouvez à présent enregistrer vos choix en cliquant sur le bouton « *Créer le nouveau composant* » :

Ajouter au paquet

Nouveau fichier Nouveau composant Nouvelle condition Ajouter des fichiers


Type d'ancêtre : ☒ Tout afficher

Nouveau nom de classe :

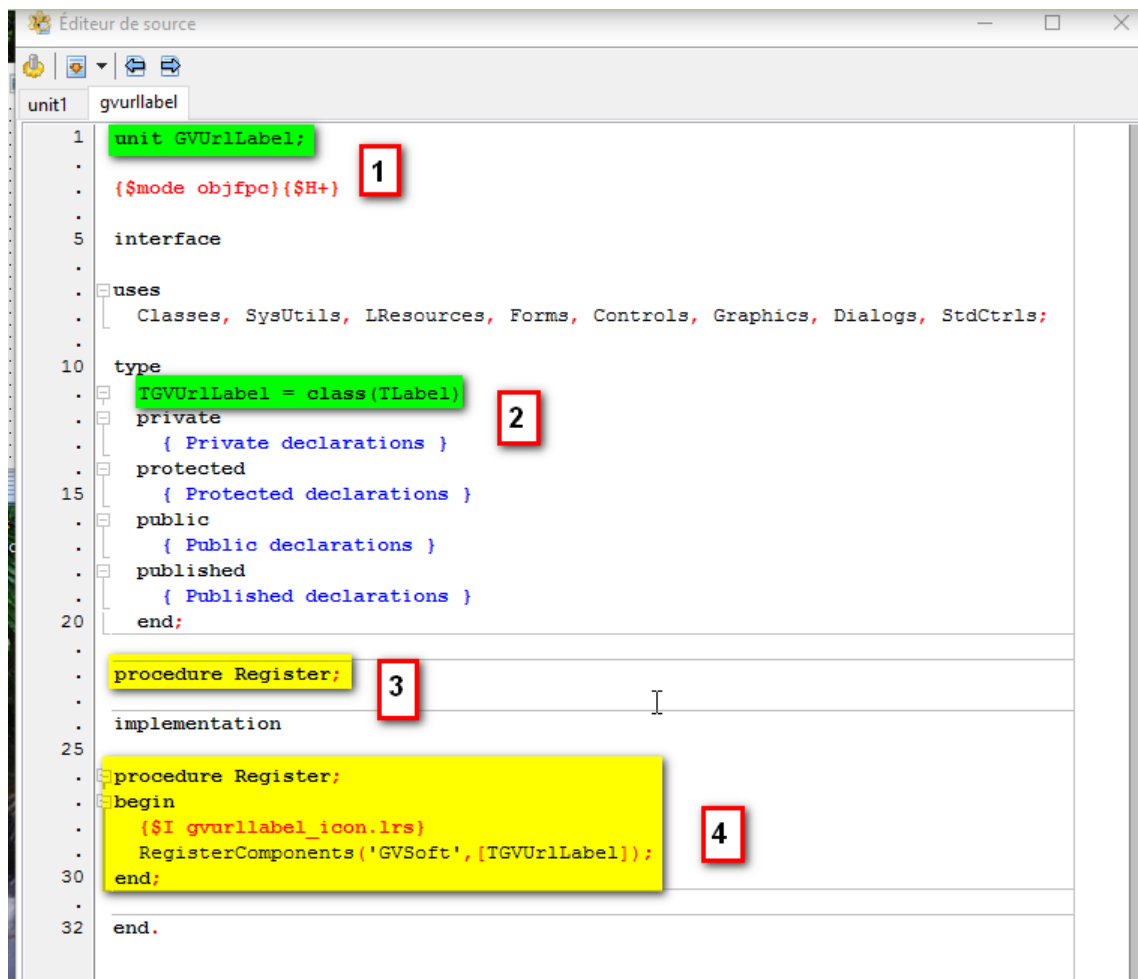
Page de palette :

Nom du fichier de l'unité : ... <>

Nom d'unité :

Icon (maximum 24x24) 

Quasi immédiatement, **Lazarus** ouvre une fenêtre de l'éditeur qui contient le squelette du nouveau composant :



L'unité porte le nom généré automatiquement [1].

Le squelette de la classe **TGVUrlLabel** apparaît bien dans la partie interface de l'unité [2]. La seule section qui n'est pas nécessaire en général, mais qui revêt toute son importance lorsqu'on crée un composant, est celle des déclarations publiées (« *published declarations* »). Il s'agit des propriétés qui figureront dans l'éditeur de propriétés et que vous pourrez modifier pendant la conception.

Plus intéressant, la procédure **Register** est déclarée [3] puis définie [4] : c'est elle qui associe l'icône choisie au composant grâce au chargement d'un fichier ressource créé lui aussi automatiquement à partir de l'icône déterminée à l'étape précédente, puis qui procède à l'enregistrement du composant dans la bonne page de la palette des composants.

Votre composant est ainsi prêt à être intégré à **Lazarus** ! Cependant, comme il ne fait que descendre du composant **TLabel**, il se comporterait tout comme lui, ce qui ne présente aucun intérêt ! À présent, votre travail va consister à apporter à ce squelette la chair dont il a besoin, c'est-à-dire les fonctionnalités que vous voulez ajouter à une simple étiquette bien connue.



Si vous êtes curieux, vous observerez que le répertoire de travail concernant ce projet contient alors, outre un sous-répertoire *backup* qui abrite les versions précédentes des fichiers, le fichier *gvsoft.lpk* qui décrit le paquet, le fichier *gvurllabel.pas* pour le code source vu ci-avant et un fichier *gvurllabel-icon.lrs* pour la ressource nécessaire à la gestion de l'icône :

Documents > Lazarus >

Nom	Modifié le	Type	Taille
backup	03/11/2015 16:51	Dossier de fichiers	
Composants	24/05/2015 11:13	Dossier de fichiers	
Forth	24/05/2015 11:16	Dossier de fichiers	
Graphisme	24/05/2015 11:12	Dossier de fichiers	
Logiciels	24/05/2015 11:13	Dossier de fichiers	
Logo	25/05/2015 10:13	Dossier de fichiers	
OpenGL	24/05/2015 11:13	Dossier de fichiers	
Sons	03/04/2015 22:07	Dossier de fichiers	
Techniques	24/05/2015 11:14	Dossier de fichiers	
TO7	24/05/2015 11:17	Dossier de fichiers	
gvsoft.lpk	03/11/2015 16:51	Lazarus Package F...	1 Ko
gvurllabel.pas	03/11/2015 17:16	Fichier PAS	1 Ko
gvurllabel_icon.lrs	03/11/2015 17:16	Fichier LRS	6 Ko

LE COMPOSANT TGVURLLABEL

Dans la partie interface de l'unité, complétez comme suit la définition de la nouvelle classe *TGVUrlLabel* :

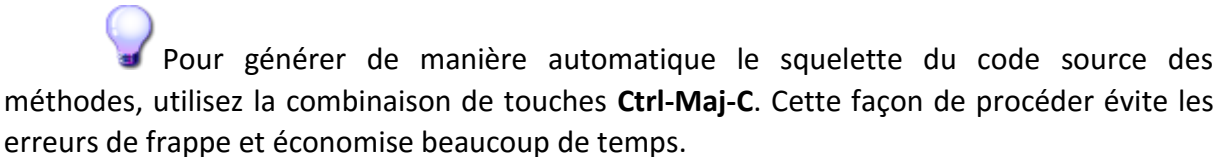
```
uses
  Classes, SysUtils, LResources, Forms, Controls, Graphics, Dialogs, StdCtrls;
type
  { TGVUrlLabel }
  TGVUrlLabel = class(TLabel)
  private
    fURLHint: Boolean; // lien dans l'aide ?
    procedure SetURLHint(AValue: Boolean); // changement de type de lien
  protected
    // entrée de la souris dans le champ du composant
    procedure MouseEnter; override;
    // sortie de la souris du champ du composant
    procedure MouseLeave; override;
    // clic sur le composant
    procedure Click; override;
  public
    // création du composant
    constructor Create(TheOwner: TComponent); override;
  published
    // URL dans l'aide contextuelle ?
```



```
property URLHint: Boolean read fURLHint write SetURLHint default True;  
end;
```

Le composant s'appuie sur le clic de la souris pour l'affichage de la page Web désirée : on redéfinit par conséquent la méthode virtuelle `Click`. L'adresse de la page Web est contenue soit dans la propriété `Hint` (bulle d'aide), soit dans la propriété `Caption` (légende), suivant la valeur de la nouvelle propriété booléenne `URLHint`. Par défaut, l'adresse est dans la propriété `Hint`.

Enfin, les méthodes virtuelles `MouseEnter` (le curseur de la souris pénètre dans la zone d'affichage du composant) et `MouseLeave` (le curseur sort de cette même zone) sont surchargées. La surcharge de ces procédures qui traitent les deux événements permet de rendre plus agréable l'affichage : si la légende contient directement l'adresse, le texte sera écrit en bleu par défaut et deviendra rouge et souligné lorsque le curseur modifié en doigt pointé le survolera. Quand ce dernier quittera cette zone, l'affichage sera rétabli dans son état initial.



Vous devriez voir apparaître les squelettes des méthodes suivantes dans la partie implémentation de l'unité :

```
procedure Register;
implementation
procedure Register;
begin
  {$I gvuurlabel_icon.lrs}
  RegisterComponents('GVSoft',[TGVUrlLabel]);
end;

{ TGVUrlLabel }

procedure TGVUrlLabel.SetURLHint(AValue: Boolean);
begin
  inherited SetURLHint(AValue);
end;

procedure TGVUrlLabel.MouseEnter;
begin
  inherited MouseEnter;
end;

procedure TGVUrlLabel.MouseLeave;
begin
  inherited MouseLeave;
end;

procedure TGVUrlLabel.Click;
begin
  inherited Click;
end;

constructor TGVUrlLabel.Create(TheOwner: TComponent);
begin
  inherited Create(TheOwner);
end;

end.
```

Comme vous allez utiliser une procédure qui concerne les URL, ajoutez sous le mot implementation la clause uses suivante :

```
implementation

uses
  LCLIntf; // pour les URL

procedure Register;
```

L'unité **LCLIntf** est à présent intégrée à votre projet : c'est elle qui contient **OpenURL**, la procédure chargée d'ouvrir une adresse dans le navigateur par défaut, et ceci quel que soit le système d'exploitation.

La méthode **SetURLHint** permet de basculer entre l'affichage *via* la propriété **Hint** et celui *via* **Caption** :

```
procedure TGVUrlLabel.SetURLHint(AValue: Boolean);
// *** changement de l'emplacement du lien ***
begin
  if fURLHint = AValue then // même valeur ?
    Exit; // on sort
  fURLHint := AValue; // nouvelle valeur affectée
  if not fURLHint then // pas dans l'aide contextuelle ?
    Font.Color := clBlue // couleur bleue pour la police
  else
    Font.Color := clDefault; // couleur par défaut pour la police
end;
```

En dehors de modifier si nécessaire la valeur booléenne du champ privé **fURLHint**, cette méthode adapte la couleur d'écriture afin qu'elle soit prise en compte immédiatement.

Quant à la méthode **MouseEnter**, elle ressemble à ceci :

```
procedure TGVUrlLabel.MouseEnter;
// *** la souris entre dans la surface de l'étiquette ***
begin
  inherited MouseEnter; // on hérite
  if not URLHint then // lien dans l'étiquette ?
  begin
    Font.Color := clRed; // couleur rouge pour la police
    Font.Style := Font.Style + [fsUnderline]; // on souligne le lien
    Cursor := crHandPoint; // le curseur est un doigt qui pointe
  end;
end;
```

Vous ne devez pas oublier d'hériter de son comportement défini par l'ancêtre **TLabel**. La seule modification apportée concerne l'affichage au cas où la légende contiendrait l'adresse Web à atteindre.

La méthode *MouseLeave* est le pendant de sa consœur *MouseEnter* :

```
procedure TGVUrlLabel.MouseLeave;
// *** la souris sort de la surface de l'étiquette ***
begin
  inherited MouseLeave; // on hérite
  if not URLHint then // lien dans l'étiquette ?
  begin
    Font.Color := clBlue; // couleur bleue pour la police
    Font.Style := Font.Style - [fsUnderline]; // on ne souligne plus le lien
    Cursor := crDefault; // le curseur est celui par défaut
  end;
end;
```

Elle hérite elle aussi de son ancêtre avant de rétablir si besoin les données concernant l'affichage.

La méthode *Click* est au cœur du nouveau travail qu'effectuera le composant *TGVUrlLabel*. Après avoir hérité son comportement de son ancêtre, elle déclenche l'affichage de la page Web indiquée en fonction de la valeur de la propriété *URLHint* :

```
procedure TGVUrlLabel.Click;
// *** clic sur l'étiquette ***
begin
  inherited Click; // on hérite
  if URLHint then // dans l'aide contextuelle ?
    OpenURL(Hint) // envoi vers le navigateur par défaut
  else
    OpenURL(Caption); // sinon envoi du texte de l'étiquette
end;
```

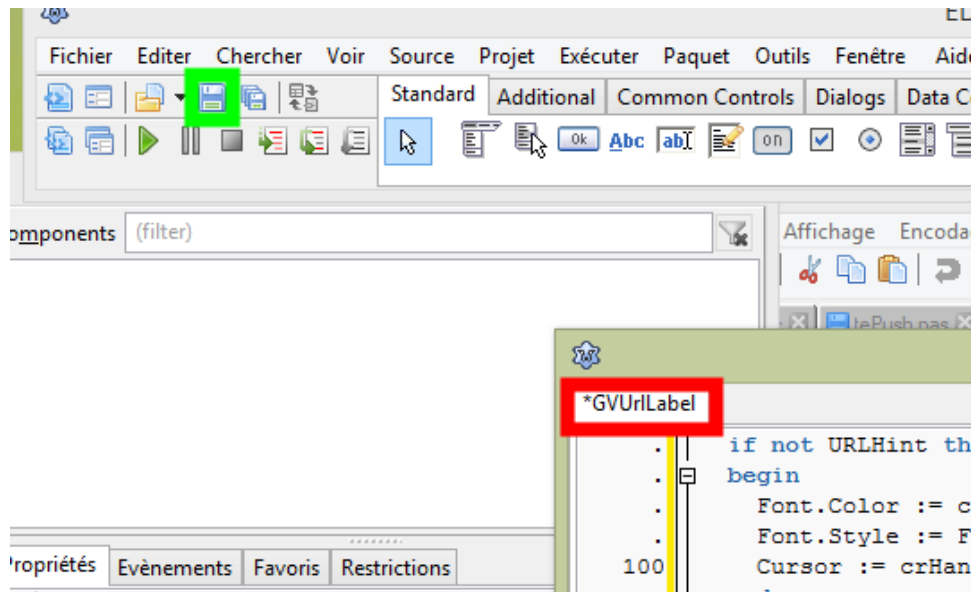
La procédure *OpenURL* est contenue dans l'unité *LCLIntf*. Elle attend en paramètre une chaîne qui contient l'adresse visée.

Enfin, il reste à compléter la méthode *Create* :

```
constructor TGVUrlLabel.Create(TheOwner: TComponent);
// *** construction du composant ***
begin
  inherited Create(TheOwner); // on hérite
  fURLHint := True; // URL dans l'aide contextuelle par défaut
end;
```

Après un héritage similaire à ceux déjà décrits, elle définit le champ *fURLHint* à *True*, conformément à la déclaration faite lors de la définition de la classe.

L'écriture du code source de votre composant est terminée ! Afin de ne pas perdre ce chef d'œuvre, pensez à l'enregistrer en cliquant sur l'icône appropriée :



L'astérisque en face du nom de l'unité (en rouge) indique que le fichier n'a pas été enregistré. Cliquez par conséquent sur l'icône représentant une disquette (en vert).



Pour les plus jeunes : la disquette, cet objet préhistorique, a bel et bien existé !

LE TEST D'UN COMPOSANT HORS INTEGRATION A L'EDI

Votre composant serait utilisable en l'état. Il suffirait d'en créer une instance comme vous le feriez d'une **TStringList**, par exemple. Simplement, comme il s'agit d'un composant visuel, il est nécessaire de lui fournir un parent afin de pouvoir l'afficher en fonction d'un contrôle donné (ici, la fiche).

Bien entendu, vous n'oublieriez pas de libérer cette instance à la fin de votre travail afin d'éviter les fuites de mémoire : la méthode **Free** serait parfaitement utilisable puisque héritée de l'ancêtre **TLabel**. Cette utilisation est d'ailleurs souvent indiquée afin de tester les composants avant de les intégrer à l'EDI.

```
unit unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs,
  gvurlabel; // l'unité qui contient le nouveau composant

type

  { TForm1 }

  TForm1 = class(TForm)
  procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
```

```

procedure FormCreate(Sender: TObject);
private
  { private declarations }
  MyUrlLabel: TGVURLLabel; // l'étiquette personnalisée
public
  { public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

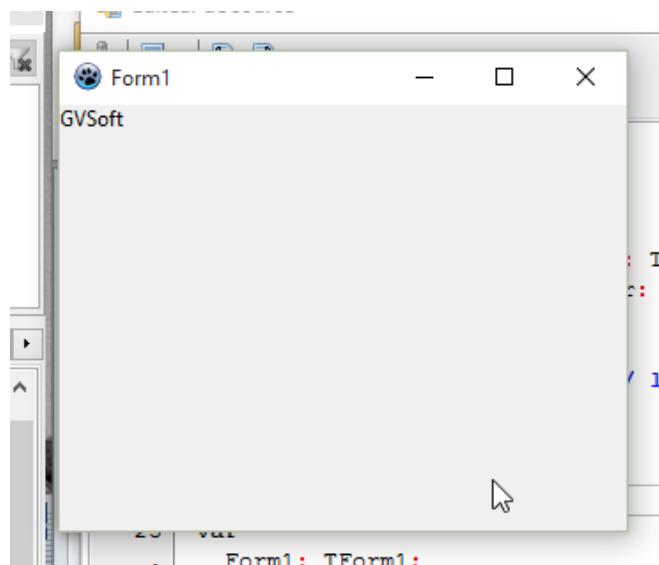
procedure TForm1.FormClose(Sender: TObject; var CloseAction: TCloseAction);
// *** l'étiquette est détruite ***
begin
  MyUrlLabel.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
// *** l'étiquette est créée ***
begin
  MyUrlLabel := TGVURLLabel.Create(Self); // le propriétaire est la fiche
  MyUrlLabel.Parent := Form1; // obligatoire pour l'affichage
  MyUrlLabel.Caption := 'GVSoft'; // le texte à afficher
end;

end.

```

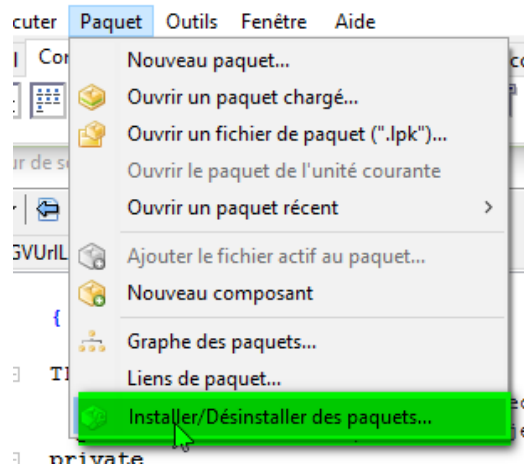
L'exécution de ce petit programme affichera « GVSoft » aux coordonnées 0, 0 de la fiche, c'est-à-dire en haut à gauche.



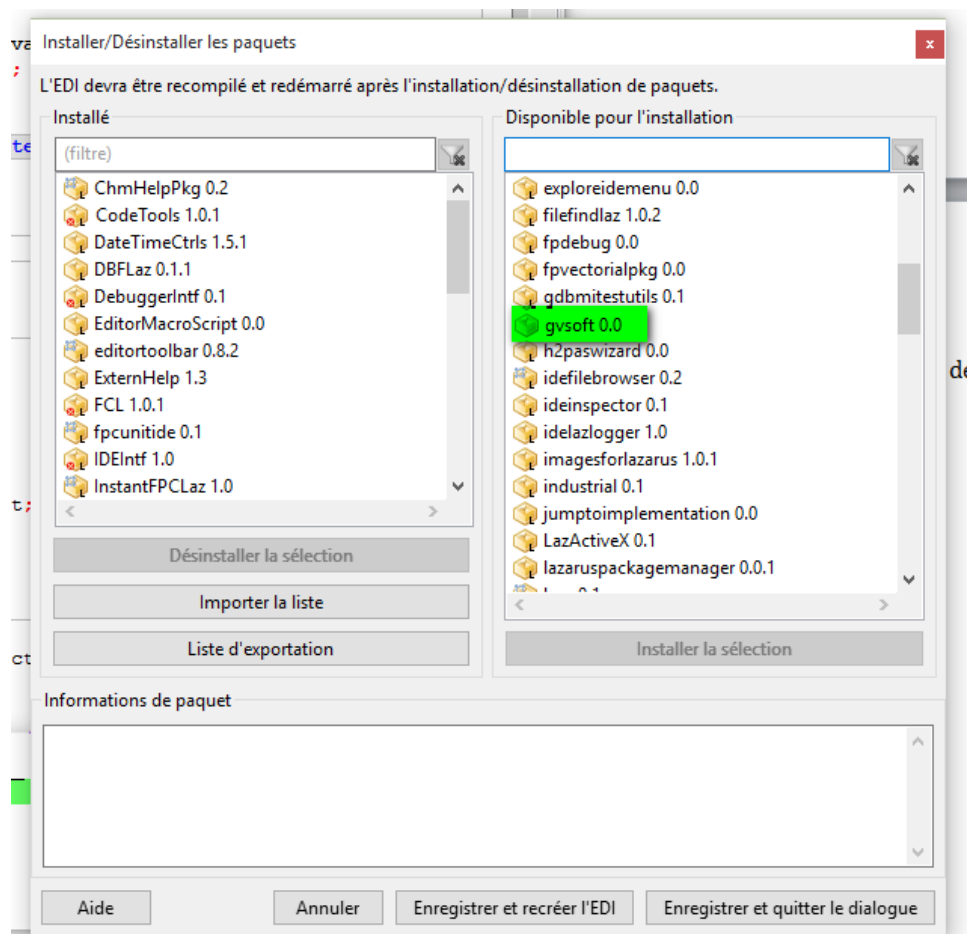
L'INSTALLATION D'UN PAQUET

Votre propos est cependant d'intégrer ce composant à la palette des composants déjà disponibles. Pour cela, il vous faut procéder à l'installation du paquet avec ses composants (en l'occurrence, le vôtre n'en contient qu'un, mais peu importe).

Cliquez sur l'option « *Installer/Désinstaller des paquets...* » du menu « *Paquet* » :

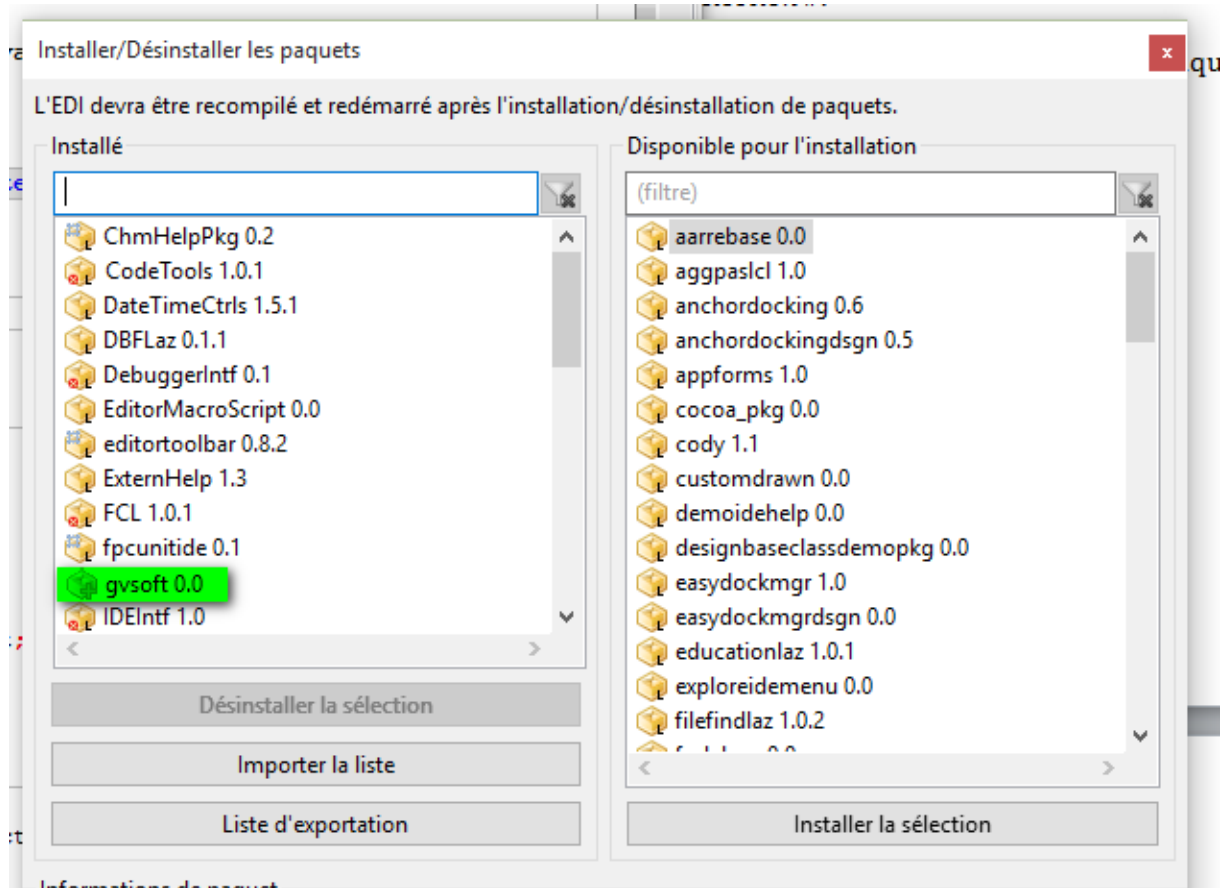


Dans la fenêtre qui s'ouvre, dans la colonne de droite, vous devriez trouver le paquet en attente d'être installé :

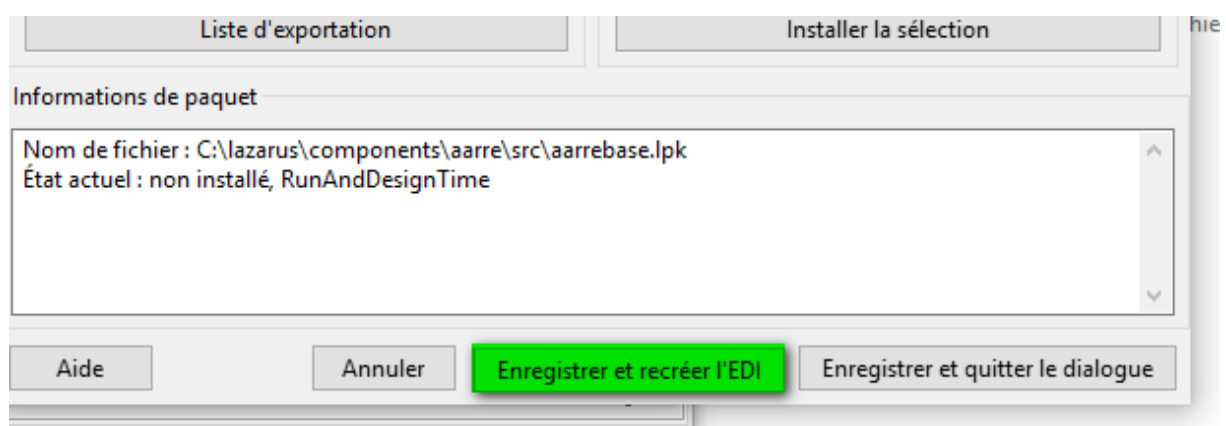


Après avoir, si besoin, fait défiler la liste des paquets en attente jusqu'à sélectionner celui qui convient, cliquez sur « *Installer la sélection* ».

Le paquet *gvsoft* est alors déplacé vers la liste de gauche, celle des paquets installés :

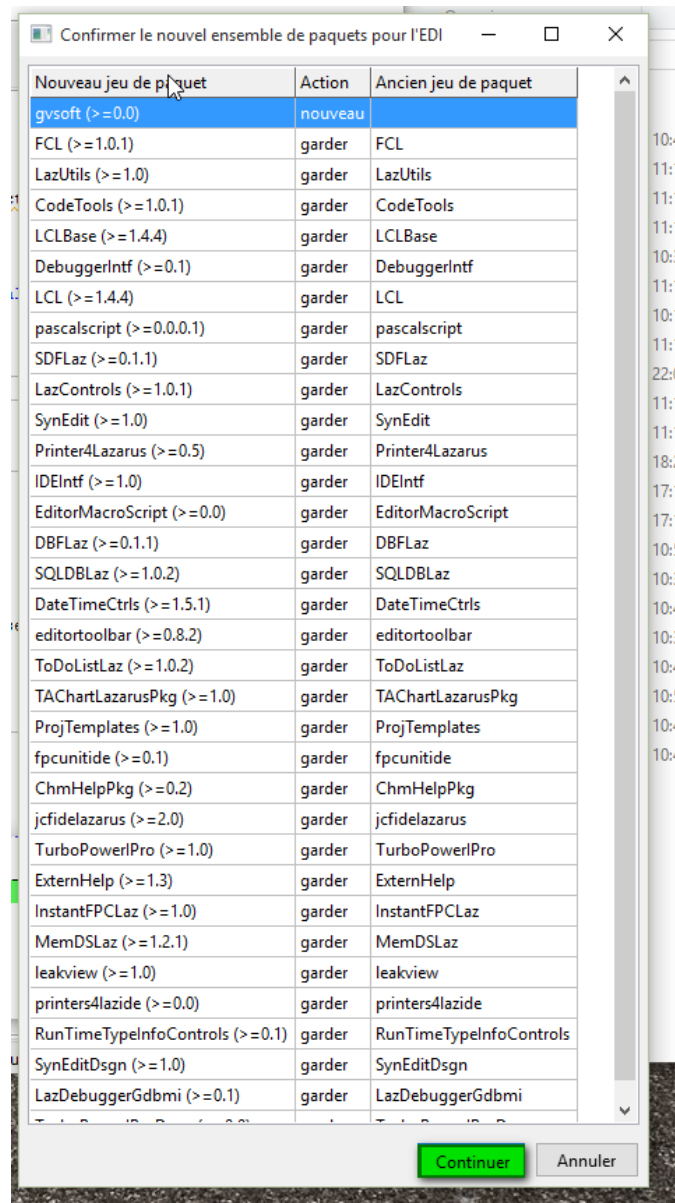


Cliquez alors sur « *Enregistrer et recréer l'EDI* ».



Contrairement à Delphi, **Lazarus** ne sait pas intégrer dynamiquement des paquets. Voilà pourquoi il est nécessaire de recompiler totalement l'EDI lui-même en cas de modifications concernant les paquets. Heureusement, cete opération ne prend que quelques secondes !

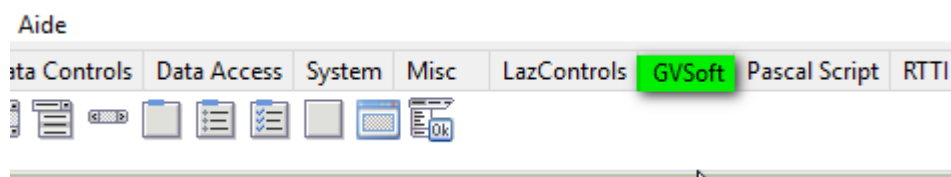
Lazarus demande ensuite confirmation des changements que vous voulez apporter :



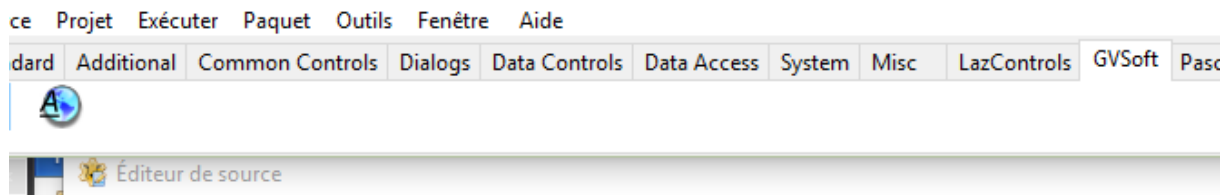
Cliquez simplement sur le bouton « *Continuer* ».

L'EDI **Lazarus** est reconstruit devant vos yeux ébahis. Après une courte disparition, l'écran de départ réapparaît, apparemment sans changement...

Une modification minime a bien eu lieu pour qui est attentif à la palette des composants :



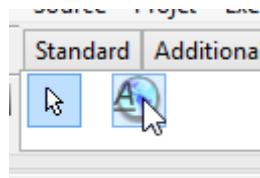
En effet, la palette comprend à présent un onglet « GVSoft ». En cliquant sur cet onglet, vous découvrirez que votre composant est intégré à l'EDI :



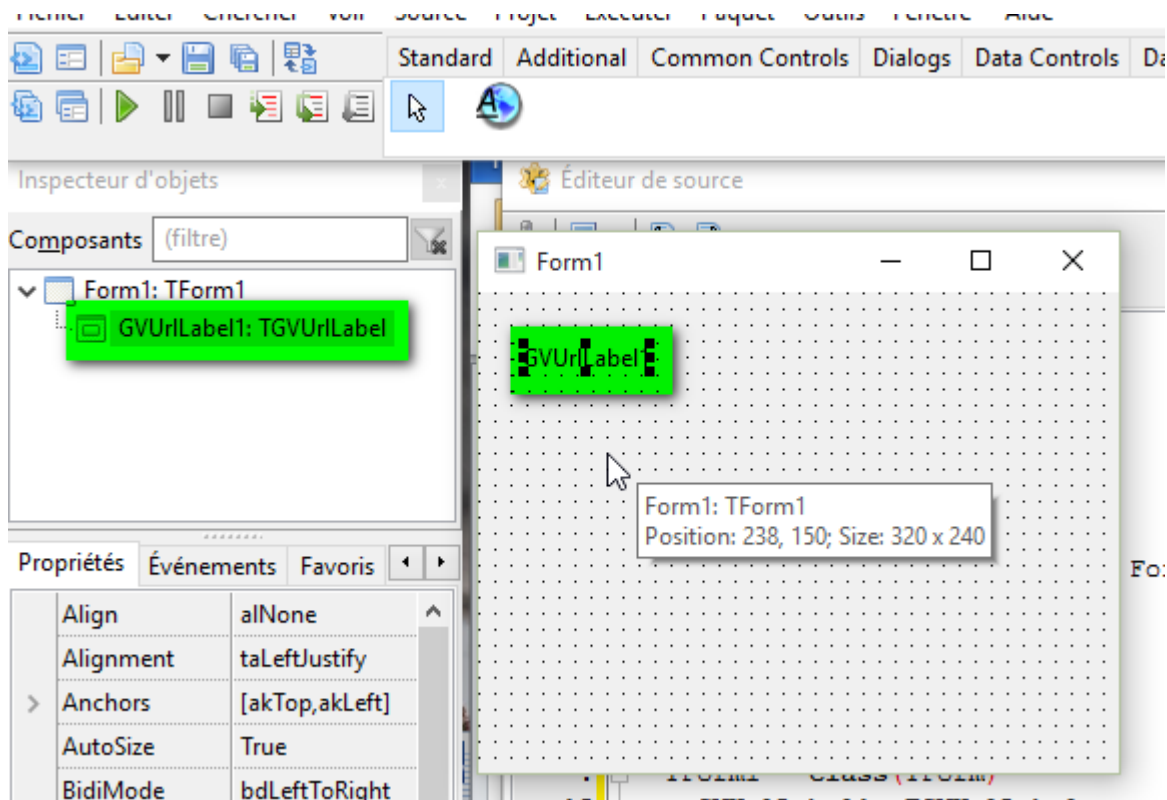
Félicitations : vous avez créé et installé un paquet comprenant un nouveau composant !

L'EXPLOITATION DU COMPOSANT CREE

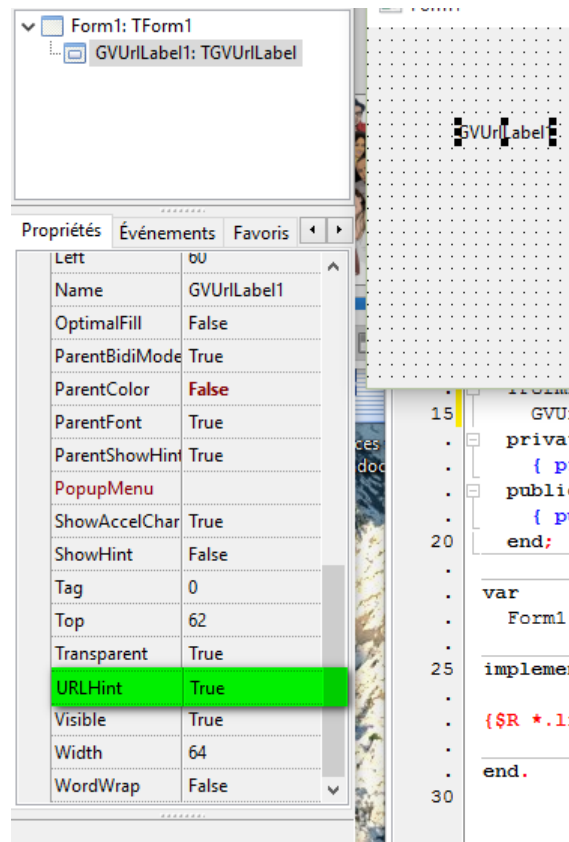
Pour tester ce composant, créez une nouvelle application et ajoutez à la fiche par défaut une instance de *TGVUrlLabel* :



Aussitôt, vous obtenez ce qui paraît être une instance de *TLabel*. Le type indiqué par l'inspecteur d'objets ne trompe néanmoins pas : il s'agit bien d'une instance de votre composant.

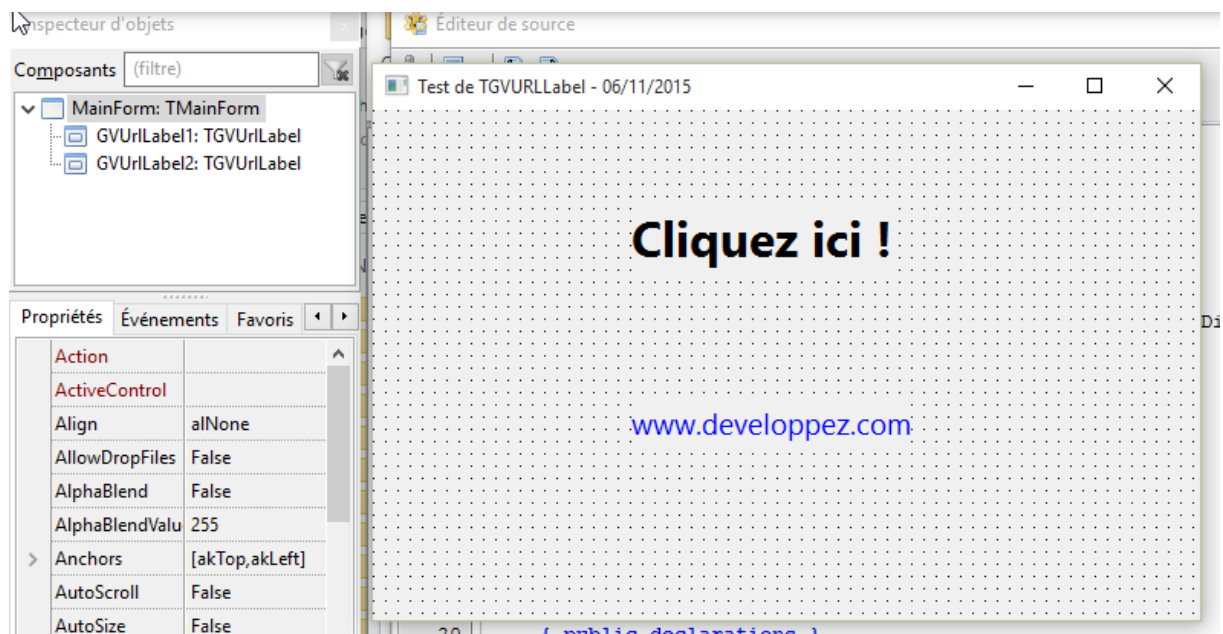


Un coup d'œil aux propriétés modifiables dans l'inspecteur d'objets montre que la propriété publiée *URLHint* est bien accessible :



[Exemple Comp-02]

Il ne reste qu'à examiner le comportement du composant suivant la valeur de cette unique propriété ajoutée. C'est ce que propose le petit programme suivant :



Il n'est nul besoin de taper le moindre code pour le faire fonctionner. Il suffit de déposer deux *TGVUrlLabel* sur la fiche et de configurer leurs propriétés ainsi :

- entrez les textes qui devront s'afficher dans chacune des étiquettes ;
- modifiez la taille des polices comme vous l'entendez ;
- complétez la propriété *Hint* de *GVUrlLabel1* avec la valeur www.developpez.com ou tout autre valeur qui convient ;
- passez la propriété *URLHint* de *GVUrlLabel2* de *True* à *False* afin d'indiquer que l'adresse à atteindre est contenue dans la propriété *Caption* du composant ;
- lancez l'exécution du programme.

En cliquant sur la première étiquette, la page documentée dans la propriété *Hint* est ouverte dans le navigateur par défaut. La seconde étiquette réagit au survol de la souris en changeant de couleur et la forme du curseur avant d'ouvrir le navigateur à la page indiquée par sa légende.

CREER DES COMPOSANTS DE QUALITE

Précédemment, vous avez créé un paquet auquel vous avez incorporé un composant. La procédure indiquée est celle que vous utiliserez si vous avez besoin de rassembler des unités, en particulier celles implémentant des composants. En revanche, le composant *TGVUrlLabel* est critiquable en l'état actuel : en effet, il fige un certain nombre de comportements alors que l'utilisateur final peut en vouloir d'autres. En particulier, pourquoi fixer la couleur de survol de la légende à rouge et celle de repos à bleu ? Le fait que ces couleurs soient celles habituellement utilisées n'exclut pas des adaptations que le concepteur initial du composant ne soupçonne même pas.

Une règle prévaut : il faut chercher à rendre le composant le plus général et le plus adaptable possible afin que ses éventuels descendants soient à même de l'ajuster à leurs besoins spécifiques. Non seulement le composant doit accomplir correctement sa tâche, mais il doit être suffisamment ouvert pour ne jamais avoir à être modifié lui-même : sinon, le gain de réemploi apporté par la programmation orientée objet disparaît.

[Exemple Comp-03]

Dans le cas de *TGVUrlLabel*, vous allez ajouter des propriétés : *Underlined* déterminera si le texte doit être souligné lors du survol, *Colored* dira s'il doit être coloré, tandis que *EnterColor* et *LeaveColor* permettront de choisir les couleurs d'entrée et de sortie de la souris.

```
type
{ TGVUrlLabel }

TGVUrlLabel = class(TLabel)
private
    fColored: Boolean;
    fEnterColor: TColor;
    fLeaveColor: TColor;
```

```

fUnderlined: Boolean;
fURLHint: Boolean; // lien dans l'aide ?
procedure SetColored(AValue: Boolean);
procedure SetEnterColor(AValue: TColor);
procedure SetLeaveColor(AValue: TColor);
procedure SetUnderlined(AValue: Boolean);
procedure SetURLHint(AValue: Boolean); // changement de type de lien
protected
    // entrée de la souris dans le champ du composant
    procedure MouseEnter; override;
    // sortie de la souris du champ du composant
    procedure MouseLeave; override;
    // clic sur le composant
    procedure Click; override;
public
    // création du composant
    constructor Create(TheOwner: TComponent); override;
published
    // URL dans l'aide contextuelle ?
    property URLHint: Boolean read fURLHint write SetURLHint default True;
    // soulignement du texte lors du survol ? ### 1.0.1
    property Underlined: Boolean read fUnderlined write SetUnderlined default True;
    // coloration lors de l'activation du texte ? ### 1.0.1
    property Colored: Boolean read fColored write SetColored default True;
    // couleur lors du survol ### 1.0.1
    property EnterColor: TColor read fEnterColor write SetEnterColor default clRed;
    // couleur lors de la fin du survol ### 1.0.1
    property LeaveColor: TColor read fLeaveColor write SetLeaveColor default clBlue;
end;

```

L'écriture du composant ne présente pas de réelles difficultés, car il suffit de prendre en compte les nouvelles propriétés au lieu des constantes d'abord utilisées :

implementation

```

uses
    LCLIntf; // pour les URL

procedure Register;
begin
    {$I gvurlabel_icon.lrs}
    RegisterComponents('GVSoft',[TGVUrlLabel]);
end;

{ TGVUrlLabel }

procedure TGVUrlLabel.SetURLHint(AValue: Boolean);
// *** changement de l'emplacement du lien ***
begin
    if fURLHint = AValue then // même valeur ?
        Exit; // on sort
    fURLHint := AValue; // nouvelle valeur affectée
end;

procedure TGVUrlLabel.SetColored(AValue: Boolean);
// *** couleur lors de l'activation ?

```

```

begin
  if fColored = AValue then // même valeur ?
    Exit; // on sort
    fColored := AValue; // nouvelle valeur
end;

procedure TGVUrlLabel.SetEnterColor(AValue: TColor);
// *** couleur lors du survol du texte
begin
  if fEnterColor = AValue then // même valeur ?
    Exit; // on sort
    fEnterColor := AValue; // nouvelle valeur de la couleur
end;

procedure TGVUrlLabel.SetLeaveColor(AValue: TColor);
// *** couleur lorsque le curseur quitte le composant
begin
  if fLeaveColor = AValue then // même couleur ?
    Exit; // on sort
    fLeaveColor := AValue; // nouvelle couleur
end;

procedure TGVUrlLabel.SetUnderlined(AValue: Boolean);
// *** soulignement lors du survol du curseur ?
begin
  if fUnderlined = AValue then // même valeur ?
    Exit; // on sort
    fUnderlined := AValue; // nouvelle valeur
end;

procedure TGVUrlLabel.MouseEnter;
// *** la souris entre dans la surface de l'étiquette ***
begin
  inherited MouseEnter; // on hérite
  if Colored then // coloration ?
    Font.Color := EnterColor; // couleur d'entrée pour la police
  if Underlined then // soulignement ?
    Font.Style := Font.Style + [fsUnderline]; // on souligne le lien
  Cursor := crHandPoint; // le curseur est un doigt qui pointe
end;

procedure TGVUrlLabel.MouseLeave;
// *** la souris sort de la surface de l'étiquette ***
begin
  inherited MouseLeave; // on hérite
  if Colored then
    Font.Color := LeaveColor; // couleur bleue pour la police
  if Underlined then // soulignement ?
    Font.Style := Font.Style - [fsUnderline]; // on ne souligne plus le lien
  Cursor := crDefault; // le curseur est celui par défaut
end;

procedure TGVUrlLabel.Click;
// *** clic sur l'étiquette ***
begin
  inherited Click; // on hérite
  if URLHint then // dans l'aide contextuelle ?

```

```

    OpenURL(Hint) // envoi vers le navigateur par défaut
else
    OpenURL(Caption); // sinon envoi du texte de l'étiquette
end;

constructor TGVUrlLabel.Create(TheOwner: TComponent);
// *** construction du composant ***
begin
    inherited Create(TheOwner); // on hérite
    fURLHint := True; // URL dans l'aide contextuelle par défaut
    fColored := True; // coloration par défaut
    fUnderlined := True; // soulignement par défaut
    fEnterColor := clRed; // couleur d'entrée par défaut
    fLeaveColor := clBlue; // couleur de sortie par défaut
end;

end.

```

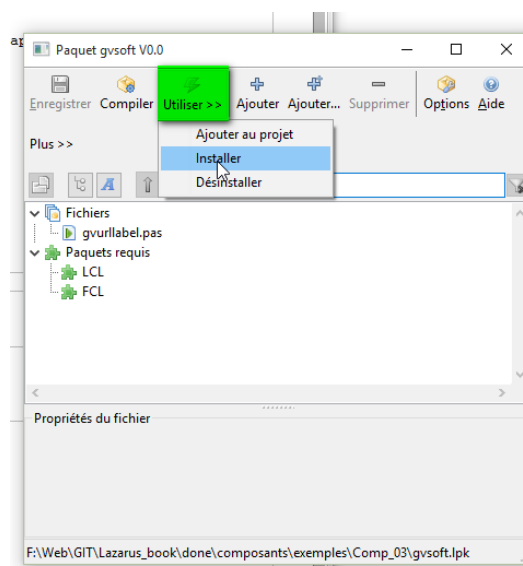


Dans la méthode *Create*, il ne faut pas oublier d'initialiser les propriétés en fonction des déclarations *default* de l'interface. En fait, *default* indique ce à quoi l'utilisateur doit s'attendre lorsqu'il utilise le composant (ce sont les valeurs des propriétés qui n'auront pas besoin d'être stockées avec la fiche), mais ne fait pas le travail d'initialisation lui-même.

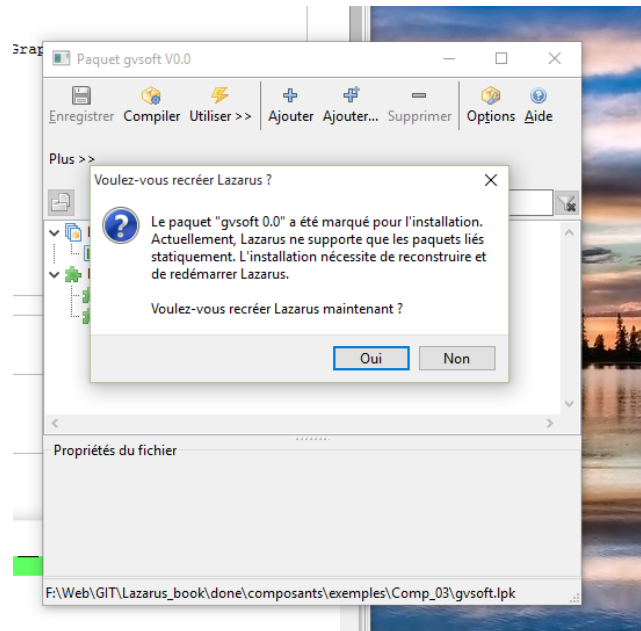
À présent, votre composant est bien plus souple : à titre d'exercice, vous pourriez l'assouplir encore plus en vous occupant du dernier élément figé, à savoir le curseur de la souris.

LA MODIFICATION D'UN PAQUET

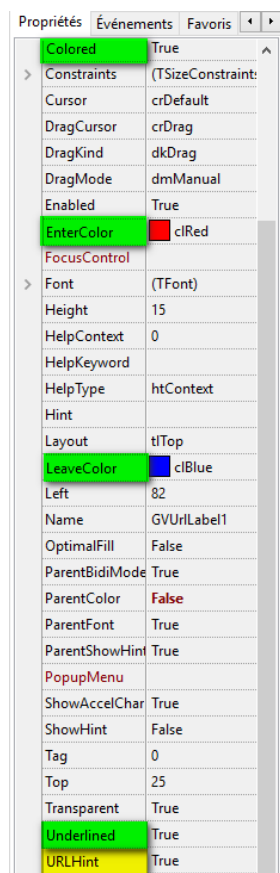
Pour intégrer dans le paquet les modifications apportées à votre composant, chargez le paquet *gvsoft.lpk* et réinstallez-le en cliquant sur « Utiliser » puis « Installer » :



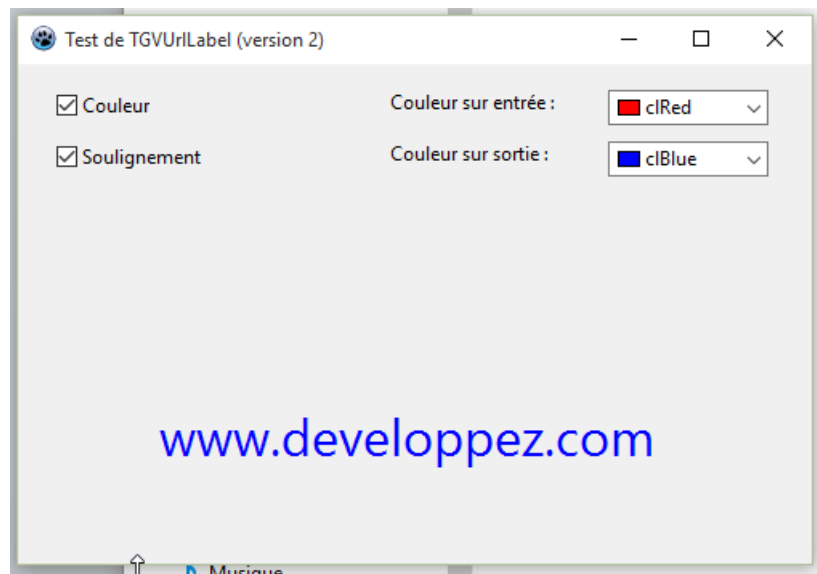
Une nouvelle fois, **Lazarus** demande confirmation de sa reconstruction :



Acceptez la reconstruction et procédez au test de votre composant modifié. Vous constaterez que les nouvelles propriétés ont bien été intégrées à l'EDI :



Voici une copie d'écran du programme de test fourni pour ce composant amélioré :



BILAN

Dans ce chapitre, vous avez appris à :

- ✓ créer un paquet ;
- ✓ créer un composant dérivé d'un composant de **Lazarus** ;
- ✓ intégrer un composant à un paquet ;
- ✓ installer un paquet ;
- ✓ modifier un paquet.