

# Installer SynEdit dans Delphi 2005 ( Win 32 )

par

Date de publication : 13 décembre 2005

Dernière mise à jour :

Installation de SynEdit V1.1 sous Delphi 2005 à titre d'exemple d'installation d'un paquet Win 32.

- I - Introduction
- II - SynEdit
- III - Installer et Compiler le paquet
  - III-A - Ouvrir le paquet
  - III-B - Modifier le fichier d'inclusion
  - III-C - Définir les chemins
  - III-D - Première compilation
  - III-E - Contourner le problème
  - III-F - Installer le paquet
- IV - Conclusion

## I - Introduction

Delphi 2005 permet aussi de compiler des applications Win 32. Je me suis donc demandé s'il était facile d'installer les paquets de composants Delphi 6 sous Delphi 2005.

Pour tester j'ai pris comme exemple la suite de composants SynEdit. Ce n'est bien sûr qu'un test mais comme ce sont des composants que j'utilise régulièrement...

## II - SynEdit

Dans cet article je suis parti de la version stable V1.1 de SynEdit, cette version est disponible sur le site de SourceForge ici : <http://synedit.sourceforge.net>

Télécharger la version 1.1 de SynEdit et décompresser les fichiers dans un répertoire \SynEdit\

### III - Installer et Compiler le paquet

#### III-A - Ouvrir le paquet

Pour installer le paquet de composants de SynEdit nous allons nous baser sur le paquet prévu pour Delphi 6 livré avec les sources.

Ouvrez le fichier \synedit\packages\SynEdit\_D6.dpk, à ce moment Delphi demande s'il faut convertir le paquet en paquet Win 32 ou .NET pour Delphi 2005.

Choisissez Win 32.


#### III-B - Modifier le fichier d'inclusion

Tous les fichiers SynEdit, appellent un fichier inclu nommé SynEdit.Inc contenant les tests de version de compilateurs. Les variables ainsi définies permettent de compiler différemment les unités en fonction des versions de Delphi ou BCB.

Nous allons donc modifier ce fichier pour tenir compte de la version 9 du compilateur de Borland.

En premier lieu il faut tester la version du compilateur afin de créer les variables de versions utilisées par la suite :

```
{$IFDEF VER170}
{$DEFINE SYN_COMPILER_9}
{$DEFINE SYN_DELPHI}
{$DEFINE SYN_DELPHI_9}
{$ENDIF}
```

 La variable prédéfinie VER170 permet de savoir que l'on utilise Delphi 2005.

Il faut ensuite utiliser la version du compilateur afin de signaler que Delphi 2005 est une évolution de Delphi 6 :

```
{$IFDEF SYN_COMPILER_9}
{$DEFINE SYN_COMPILER_1_UP}
{$DEFINE SYN_COMPILER_2_UP}
{$DEFINE SYN_COMPILER_3_UP}
{$DEFINE SYN_COMPILER_4_UP}
{$DEFINE SYN_COMPILER_5_UP}
{$DEFINE SYN_COMPILER_6_UP}
{$DEFINE SYN_COMPILER_9_UP}
{$ENDIF}

{$IFDEF SYN_DELPHI_9}
{$DEFINE SYN_DELPHI_2_UP}
{$DEFINE SYN_DELPHI_3_UP}
{$DEFINE SYN_DELPHI_4_UP}
```

```
{ $DEFINE SYN_DELPHI_5_UP }
{ $DEFINE SYN_DELPHI_6_UP }
{ $DEFINE SYN_DELPHI_9_UP }
{ $ENDIF }
```

### III-C - Définir les chemins

Afin que Delphi puisse trouver les sources et les compiler, il faut ajouter le chemin des sources dans les options du projet :

Dans le chemin de recherche, indiquez le chemin d'accès aux sources de SynEdit.

### III-D - Première compilation

Essayons de compiler...

Delphi va bloquer sur la compilation de SynEditHighLighter et indiquer des erreurs sur :

```
{ $IFDEF SYN_CPPB_1 } class { $ENDIF }
function GetCapabilities: TSynHighlighterCapabilities; virtual;
property Capabilities: TSynHighlighterCapabilities read GetCapabilities;
...
{ $IFDEF SYN_CPPB_1 } class { $ENDIF }
function GetLanguageName: string; virtual;
property LanguageName: string read GetLanguageName;
```

*"[Error] SynEditHighlighter.pas(189): E2356 Property accessor must be an instance field or method"*

Il semble donc que l'on soit tombé sur une nuance dans le langage Delphi modifié avec Delphi 2005.

En regardant de plus près on remarque que **Capabilities** est une propriété dont la méthode **Get** est une méthode de Classe. Dans Delphi 2005 ceci n'est plus possible, à moins que **Capabilities** soit une propriété de Classe. Les propriétés de classes sont une évolution du langage dans Delphi 2005.

### III-E - Contourner le problème

Les deux propriétés étant utilisées avec une méthode de classe pour méthode Get, nous allons transformer la déclaration de ces deux propriétés en propriétés de Classe.

Ainsi nous gardons la philosophie du code et nous rendons la propriété accessible à partir de la classe directement.

Le code de déclaration devient :

```
{ $IFDEF SYN_COMPILER_9_UP } class { $ENDIF }
property LanguageName: string read GetLanguageName;
{ $IFDEF SYN_COMPILER_9_UP } Class { $ENDIF }
```

```
property Capabilities: TSynHighlighterCapabilities read GetCapabilities;
```

En compilant de nouveau le paquet, delphi continue de ne pas vouloir accepter ces lignes :

*"[Error] SynEditHighlighter.pas(189): E2355 Class property accessor must be a class field or class static method"*

Eh oui... en plus il faut que la méthode Get soit statique. Ce serait une erreur de transformer les méthodes existantes en méthodes statiques car la virtualité est utilisée par les composants descendants. Nous allons donc créer deux méthodes de classe statiques, afin de pouvoir déclarer les propriétés statiques. Ces deux nouvelles méthodes vont appeler les anciennes méthodes virtuelles de classe. Ainsi indirectement les méthodes Get des propriétés appelleront le code surchargé par les descendant.

La déclaration devient donc :

```
{$IFDEF SYN_COMPILER_9_UP} class {$ENDIF}
property LanguageName: string read GetLanguageNameC;
{$IFDEF SYN_COMPILER_9_UP} Class {$ENDIF}
property Capabilities: TSynHighlighterCapabilities read GetCapabilitiesC;
```

La définition des deux nouvelles méthodes est la suivante :

```
public
{$IFNDEF SYN_CPPB_1} class {$ENDIF}
function GetCapabilities: TSynHighlighterCapabilities; virtual;
{$IFNDEF SYN_CPPB_1} class {$ENDIF}
function GetLanguageName: string; virtual;
{$IFDEF SYN_COMPILER_9_UP}
class function GetCapabilitiesC: TSynHighlighterCapabilities;static;
class function GetLanguageNameC: string;static;
{$ENDIF}
```

L'implémentation devient :

```
{$IFNDEF SYN_CPPB_1} class {$ENDIF}
function TSynCustomHighlighter.GetCapabilities: TSynHighlighterCapabilities;
begin
Result := [hcRegistry]; //registry save/load supported by default
end;

{$IFDEF SYN_COMPILER_9_UP}
class function TSynCustomHighlighter.GetCapabilitiesC: TSynHighlighterCapabilities;
begin
Result := GetCapabilities;
end;
{$ENDIF}

//....

{$IFNDEF SYN_CPPB_1} class {$ENDIF}
function TSynCustomHighlighter.GetLanguageName: string;
begin
{$IFDEF SYN_DEVELOPMENT_CHECKS}
raise Exception.CreateFmt('%s.GetLanguageName not implemented', [ClassName]);
```

```
{$ENDIF}
  Result := '<Unknown>';
end;

{$IFDEF SYN_COMPILER_9_UP}
class function TSynCustomHighlighter.GetLanguageNameC: string;
begin
  Result := GetLanguageName;
end;
{$ENDIF}
```

Le paquet se compile correctement maintenant.

### III-F - Installer le paquet

Pour installer le paquet, utilisez un clic droit sur le projet dans l'explorateur de projet sur la droite :

Delphi devrait signaler l'installation des nouveaux composants :

## IV - Conclusion

Et voilà les composants sont installés, vous pouvez les essayer directement en plaçant un TSynEdit sur une fiche ainsi qu'un composant de mise en évidence de la syntaxe.

L'installation n'est donc pas différente des habitudes prises avec Delphi 6/7. Seules les nouveautés du langage peuvent nous réserver quelques surprises, mais rien de bien long à modifier.

Merci à [Oliver Lance](#) et [Pierre Castelain](#) pour la correction orthographique.