Table des matières

Principe général

Une exception est l'interruption de l'exécution du programme à la suite d'un événement particulier.

Le but des exceptions est de réaliser des traitements spécifiques aux événements qui en sont la cause. Ces traitements peuvent rétablir le programme dans son mode de fonctionnement normal, auquel cas son exécution reprend. Il se peut aussi que le programme se termine, si aucun traitement n'est approprié.

Le C++ (et Borland Builder) supporte les exceptions logicielles, dont le but est de gérer les erreurs qui surviennent lors de l'exécution des programmes. Lorsqu'une telle erreur survient, le programme doit lancer une exception (***throw***). L'exécution normale du programme s'arrête dès que l'exception est lancée, et le contrôle est passé à un gestionnaire d'exception. Lorsqu'un gestionnaire d'exception s'exécute, on dit qu'il a attrapé l'exception (***catch***).

Les exceptions permettent une gestion simplifiée des erreurs, parce qu'elles en reportent le traitement. Le code peut alors être écrit sans se soucier des cas particuliers, ce qui le simplifie grandement. Les cas particuliers sont traités dans les gestionnaires d'exception.

C++

Lancer une exception

Lancer une exception consiste à retourner une erreur sous la forme d'une valeur (message, code, objet exception) dont le type peut être quelconque (`int`, `char*`, `MyExceptionClass`, ...).

Le lancement se fait par l'instruction `throw`.

Attraper une exception

Pour attraper une exception, il faut qu'un bloc encadre l'instruction directement, ou indirectement, dans la fonction même ou dans la fonction appelante, ou à un niveau supérieur. Dans le cas contraire, le système récupère l'exception et met fin au programme.

Les instructions `try` et `catch` sont utilisées pour attraper les exceptions.

Remarque : l'exécution normale reprend après le bloc ***try{...}catch{...}*** et non après le *throw*.

Borland C++ Builder

Introduction

C++Builder comprend un ensemble de classes d'exception incorporées pour gérer automatiquement les erreurs de division par zéro, les erreurs d'E/S, les transtypages incorrects et de nombreuses autres conditions d'exception. Toutes les classes d'exception VCL et CLX dérivent d'un objet racine appelé Exception.

La classe Exception encapsule les propriétés et les méthodes fondamentales de toutes les exceptions VCL et fournit une interface pour les applications gérant les exceptions.

On peut transmettre des exceptions à un bloc catch qui prend un paramètre de type Exception. Pour intercepter des exceptions VCL et CLX, on utilisera la syntaxe suivante en spécifiant la classe d'exception que l'on souhaite intercepter :

`catch (exception_class &exception_variable)` // on utilisera de préférence une référence

Exemple déclenchant une exception VCL ou CLX :

```
void __fastcall TForm1::ThrowException(TObject *Sender)
{
  try
  {
    throw Exception("Une erreur s'est produite");
  }
  catch(const Exception &E)
  {
    ShowMessage(AnsiString(E.ClassName())+ E.Message);
  }
}
```

Les classes d'exception VCL/CLX sont décrites dans le tableau suivant.

Classe d'exception	***Description***
EAbort	Interrompt une séquence d'événements sans afficher de boîte de dialogue de message d'erreur.
EAccessViolation	Vérifie la présence d'erreurs d'accès mémoire.
EBitsError	Empêche les tentatives incorrectes d'accès à un tableau booléen.
EComponentError	Signale une tentative incorrecte de recensement ou de modification du nom d'un composant.
EConvertError	Indique des erreurs de conversion d'objet ou de chaîne.
EDatabaseError	Spécifie une erreur d'accès à une base de données.
EDBEditError	Intercepte des données incompatibles avec un masque spécifié.
EDivByZero	Intercepte des erreurs de division par zéro des entiers.
EExternalException	Indique un code d'exception non reconnu.
EInOutError	Représente une erreur d'E/S de fichier.
EIntOverflow	Spécifie des calculs d'entiers dont les résultats sont trop élevés pour le registre alloué.
EInvalidCast	Vérifie s'il existe des transtypages incorrects.
EInvalidGraphic	Indique une tentative d'utilisation d'un format de fichier graphique incorrect.
EInvalidOperation	Se produit lorsque des opérations incorrectes sont tentées sur un composant.
EInvalidPointer	Se produit suite à des opérations de pointeur incorrect.
EMenuError	Implique un problème dû à un élément de menu.
EOleCtrlError	Détecte des problèmes de liaison avec les contrôles ActiveX.
EOleError	Spécifie des erreurs d'automation OLE.
EPrinterError	Signale une erreur d'impression.

EPropertyError	Se produit suite à des tentatives infructueuses de définition de la valeur d'une propriété.
ERangeError	Indique que la valeur d'un entier est trop élevée pour le type déclaré auquel il est affecté.
ERegistryException	Spécifie des erreurs de registre.
EZeroDivide	Intercepte des erreurs de division par zéro de virgule flottante.

Remarque : les classes d'exception VCL et CLX incorporées gèrent par défaut la plupart des et simplifient la programmation. Dans certaines situations, il est nécessaire de créer ses propres classes d'exception pour gérer des situations uniques. On peut alors déclarer une nouvelle classe d'exception en créant un descendant (héritage) de type Exception et en créant autant de constructeurs que nécessaire (le plus simpe étant de copier les constructeurs depuis une classe existant dans Sysutils.hpp).

La classe Exception

C' est la classe de base de toutes les exceptions d'exécution dans Borland C++ Builder.

Exception encapsule les propriétés et méthodes fondamentales à toutes les exceptions. On l'utilise comme classe de base pour la création d'exceptions personnalisées.

Toutes les méthodes introduites pour l'objet Exception sont des constructeurs offrant des moyens alternatifs pour créer des messages d'exception. Certains constructeurs créent également des identificateurs de contexte d'aide. Une application appelle généralement ces constructeurs de manière dynamique quand une exception se produit.

Les messages des exceptions peuvent être des chaînes codées "en dur", des chaînes formatées ou des chaînes (éventuellement formatées) chargées depuis une ressource de l'application.

Les exceptions sont déclenchées quand une erreur d'exécution se produit dans une application, par exemple une tentative de division par zéro. Généralement, quand une exception est déclenchée, une instance d'exception affiche une boîte de dialogue décrivant la condition d'erreur. Si une application ne traite pas la condition d'exception, le gestionnaire d'exception par défaut est appelé. Ce gestionnaire affiche également une boîte de dialogue avec un bouton OK qui permet normalement à une application de poursuivre les traitements quand l'utilisateur clique sur OK.

L'objet Exception offre une interface homogène aux conditions d'erreur et permet aux applications de gérer les conditions d'erreur d'une manière élégante. Les applications peuvent :

- Intercepter et gérer des exceptions spécifiques dans des blocs try..catch.
- Protéger le code allouant des ressources de conditions d'exception en incorporant le code dans les blocs try..__finally (__finally est une extension de langage C++Builder qui n'existe pas en C++).
- Déclencher des exceptions ou des exceptions personnalisées en réponse aux conditions du programme.

Remarque : la pratique courante veut que le nom de toutes les exceptions dérivées commence par la lettre 'E' suivie d'un nom abrégé descriptif.

Remarque : __finally, qui est spécifique à Builder et à Java, permet de définir un bloc d'instructions qui sera exécuté dans tous les cas (exceptions ou pas).

L'événement OnException

Il se produit quand une exception non gérée survient dans l'application.

L'événement OnException permet de modifier le comportement par défaut ayant lieu quand une exception non gérée a lieu dans l'application. Le gestionnaire d'événement OnException est appelé automatiquement par la méthode HandleException.

OnException ne gère que les exceptions se produisant durant le traitement des messages. Les exceptions ayant lieu avant ou après l'exécution de la méthode Run de l'application ne génèrent pas d'événements OnException.

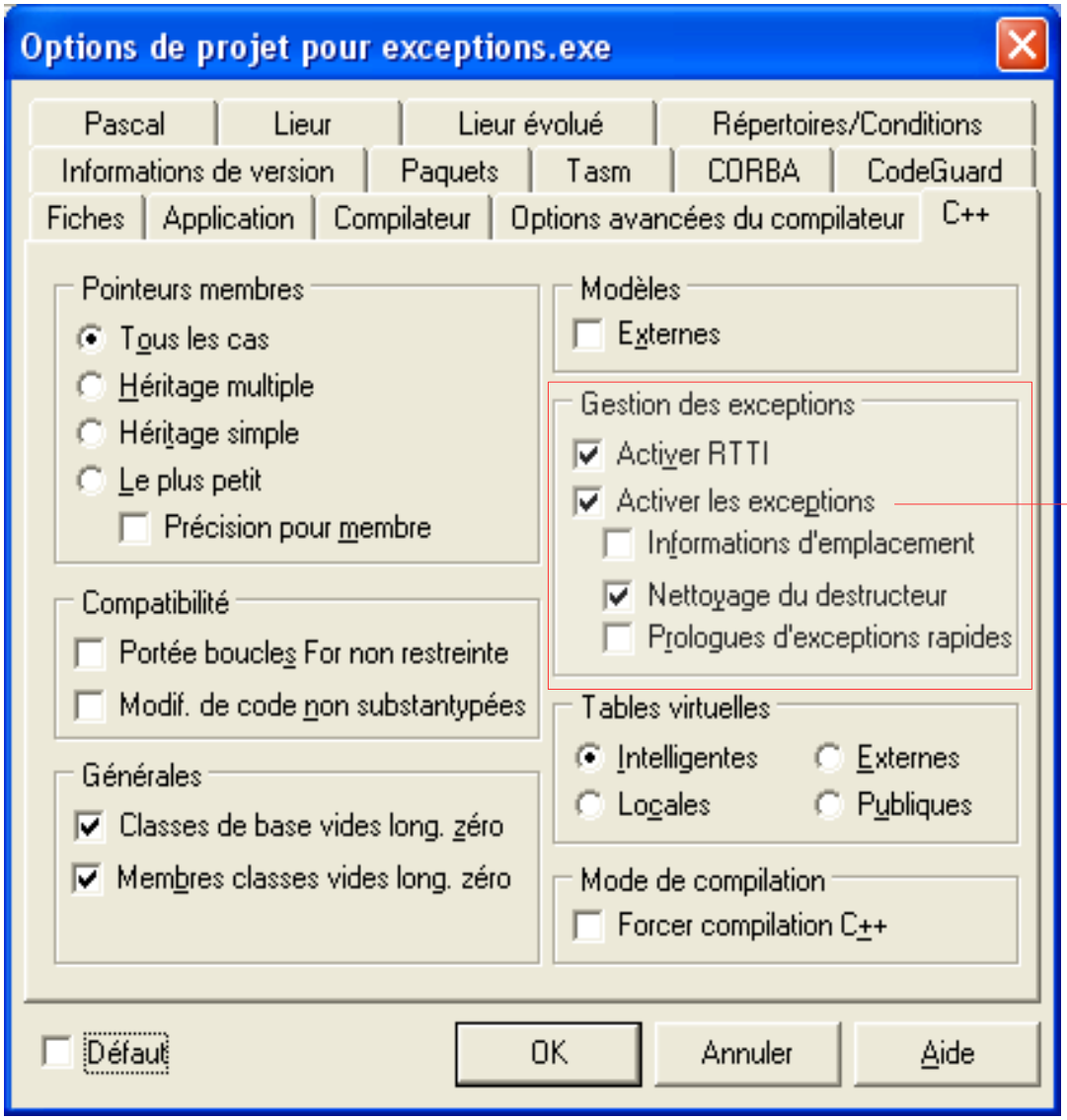
Si une exception se produit dans les blocs try du code de l'application, l'application appelle automatiquement la méthode HandleException. HandleException appelle le gestionnaire OnException, s'il est défini (sauf si l'objet exception est EAbort). S'il n'y a pas de gestionnaire, elle appelle ShowException pour afficher une boîte de dialogue indiquant qu'une erreur a eu lieu.

TExceptionEvent est le type des événements OnException. Il désigne une méthode gérant les exceptions dans l'application. Le paramètre Sender désigne l'objet ayant déclenché l'exception, et E indique l'objet de l'exception.

Configuration

Activer les exceptions

Pour cela, il suffit d'aller dans Projet | Options :

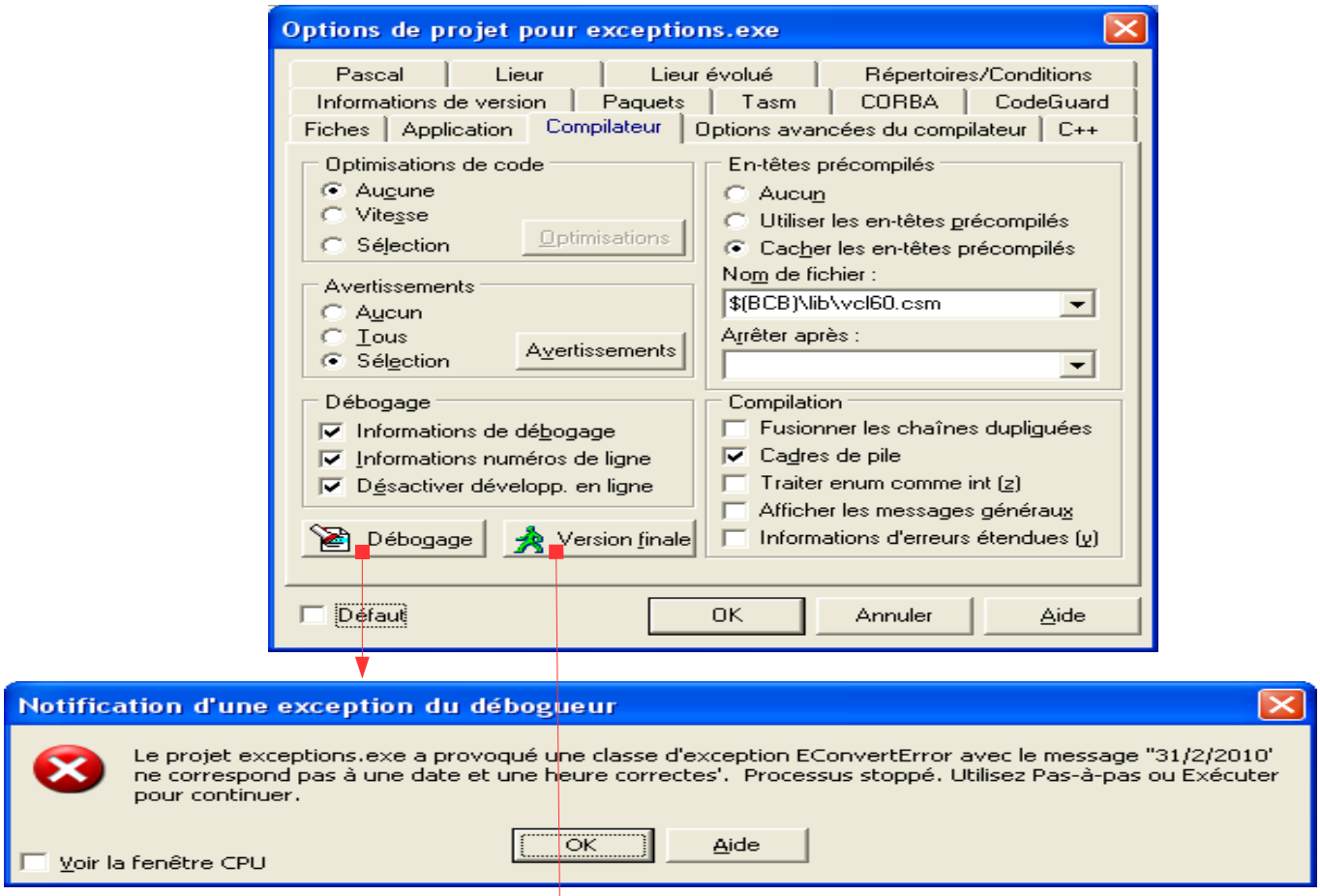


Pour gérer les exceptions, il faut activer cette option

Arrêter les exceptions

Il peut parfois être utile d'ignorer les exceptions dans C++Builder. En effet, celui-ci signale même les exceptions "rattrapées" par le bloc try/catch. Pour cela, il suffit d'aller dans Outils | Options du débogueur, puis dans l'onglet Exceptions du langage. Décochez ensuite la case Arrêter sur exceptions C++.

Version débogage vs Version Finale

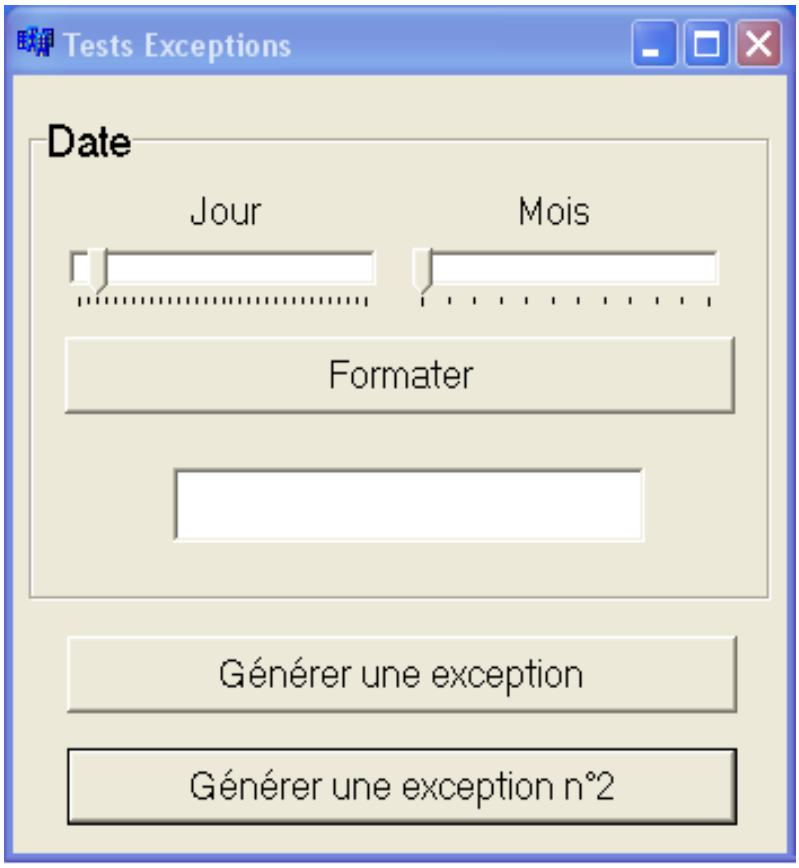


La date du 31 février génère une exception :

- en **mode « Débogage »**, l'exception est capturée par l'EDI Builder et permet de reprendre la main pour une exécution en mode pas à pas par exemple ;

- en **mode « Version Finale »** ***(release)***, le bloc *catch* s'exécute ...

Exemples sous Borland Builder



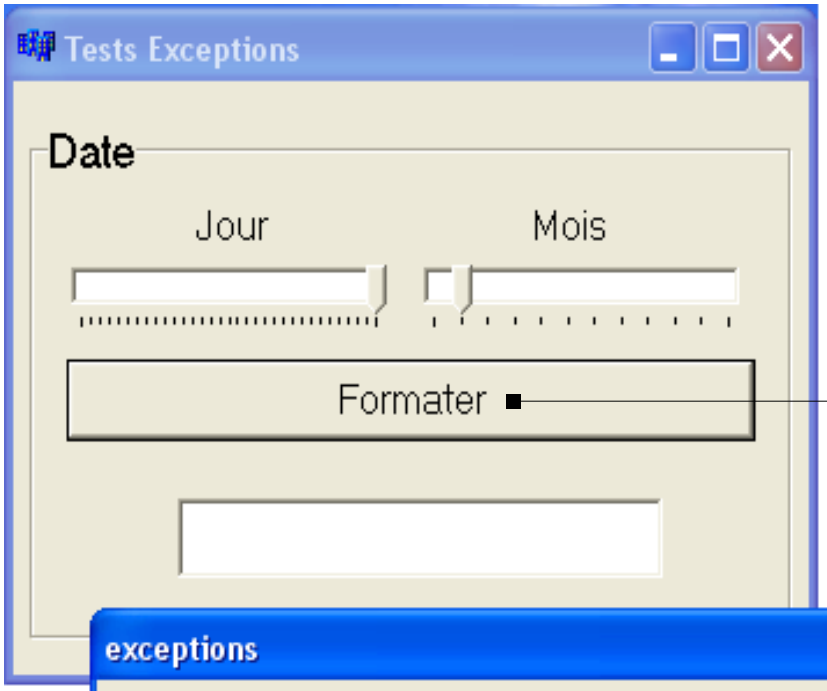
Cette application permet :

- Capturer une exception EConvertError
- Générer une exception avec throw
- Générer une exception de type int

- Intercepter le gestionnaire d'exception par défaut de Builder (événement OnException)

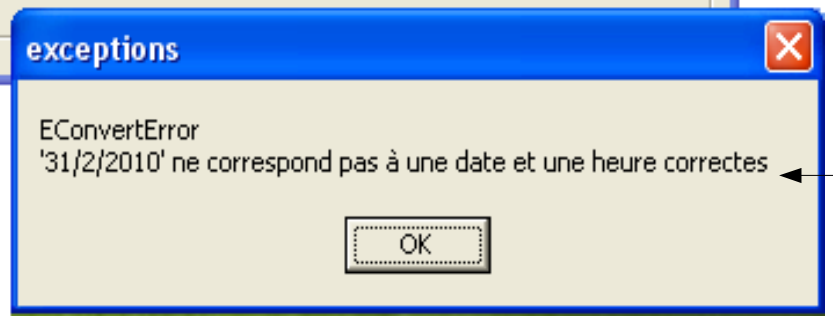
- Créer une classe héritant de Exception

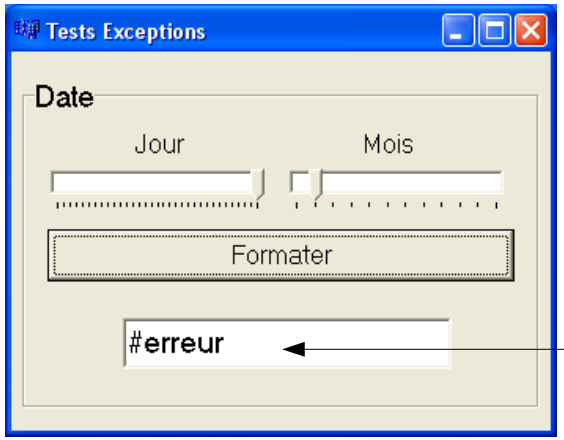
- Capturer toutes les exceptions

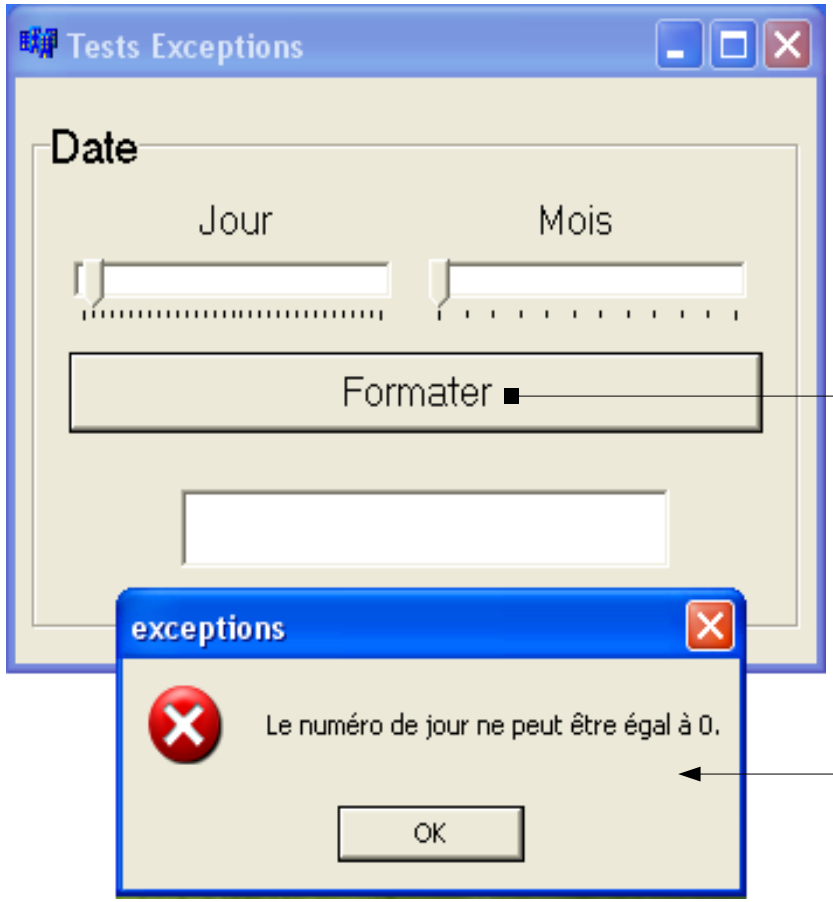


Le 31 février ?

```
try
{
    sprintf(strDate, "%d/%d/2010",
            trackBarJour->Position,
            trackBarMois->Position);
    tempDate = StrToDateTime(strDate);
}
catch (EConvertError &E)
{
    ShowMessage(AnsiString(E.ClassName())
+ "\n" + AnsiString(E.Message));
    editDate->Text = "#erreur";
}
```

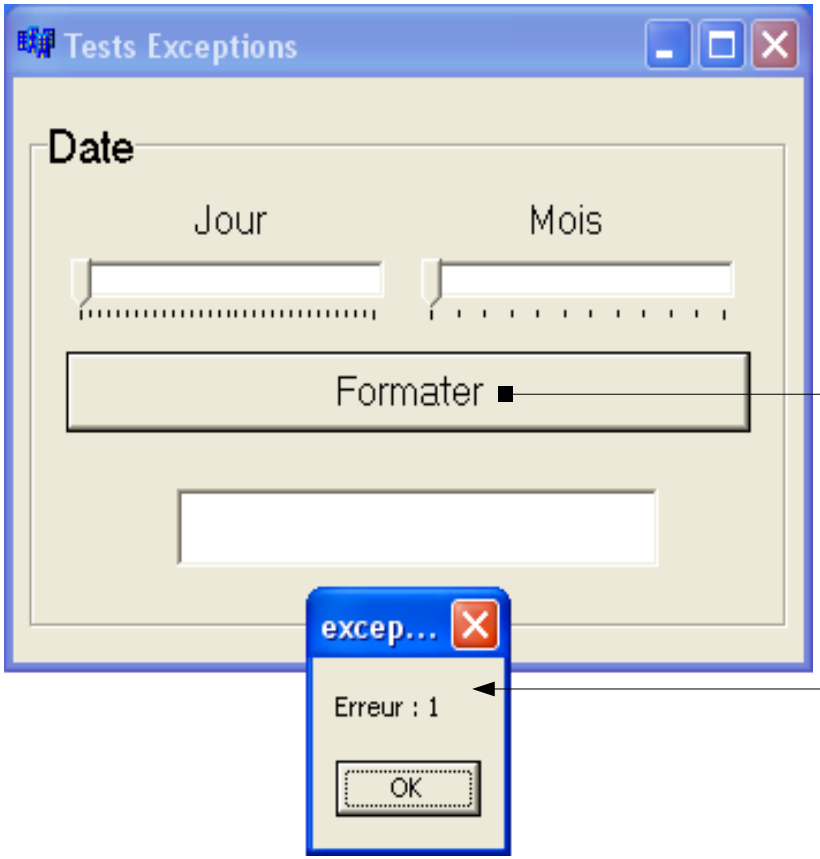






Le 0 janvier ?

```
try
{
    if(trackBarJour->Position == 0)
       throw Exception("Le numéro de jour
ne peut être égal à 0");
}

//Builder fournit une gestion des
exceptions par défaut :
Application->ShowException(E);
```



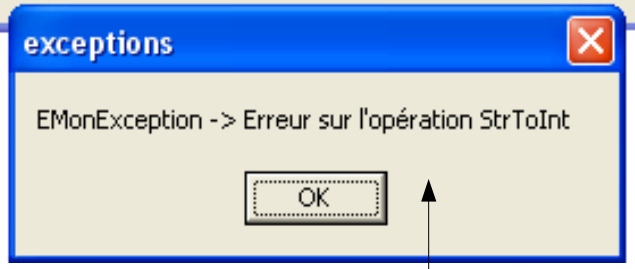
Le -1 janvier ?

```
try
{
    if(trackBarJour->Position == -1)
     throw 1; // lance une exception de type "int"
}
catch (int numErreur)
{
    ShowMessage("Erreur: "+ IntToStr(numErreur));
}
```

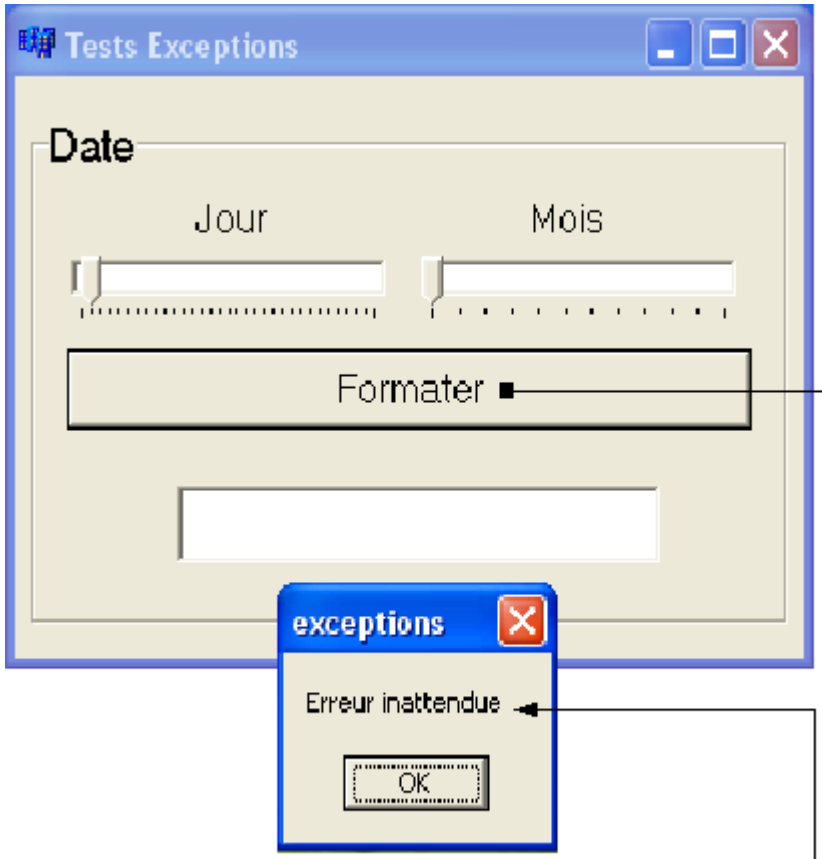

```
void __fastcall TihmTestExceptions::FormCreate(TObject *Sender)
{
    //installe le gestionnaire d'exceptions
    Application->OnException = AppException;
}

/*********************************************/

void __fastcall TihmTestExceptions::boutonGenererExceptionClick(
      TObject *Sender)
{
  int i = StrToInt("AAA"); // génère une exception !
}

void __fastcall TihmTestExceptions::AppException(TObject *Sender,
Sysutils::Exception *E)
{
  AnsiString Message;

  //Affichage personnalisé :
  Message = E->ClassName();
  Message+= " -> ";
  Message+= E->Message;
  ShowMessage(Message);

  //Faire le choix de terminer l'application ?
  //Application->Terminate();
}
```


```
void __fastcall TihmTestExceptions::FormCreate(TObject *Sender)
{
    //installe le gestionnaire d'exceptions
    Application->OnException = AppException;
}

void __fastcall TihmTestExceptions::AppException(TObject *Sender,
Sysutils::Exception *E)
{
  // vide : permet de détourner la prise en charge des
exceptions par défaut de Builder
}
/*****************************************/
```

```
void __fastcall
TihmTestExceptions::boutonGenererException2Click(
      TObject *Sender)
{
  //throw new EMonException("Mon exception !");

  //ou :
  try
  {
    int i = StrToInt("AAA"); // génère une exception
  }
  catch(...)
  {
    throw new EMonException("Erreur sur l'opération
StrToInt");
  }
}
```




```
class EMonException : public Exception
{  public:
    __fastcall EMonException(const
AnsiString Msg)
{ AnsiString Message;
  Message = this->ClassName()+" -> ";
  Message+= Msg;
  ShowMessage(Message);
}};
```



Le 0 janvier ?

```
try
{
    sprintf(strDate, "%d/%d/2010",
            trackBarJour->Position,
            trackBarMois->Position);
     tempDate = StrToDateTime(strDate);
     editDate->Text = tempDate.DateString();
}
catch(...)
{
    //Ce catch capture toutes les exceptions
qui n'ont pas déjà été prises
    ShowMessage("Erreur inattendue");
}
```

Annexe 1 : code source

Fichier ihmExceptions.h :

```
#ifndef ihmExceptionsH
#define ihmExceptionsH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <SysUtils.hpp>

#include <stdio.h>
#include <string.h>

//---------------------------------------------------------------------------
class TihmTestExceptions : public TForm
{
__published:    // Composants gérés par l'EDI
    TTrackBar *trackBarJour;
    TTrackBar *trackBarMois;
    TButton *boutonFormater;
    TGroupBox *boxDate;
    TLabel *labelJour;
    TLabel *labelMois;
    TEdit *editDate;
    TButton *boutonGenererException;
    TButton *boutonGenererException2;
    void __fastcall boutonFormaterClick(TObject *Sender);
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall boutonGenererExceptionClick(TObject *Sender);
    void __fastcall boutonGenererException2Click(TObject *Sender);
private:  // Déclarations de l'utilisateur
public:         // Déclarations de l'utilisateur
    __fastcall TihmTestExceptions(TComponent* Owner);
    void __fastcall AppException(TObject *Sender, Sysutils::Exception *E);
};
//---------------------------------------------------------------------------
extern PACKAGE TihmTestExceptions *ihmTestExceptions;
//---------------------------------------------------------------------------
#endif
```

Fichier ihmExceptions.cpp :

```
#include <vcl.h>
#pragma hdrstop
#include "ihmExceptions.h"
#include "EMonException.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
```

```
TihmTestExceptions *ihmTestExceptions;

__fastcall TihmTestExceptions::TihmTestExceptions(TComponent* Owner)
    : Tform(Owner) {  }
//---------------------------------------------------------------------------
void __fastcall TihmTestExceptions::boutonFormaterClick(TObject *Sender)
{
  TDateTime tempDate;
  char strDate[16];

  editDate->Text = "";
  try
  {
    if(trackBarJour->Position == -1)
    throw 1;    // lance une exception de type "int"
    if(trackBarJour->Position == 0)
      throw Exception("Le numéro de jour ne peut être égal à 0"); //on
retrouvera la description de l'exception dans E.Message

    /* formatage de la date jj/mm/aaaa */
    sprintf(strDate, "%d/%d/2010", trackBarJour->Position, trackBarMois-
>Position);
    tempDate = StrToDateTime(strDate);
    editDate->Text = tempDate.DateString();
  }
  catch (EConvertError &E)
  {
     ShowMessage(AnsiString(E.ClassName()) + "\n" + AnsiString(E.Message));
     editDate->Text = "#erreur";
  }
  catch(int numErreur)
  {
    // capture toutes les exceptions de type int
    ShowMessage("Erreur : " + IntToStr(numErreur));
  }
  catch(...)
  {
      // Ce catch capture toutes les exceptions qui n'ont pas déjà été prises
      ShowMessage("Erreur inattendue");;
  }
}
//---------------------------------------------------------------------------

void __fastcall TihmTestExceptions::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TihmTestExceptions::AppException(TObject *Sender,
Sysutils::Exception *E)
{
  AnsiString Message;

  //Affichage personnalisé :
  /*Message = E->ClassName();
  Message+= " -> ";
  Message+= E->Message;
  ShowMessage(Message);*/

  //ou tout simplement :
  //Application->ShowException(E);

  //Faire le choix de terminer l'application ?
  //Application->Terminate();
}
void __fastcall TihmTestExceptions::boutonGenererExceptionClick(
      TObject *Sender)
{
  int i = StrToInt("AAA");
}
//---------------------------------------------------------------------------

void __fastcall TihmTestExceptions::boutonGenererException2Click(
      TObject *Sender)
{
  //throw new EMonException("Mon exception !");
  //ou :
  try
  {
    int i = StrToInt("AAA");
  }
  catch(...)
  {
    throw new EMonException("Erreur sur l'opération StrToInt");
  }
}
//---------------------------------------------------------------------------
```

Fichier EMonException.h :

```
#ifndef EMonExceptionH
#define EMonExceptionH

class EMonException : public Exception
{
   public:
    __fastcall EMonException(const AnsiString Msg);
    __fastcall EMonException(const AnsiString Msg, System::TVarRec const *
Args, const int Args_Size);
    __fastcall EMonException(int Ident)/* overload */;
    __fastcall EMonException(System::PResStringRec ResStringRec)/* overload
*/;
    __fastcall EMonException(int Ident, System::TVarRec const * Args, const
```

```
int Args_Size)/* overload */;
    __fastcall EMonException(System::PResStringRec ResStringRec,
System::TVarRec const * Args, const int Args_Size)/* overload */;
    __fastcall EMonException(const AnsiString Msg, int AHelpContext);
    __fastcall EMonException(const AnsiString Msg, System::TVarRec const *
Args, const int Args_Size, int AHelpContext);
    __fastcall EMonException(int Ident, int AHelpContext)/* overload */;
    __fastcall EMonException(System::PResStringRec ResStringRec, int
AHelpContext)/* overload */;
    __fastcall EMonException(System::PResStringRec ResStringRec,
System::TVarRec const * Args, const int Args_Size, int AHelpContext)/*
overload */;
    __fastcall EMonException(int Ident, System::TVarRec const * Args, const
int Args_Size, int AHelpContext)/* overload */;
};

#endif
```

Fichier EMonException.cpp :

```
#include <vcl.h>
#pragma hdrstop
#include "EMonException.h"
#pragma package(smart_init)

__fastcall EMonException::EMonException(const AnsiString Msg)
    : Exception( Msg )
{
  AnsiString Message;

  //Affichage personnalisé
  Message = this->ClassName();
  Message+= " -> ";
  Message+= Msg;
  ShowMessage(Message);
}
__fastcall EMonException::EMonException(const AnsiString Msg,
            System::TVarRec const * Args, const int Args_Size) :
              Exception( Msg, Args, Args_Size )
{

}

__fastcall EMonException::EMonException(int Ident) :
              Exception( Ident )
{

}
__fastcall EMonException::EMonException(
                                System::PResStringRec ResStringRec) :
                                   Exception( ResStringRec )
{

}

__fastcall EMonException::EMonException(int Ident,
                                        System::TVarRec const * Args,
                                        const int Args_Size) :
                  Exception( Ident, Args, Args_Size )
{

}
```

```
__fastcall EMonException::EMonException(
               System::PResStringRec ResStringRec, System::TVarRec const * Args,
               const int Args_Size) :
                  Exception( ResStringRec, Args, Args_Size )
{

}

__fastcall EMonException::EMonException(const AnsiString Msg,
                  int AHelpContext) : Exception( Msg, AHelpContext )
{

}

__fastcall EMonException::EMonException(const AnsiString Msg,
                  System::TVarRec const * Args,
                  const int Args_Size,
                  int AHelpContext) :
                  Exception( Msg, Args, Args_Size, AHelpContext )
{

}

__fastcall EMonException::EMonException(int Ident,
                                                    int AHelpContext) :
                                       Exception( Ident, AHelpContext )
{

}

__fastcall EMonException::EMonException(
                                 System::PResStringRec ResStringRec,
                                 int AHelpContext) :
                                 Exception( ResStringRec,
                                             AHelpContext )
{

}

__fastcall EMonException::EMonException(
                               System::PResStringRec ResStringRec,
                               System::TVarRec const * Args,
                               const int Args_Size,
                               int AHelpContext) :
                               Exception( ResStringRec, Args, Args_Size,
                                          AHelpContext )
{

}

__fastcall EMonException::EMonException(int Ident,
                                        System::TVarRec const * Args,
                                        const int Args_Size,
                                        int AHelpContext) :
                               Exception( Ident, Args, Args_Size, AHelpContext )

{

}
```

Annexe 2 : aide Borland Builder