Build automatique avec Make

par Laurent Dardenne>

Date de publication : 08/11/2006

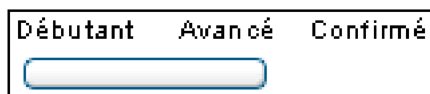
Dernière mise à jour : 18/11/2006

Delphi est livré avec un outil très intéressant et pouvant être d'une aide appréciable dans la gestion de projet de développement logiciel, à savoir Make.exe. Son objectif est de construire une unité, un package, une librairie, un ou plusieurs projets automatiquement. Cet automatisme facilite la livraison d'un exécutable à une cellule de test, ou d'une librairie de composants destinée à une équipe de développement, et bien évidemment la livraison du produit finalisé au client. Je tiens à remercier Yann Le Quéré, Hélène Auerbach, tiki06 et SJRD pour leurs corrections orthographiques et relectures attentives.

1 - Public concerné
2 - Les sources
3 - Principe de Make
4 - Un fichier Make basique
5 - Dépendance des cibles
6 - Suppression des fichiers intermédiaires
7 - De l'usage de règles implicites
8 - Une compilation sans dépendances
9 - Gestion des fichiers ressources
10 - Prise en compte des options de compilation
11 - Compilation de multiples projets
    11-1 - Les fichiers Delphi de groupe de projets
    11-2 - Exemple concret
    11-3 - Fichier .obj et Delphi
12 - Concepts et termes utilisés
    12-1 - Cible
    12-2 - Cible symbolique
    12-3 - Dépendances
    12-4 - Commande
    12-5 - Règle
    12-6 - Règle explicite
    12-7 - Règle implicite
    12-8 - Les directives conditionnelles
    12-9 - Les macros ou variables de MAKE
13 - Liens

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

1 - Public concerné



Testé sous XP sp2 avec Make.exe de BDS 2006.

Version 1.0

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

2 - Les sources

Les fichiers sources des différents exemples :

FTP.

HTTP.

L'article au format PDF.

Il est regrettable de ne pas retrouver dans l'aide en ligne des versions BDS 2005-2006 la documentation de Make, il est en revanche possible de la retrouver dans celle de Delphi 7 ou sur le FTP Borland.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

3 - Principe de Make

Son principe est simple, gérer des dépendances de fichiers nécessaires à la construction d'un fichier final. Un fichier Make n'est rien d'autre qu'un fichier ASCII contenant un ensemble normé d'instructions nécessaires à cette construction.

Nous aborderons ici les principes de base du build automatique sous son aspect technique. Vous trouverez dans ce tutoriel les principes de base de l'intégration automatique, ici sous son aspect technique et organisationnel.

Notez que ce document contient de nombreux extraits de l'aide en ligne de Delphi 7 concernant Make.exe (cf. *DocumentationBorland.zip*).

> *Make peut être utilisé pour d'autres tâches que la compilation de codes sources.*
>
> *Vous trouverez dans le chapitre "Concepts et termes utilisés" des informations complémentaires sur certains termes spécifiques utilisés dans ce tutoriel.*

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

4 - Un fichier Make basique

Il est possible d'utiliser directement le compilateur en ligne de commande dans un batch mais son association avec Make.exe renforcera son efficacité. Certaines des options de DCC32.exe peuvent modifier le déroulement du fichier make, par exemple l'option ***-m*** force la reconstruction des unités.

Regardons de plus près un exemple (*Demo-1.mak*) contenant des directives et des macros :

```
!message Les options en ligne de commande de Make sont : $(MAKEFLAGS)
# teste si la macro ROOT n'est pas définie et si oui la définit avec le répertoire où
# se trouve l'exécutable MAKE
!ifndef ROOT
ROOT=$(MAKEDIR)
!endif

fichiers= FMain.pas
# Affiche un message et le contenu d'une macro
!message Le répertoire de make est : $(ROOT)

# Définit le chemin complet du compilateur en ligne de commande
DCC = $(ROOT)\dcc32.exe -q -w

!message La ligne de commande pour la compilation avec DCC est : $(DCC)

# Supprime une macro
!undef ROOT

!message Après suppression de la macro ROOT,
!message la ligne de commande pour la compilation avec DCC est : $(DCC)

!message AVANT. La macro fichier = $(fichiers)

!message Substitution provisoire de la macro fichiers = $(fichiers:.pas=.dcu)

!message APRES. La macro fichiers = $(fichiers)

!ifdef FICHIERS
!message La macro FICHIERS existe
!else
!message ** La macro FICHIERS n'existe pas **
!endif

!message Variables d'environnement du shell = $(COMSPEC) $(USERNAME) ...

# Affiche une erreur et stoppe l'opération de construction
!error La commande !error stoppe l'opération
```

Son exécution en ligne de commande se fait de la manière suivante, on suppose l'installation de Delphi complète et le path à jour :

```
Make -fDemo-1
```

Voici le résultat de son exécution :

```
C:\...>make -fDemo-1
MAKE Version 5.2  Copyright (c) 1987, 2000 Borland
Les options en ligne de commande de Make sont : l -o

Le répertoire de make est : C:\PROGRA~1\Borland\BDS\4.0\Bin

La ligne de commande pour la compilation avec DCC est :
C:\PROGRA~1\Borland\BDS\4.0\Bin\dcc32.exe -q -w -m

Après suppression de la macro ROOT, la ligne de commande pour la compilation avec DCC est :
 \dcc32.exe -q -w -m
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

```
AVANT. La macro fichier = FMain.pas

Substitution provisoire de la macro fichiers = FMain.dcu

APRES. La macro fichiers = FMain.pas

** La macro FICHIERS n'existe pas **

Variables d'environnement du shell = C:\WINDOWS\system32\cmd.exe Laurent ...

Fatal Demo-1.mak 36: Error directive: La commande !error stoppe l'opération

C:\...>
```

Après son exécution notez que :

- Pour la macro **$(DCC)**, l'imbrication du nom de macro **ROOT**, ne recopie pas le contenu de cette dernière mais uniquement son nom. On peut donc dire que la macro DCC référence la macro ROOT,
- les noms de macro sont sensibles à la casse,
- les commentaires débutent par un signe dièse (**#**),
- les variables d'environnement du shell sont toutes accessibles,
- la substitution d'une macro peut être provisoire,
- certains paramètres sont activés sans qu'il soit nécessaire de les spécifier sur la ligne de commande. Make.exe peut les modifier via le paramètre ***-Wfilename***.

Pour obtenir l'aide en ligne de MAKE, saisir : ***Make /?***

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

5 - Dépendance des cibles

Pour les exemples suivants nous utiliserons un fichier projet Delphi (UnProjet.dpr), une fiche (FMain.pas et FMain.dfm) et une unité (UGlobal.pas).

Le fichier make suivant (*Demo-2.mak*) énumère les cibles nécessaires à la compilation du projet :

```
# Déclare une cible dépendante des cibles FMain.dcu et UnProjet.dpr
# Les cibles sont ordonnées
UnProjet.exe : FMain.dcu UnProjet.dpr
                $(DCC) UnProjet.dpr

# Déclare une cible dépendante du fichier FMain.pas
FMain.dcu : FMain.pas
                $(DCC) FMain.pas
```

Il est possible de simuler l'exécution de Make en utilisant le paramètre **-n** qui peut être couplé au paramètre **-p**.

- Le premier (**-n**) affiche les commandes sans les exécuter,
- le second (**-p**) affiche toutes les définitions de macros et toutes les règles implicites.

Ce qui dans notre exemple nous donnera :

```
C:...>make -fdemo-2 -n -p
MAKE Version 5.2  Copyright (c) 1987, 2000 Borland

Macros:
#
    SESSIONNAME = Console
Implicit Rules:

Targets:
    UnProjet.exe:
        flags:
        dependents:  FMain.dcu UnProjet.dpr
        commands:    $(DCC) UnProjet.dpr
    FMain.dcu:
        flags:
        dependents:  FMain.pas
        commands:    $(DCC) FMain.pas

        C:\PROGRA~1\Borland\BDS\4.0\Bin\dcc32.exe -w FMain.pas
        C:\PROGRA~1\Borland\BDS\4.0\Bin\dcc32.exe -w UnProjet.dpr
```

On peut voir que les cibles ne sont pas ordonnées dans le fichier .mak. En revanche, les dépendances le sont et suivent celle de la déclaration de la cible *UnProjet.exe*, ce qui est confirmé lors de la compilation.

Si on ne précise pas de cible sur la ligne de commande c'est la première cible du fichier .mak qui sert de point d'entrée à la construction.

Make effectue donc 2 passes, la première pour rechercher les dépendances des cibles et la seconde pour construire les cibles.

Voici le rôle de Make,

" # Après le chargement du makefile, MAKE essaie de construire seulement la première cible explicite listée dans

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

le makefile en vérifiant la date et l'heure des fichiers dépendants de cette cible. Si les fichiers dépendants sont plus récents que le fichier cible, MAKE exécute les commandes pour mettre à jour la cible. Si l'un des fichiers dépendants de la première cible est utilisé comme cible ailleurs dans le makefile, MAKE vérifie les dépendances de cette cible et la construit avant de construire la première cible. Cette réaction en chaîne est appelée dépendance de liens. #"

La dépendance des liens est traitée par les règles explicites qui spécifient les instructions que MAKE doit respecter lors de la construction de cibles spécifiques.

Avec ce fichier make tous les fichiers dépendants du projet ne sont pas gérés lors de la recherche de la dépendance des liens.

Par exemple en cas de modification de l'unité *UGlobal.pas* sa recompilation ne sera pas prise en compte.

Dans l'exemple suivant (*demo-2-1.mak*) l'unité *UGlobal.pas* se trouve désormais référencée :

```
# Déclare une cible dépendante des cible FMain.dcu, UGlobal.dcu et UnProjet.dpr
# Les cibles sont ordonnées
UnProjet.exe : UGlobal.dcu FMain.dcu UnProjet.dpr
                $(DCC) UnProjet.dpr

# Déclare une cible dépendante du fichier UGlobal.pas
UGlobal.dcu :          UGlobal.pas
                $(DCC) UGlobal.pas

# Déclare une cible dépendante du fichier FMain.pas
FMain.dcu : FMain.pas
                $(DCC) FMain.pas
```

Make reconstruira l'unité *UGlobal.pas* puis la fiche *FMain.pas* et enfin le projet *UnProjet.dpr*.

Il reste un problème : si on modifie uniquement la fiche (.dfm), par exemple la propriété **TAG**, cela n'impactera pas l'unité associée (.pas) et la recompilation du projet ne se fera pas.

Modifions donc en conséquence le fichier .mak (*demo-2-2.mak*):

```
UnProjet.exe : UGlobal.dcu FMain.dcu UnProjet.dpr
                $(DCC) UnProjet.dpr

# Déclare une cible dépendante du fichier UGlobal.pas
UGlobal.dcu :          UGlobal.pas
                $(DCC) UGlobal.pas

# Déclare une cible dépendante du fichier FMain.pas et du fichier FMain.dfm
FMain.dcu : FMain.pas FMain.dfm
                $(DCC) FMain.pas
```

La cible **FMain.dcu** dépend désormais des fichiers .pas et .dfm.

*Les fichiers inclus, via la directive **{$I fichier.ext}**, sont également à citer dans les dépendances.*

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

6 - Suppression des fichiers intermédiaires

Il est parfois souhaitable de reconstruire entièrement un projet, à cette fin on peut utiliser une cible dédiée à la suppression des fichiers intermédiaires, un fichier intermédiaire est la cible d'une dépendance, qui forcera la recompilation.

Ajoutons une cible nommée *clean* prenant en charge cette opération (*Demo-2-3.mak*) :

```
clean:
#Multiple exécution possible
        @if exist *.dcu del *.dcu
        -@if exist UnProjet.exe del UnProjet.exe
```

Le @ indique à Make de ne pas afficher la commande, le caractère - placé en début de commande indique à Make de continuer son déroulement même s'il rencontre une erreur lors de l'exécution de la commande concernée.

Exécutons le fichier en précisant sur la ligne de commande le nom de la cible :

```
Make -fDemo-2-3.mak clean
```

Cette cible n'est pas prise en compte automatiquement par Make car elle ne référence aucun fichier à construire et n'est référencée par aucune autre cible, il s'agit d'une cible "bidon". Sa prise en compte pourrait se faire de la manière suivante :

```
UnProjet.exe : clean FMain
```

Dans ce cas à chaque exécution on supprimera tous les fichiers intermédiaires.

Make propose une cible particulière nommée .PHONY dont les dépendances seront systématiquement reconstruites, elle est préférable à la construction précédente (*UnProjet.exe : clean FMain*).

> *Ici le nom de la cible clean est plus une convention qu'une obligation, vous pouvez lui donner le nom que vous voulez...*

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

7 - De l'usage de règles implicites

Ces règles permettent de rechercher automatiquement une opération associée à une expression, sans qu'elles soient indiquées explicitement dans le fichier .mak, tout en évitant une multiplication de cibles redondantes.

L'exemple *Demo-3.mak* utilise ce type de règle :

```
UnProjet.exe : UGlobal.dcu FMain.dcu UnProjet.dpr
                $(DCC) UnProjet.dpr

# Règle implicite ou règle d'inférence. Ne contient pas de dépendances mais juste une
# commande
# Les cibles .dcu ne sont plus explicitées
.pas.dcu :
              $(DCC) $<

#Cible "bidon"
clean:
       ...
```

La première extension (.pas) appartient au dépendant, la seconde (.dcu) à la cible.

La macro interne **$<** renvoi le nom de la dépendance, c'est à dire qu'à un fichier **f1.dcu** correspond un fichier **f1.pas**

> *Consultez les fichiers joints pour plus d'informations sur la syntaxe de ces règles.*

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

8 - Une compilation sans dépendances

Le fichier *Demo-4.mak* contient deux macros et une règle explicite réduite à sa plus simple expression :

```
DCC=$(ROOT)\dcc32.exe -w

# Déclare une seule cible, les dépendances sont gérées par DCC32.exe
UnProjet.exe:
        # Compile le projet nommé UnProjet.dpr
        $(DCC) UnProjet.dpr
```

La commande ne fait que compiler le fichier projet du même nom, avec les options déclarées dans la variable **$(DCC)**. Les dépendances ne sont donc pas ici énumérées.

Il est possible d'utiliser des macros par défaut (*Demo-4-1.mak*) :

```
# Déclare une cible, c'est à dire un fichier résultant d'une compilation
UnProjet.exe:
        # Compile le projet nommé UnProjet en utilisant la macro $&
        $(DCC) $&.dpr
```

La macro **$&** substitue explicitement le nom de la cible sans son extension, c'est à dire par "UnProjet".

Consultez les fichiers joints pour plus d'informations sur la syntaxe de ces règles.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

9 - Gestion des fichiers ressources

Un projet Delphi est souvent associé à un ou plusieurs fichiers ressources, Make peut ici aussi nous aider à gérer les fichiers dépendants.

Le fichier suivant (*Demo-res.mak*) génère les ressources du projet :

```
# Répertoire des fichiers ressources (.rc et .res)
RSRC = .\ressources

# Compilateur de ressources
BRC = "$(ROOT)\brc32.exe" -r

# Macro interne
# Répertoire de recherche des fichiers portant l'extension .rc
.path.rc = $(RSRC)

# Macro interne
# Répertoire de recherche des fichiers portant l'extension .res
.path.res = $(RSRC)

# Règle implicite pour les fichiers ressources
.rc.res:
    # La macro $& affiche le fichier dépendant
  @echo [Compile le fichier ressource : $&]
    # Se place dans le répertoire hébergeant les fichiers ressources
  @cd $(RSRC)
  @$(BRC) -fo$&.res $&.rc

# Une seule dépendance
default: DeveloppezIcon.res
```

Cet exemple utilise la macro interne **path.ext** qui permet de retrouver les fichiers situés dans un autre répertoire que celui du fichier .mak.

La règle implicite **.rc.res** utilise la macro **$&** qui permet de manipuler le nom, sans extension, du fichier dépendant cité dans la cible symbolique **default:**.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

10 - Prise en compte des options de compilation

Il est possible de gérer les options de compilation, utilisées en paramètre par **Dcc32.exe**, directement dans un fichier make bien que leur présence dans le code source ou dans un fichier d'inclusion est préférable car dans ce cas il n'existe qu'une seule source d'informations.

La présence de directives de compilation conditionnelle dans le code source peut également vous aider à générer différentes versions : test, debug ou encore finale.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

11 - Compilation de multiples projets

L'usage d'une cible symbolique permet de construire plusieurs projets au sein d'un seul fichier Make.

Le fichier suivant (*demo-multiprojet.mak*)

```
# Déclare une cible symbolique
default: \
ProjetConsole.exe \
unProjet.exe \

# Règle implicite
.dpr.exe :
        $(DCC) $&.dpr
```

La présence dans la cible *default:* du caractère **** permet d'énumérer les dépendances sur plusieurs lignes.

Ici la suppression des fichiers *.dcu* n'impliquera pas la recompilation, seule la suppression des *.exe* ou la modification des *.dpr* forcera la recompilation.

11-1 - Les fichiers Delphi de groupe de projets

La JVCL propose quelques outils de gestion des fichiers projet de Delphi 7, voir les utilitaires bpg2make.exe et make2dof.exe.

Sous BDS les fichiers projet sont au format XML, il est donc aisé de les convertir en fichier Make.

11-2 - Exemple concret

Faute de temps je ne vous proposerai malheureusement pas d'exemple concret sur l'utilisation de Make, vous pouvez en revanche consulter les fichiers Make utilisés dans la JVCL qui sont au coeur de l'installation de cette librairie multi-version.

11-3 - Fichier .obj et Delphi

Vous trouverez dans cet article de Rudy Velthuis, de la TeamB, quelques informations à propos de l'usage de fichiers **.obj**, issus du langage C, sous Delphi.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.

12 - Concepts et termes utilisés

Les verbes anglais *to make* (faire, fabriquer) et *to build* (construire, bâtir) signifient, dans notre contexte, la même chose. Alors que l'expression *build in* implique une opération supplémentaire après la construction, i.e. l'intégration.

Vous trouverez ci-dessous les concepts liés à la construction automatique de fichiers dépendants.

12-1 - Cible

"*Spécifie le nom et l'extension du fichier à construire (une cible doit commencer une ligne dans le makefile - vous ne pouvez pas faire précéder le nom de la cible par des espaces ni par des tabulations). Pour spécifier plusieurs cibles, séparez les noms des cibles par des espaces ou des tabulations. Egalement, vous ne pouvez pas utiliser un nom de cible plusieurs fois à l'emplacement d'une cible dans une règle explicite.*"

12-2 - Cible symbolique

Une cible symbolique force MAKE à construire plusieurs cibles d'un makefile. Lorsque vous spécifiez une cible symbolique, la ligne des dépendances affiche toutes les cibles que vous souhaitez construire (une cible symbolique utilise les dépendances liées pour construire plus d'une cible).

Une cible symbolique n'a pas de commande associée.

12-3 - Dépendances

"*Est le ou les fichiers dont MAKE vérifie la date et l'heure pour savoir s'ils sont plus récents que la cible. Chaque fichier dépendant doit être précédé d'un espace. Si un fichier dépendant apparaît ailleurs dans le makefile en tant que cible, MAKE met à jour ou crée cette cible avant d'utiliser le fichier dépendant dans la cible originale (c'est ce que l'on appelle une dépendance de liens). MAKE supporte les lignes de dépendance multiples pour une même cible et une même cible peut avoir plusieurs lignes de commande.*"

Note :

L'utilitaire TOUCH.exe modifie la date et l'heure d'un ou plusieurs fichiers.

Ces modifications impacteront les dépendances temporelles d'un projet, il permet donc, couplé à Make, de forcer la recompilation de dépendances.

Vous pouvez utiliser celui de Turbo Pascal 7 ou mieux celui de la JVCL, JTouch.exe.

12-4 - Commande

"*Contient une liste de commandes qu'il faut exécuter pour construire ou reconstruire une cible.*"

Ce qui nous donne la construction suivante :

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

```
cible: dépendances
         commandes
```

12-5 - Règle

"*Une règle explique comment et quand construire certains fichiers qui sont les cibles d'une règle particulière. MAKE effectue les commandes sur les dépendances nécessaires pour créer ou mettre à jour la cible. Une règle peut également expliquer comment et quand effectuer une action. Elle énumère les fichiers qui sont nécessaires à la construction de la cible.*"

12-6 - Règle explicite

"Instructions applicables à certains fichiers. Elle s'applique à la liste précise de cibles que vous indiquez dans la règle."

12-7 - Règle implicite

Instructions générales applicables aux fichiers non concernés par les règles explicites.

Elles peuvent être vues comme des règles explicites génériques.

La différence entre ces 2 types de règles se situe dans la manière dont make décide quand la règle s'applique.

12-8 - Les directives conditionnelles

"Les directives de MAKE ressemblent aux directives des langages tels que Pascal et C. Dans MAKE, elles effectuent diverses opérations de contrôle, telles que l'affichage des commandes avant leur exécution. Les directives de MAKE commencent par un point ou par un point d'exclamation, et remplacent les options données sur la ligne de commande. Les directives qui commencent par un point d'exclamation doivent apparaître au début d'une nouvelle ligne."

12-9 - Les macros ou variables de MAKE

"Une macro est une variable que Make traduit par une chaîne à chaque fois que MAKE rencontre la macro dans le fichier make.

Par exemple : fichier = FMain.pas définit la macro appelée "fichier" qui représente la chaîne "FMain.pas". Lorsque Make rencontrera la macro ***$(fichier)****, il substituera la chaîne FMain.pas.*

Vous devez définir chaque macro sur une ligne distincte dans le fichier make, et chaque définition de macro doit commencer sur le premier caractère de la ligne. Les macros définies dans un fichier make remplacent celles définies sur la ligne de commande."

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://laurent-dardenne.developpez.com/articles/Delphi/2006/Make/

13 - Liens

Vous pouvez laisser vos commentaires sur ce tutoriel sur le blog Delphi.

Vous trouverez ici un tutoriel sur les principes de bases de l'outil Make autour du langage C.

Microsoft propose désormais l'outil MSBuild en remplacement de Make.exe.

WIKI.

Il existe plusieurs implémentations de Make offrant bien plus de fonctionnalités, notez que leurs syntaxes diffèrent de celle de Borland.

GNU Make sous Win32.

Tools Unix sous Win32.

Documentation GNU Make.

Un livre chez OReilly Managing Projects with GNU make.

Article sur l'intégration continu par Martin Fowler.

Produits commerciaux gérant les compilateurs Borland :

Automated Build Studio

FinalBuilder

Visual Build Pro

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Laurent Dardenne. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.