



無料電子ブック

学習

Embarcadero Delphi

Free unaffiliated eBook created from
Stack Overflow contributors.

#delphi

.....	1
1: Embarcadero Delphi	2
.....	2
.....	2
Examples	3
.....	3
VCL 'Hello World'	3
WinAPI MessageBox 'Hello World'	4
FireMonkeyHello World	4
2: DelphiRTTI	5
.....	5
.....	5
Examples	5
.....	5
3: Firemonkey	7
Examples	7
TRectangle	7
4: For Loops	8
.....	8
.....	8
Examples	8
.....	8
.....	8
.....	9
.....	9
.....	10
5: GUI	11
Examples	11
GUIPostMessage	11
.....	11
.....	12

6: tryexceptfinally	15
.....	15
Examples	15
try..finally	15
.....	15
.....	16
.....	16
2	17
7: TStringList	18
Examples	18
.....	18
Key-Value	18
8:	20
.....	20
Examples	20
.....	20
.....	21
.....	21
.....	22
9:	23
Examples	23
TArray.Sort	23
TList	23
TList	23
TList	24
10: TDataSet	25
.....	25
Examples	25
FireDAC	25
11:	28
.....	28
.....	28

Examples.....	28
.....	28
- Until.....	28
.....	29
12:	30
Examples.....	30
CreateProcess.....	30
13:	32
Examples.....	32
.....	32
.....	32
.....	32
.....	33
.....	33
.....	33
.....	34
14:	36
Examples.....	36
Windows API GetTickCount.....	36
TStopwatch.....	36
15:	38
.....	38
Examples.....	38
.....	38
.....	40

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [embarcadero-delphi](#)

It is an unofficial and free Embarcadero Delphi ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Embarcadero Delphi.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: Embarcadero Delphi

Delphiは、Borland Turbo PascalのルーツをつObject Pascalにつくです。なアプリケーションRADをサポートするようにされたのIDEがしています。

これは、のコードベースからクロスプラットフォームのネイティブコンパイルみアプリケーションをにします。サポートされているプラットフォームはWindows、OSX、iOS、Androidです。

それには2つのビジュアルフレームワークがしています。

- VCLビジュアルコンポーネントライブラリは、Windowsににされており、Windowsネイティブコントロールとカスタムカスタムコントロールのをサポートしています。
- FMXサポートされているすべてのプラットフォームのFireMonkeyクロスプラットフォームフレームワーク

バージョン

バージョン	バージョン		
1	1.0	ボーランドデルファイ	1995-02-14
2	2.0	ボーランドデルファイ 2	1996-02-10
3	3.0	ボーランド Delphi 3	1997-08-05
4	4.0	ボーランドデルファイ 4	1998717
5	5.0	ボーランド Delphi 5	1999-08-10
6	6.0	ボーランドデルファイ 6	2001-05-21
7	7.0	ボーランド Delphi 7	200289
8	8.0	Borland Delphi 8 for .NET	2003-12-22
2005	9.0	ボーランド Delphi 2005	20041012
2006	10.0	Borland Delphi 2006	2005-11-23
2007	11.0	CodeGear Delphi 2007	2007-03-16
2009	12.0	CodeGear Delphi 2009	2008-08-25
2010	14.0	Embarcadero RAD Studio 2010	2009-08-15
XE	15.0	Embarcadero RAD Studio XE	2010-08-30

バージョン	バージョン		
XE2	16.0	Embarcadero RAD Studio XE2	2011-09-02
XE3	17.0	Embarcadero RAD Studio XE3	2012-09-03
XE4	18.0	Embarcadero RAD Studio XE4	2013-04-22
XE5	19.0	Embarcadero RAD Studio XE5	2013-09-11
XE6	20.0	Embarcadero RAD Studio XE6	2014-04-15
XE7	21.0	Embarcadero RAD Studio XE7	2014-09-02
XE8	22.0	Embarcadero RAD Studio XE8	2015-04-07
10シアトル	23.0	Embarcadero RAD Studio 10シアトル	2015-08-31
10.1ベルリン	24.0	Embarcadero RAD Studio 10.1ベルリン	2016-04-20
10.2	25.0	Embarcadero RAD Studio 10.2 Tokyo	2017-03-22

Examples

こんにちは

*HelloWorld.dpr*というファイルにされたこのプログラムは、コンソールに "Hello World"をするコンソールアプリケーションにコンパイルされます。

```
program HelloWorld;

{$APPTYPE CONSOLE}

begin
  WriteLn('Hello World');
end.
```

VCLをして 'Hello World'をする

このプログラムは、DelphiのデフォルトのUIコンポーネントライブラリであるVCLをして、メッセージボックスに「Hello World」をします。VCLは、にされているWinAPIコンポーネントのほとんどをみんでいます。このようにして、ウィンドウハンドルでするなく、はるかににできます。

これはVcl.Dialogsなどをめるには、セミコロンでわるのコンマリリストをusesブロックをしuses。

```
program HelloWindows;
```

```
uses
  Vcl.Dialogs;

begin
  ShowMessage('Hello Windows');
end.
```

WinAPI MessageBoxを使って 'Hello World'をする

このプログラムは、Windows API WinAPIをして、"Hello World"をメッセージボックスにします。

このはWindowsなどをめるには、セミコロンでわるのコンマリリストをむusesブロックをします。

```
program HelloWorld;

uses
  Windows;

begin
  MessageBox(0, 'Hello World!', 'Hello World!', 0);
end.
```

FireMonkeyをしたクロスプラットフォームのHello World

XE2

```
program CrossPlatformHelloWorld;

uses
  FMX.Dialogs;

{$R *.res}

begin
  ShowMessage('Hello world!');
end.
```

DelphiでサポートされているほとんどのプラットフォームWin32 / Win64 / OSX32 / Android32 / iOS32 / iOS64もコンソールをサポートしているので、WriteLnはうまくします。

GUI iOSデバイスやのAndroidデバイスがなプラットフォームの、のFireMonkeyのはうまくいきます。

オンラインでEmbarcadero Delphiをむ <https://riptutorial.com/ja/delphi/topic/599/embarcadero-delphi>

2: DelphiでのRTTIの

き

Delphiは10にRTTI Runtime Type Informationをしました。しかしでも、くのはリスクとメリットをにしています。

するに、は、にメモリにされるオブジェクトのデータにするです。

RTTIは、オブジェクトのがのクラスまたはそののであるかどうかをするをします。

デルタのRTTI - EXPLAINED

Delphiのの - それはでもできますか Brian Longのでは、DelphiのRTTIをしています。 Brianは、のをするためにDelphiでのRTTIのサポートがにされたとしています。が、はのコードのをするためにもできます。このではRTTIクラスのといくつかのをします。

としては、のプロパティのみるときみ、のをたないのプロパティ、あるコンポーネントからのコンポーネントへのプロパティのコピーなどがあります。

Examples

クラス

このでは、 `ClassType` および `ClassParent` プロパティをしてコンポーネントのをするをします。ボタン `Button1: TButton` とリストボックス `ListBox1: TListBox` をフォーム `TForm1` ます。

ユーザーがボタンをクリックすると、ボタンのクラスのとクラスのがリストボックスにされます。

```
procedure TForm1.Button1Click(Sender: TObject) ;
var
    ClassRef: TClass;
begin
    ListBox1.Clear;
    ClassRef := Sender.ClassType;
    while ClassRef <> nil do
    begin
        ListBox1.Items.Add(ClassRef.ClassName) ;
        ClassRef := ClassRef.ClassParent;
    end;
end;
```

リストボックスには、ユーザーがボタンをクリックしたにのがまれます。

- TButton
- TButtonControl

- TWinControl
- TControl
- TComponent
- TPersistent
- TObject

オンラインでDelphiでのRTTIのをむ <https://riptutorial.com/ja/delphi/topic/9578/delphiでのrttiの>

3: Firemonkeyでのアニメーションの

Examples

TRectangleをする

1. のマルチデバイスFiremonkeyアプリケーションをします。
2. フォームのをドロップします。
3. オブジェクトインスペクタウィンドウF11で、RotationAngleをクリックしてドロップダウンボタンをクリックし、"Create New TFloatAnimation"をします。
4. オブジェクトインスペクタウィンドウがにしくされたTFloatAnimationにりわります。また、メニューShift + Alt
 - F11。
5. TFloatAnimationのオブジェクトインスペクタで、をのでします。たちの、1をることができます。StartValueプロパティをそのままにして、StopValueタイプで360Degrees、すべてがなくなりますとします。また、ループオプションをオンにしますこれはコードからするまでアニメーションをループします。

アニメーションをしました。すべては、それをオンにすることですフォームの2つのボタンをドロップし、の1つを ""、2の - ""をびします。のボタンきみのOnClickイベントで

```
FloatAnimation1.Start;
```

OnClick2ボタンコード

```
FloatAnimation1.Stop;
```

TFloatAnimationのをした - とをびすときにもします。

すぐあなたのプロジェクトをし、[スタート]ボタンをクリックしてお楽しみください。

オンラインでFiremonkeyでのアニメーションのをむ

<https://riptutorial.com/ja/delphi/topic/5383/firemonkeyでのアニメーションの>

4: For Loops

- OrdinalVariable= LowerOrdinalValueをUpperOrdinalValueにするdo begin {loop-body} end;
- OrdinalVariable= UpperOrdinalValue downto LowerOrdinalValue do begin {ループボディ} end;
- コレクションのEnumerableVariableにしてdo begin {loop-body} end;
- デルファイのfor-loopからステップをするにはもしていない₁のにします。
- ordinal、たとえばIntegerのローカルをしてループする、とは1だけされます。そのようなへのは、ループにしません。

Examples

シンプルなループ

forループは、からまでりします。

```
program SimpleForLoop;  
  
{$APPTYPE CONSOLE}  
  
var  
    i : Integer;  
begin  
    for i := 1 to 10 do  
        WriteLn(i);  
    end.  
end.
```

1
2
3
4
5
6
7
8
9
10

のをループする

2005

は、sをりしします。これは、ループこのではcのタイプがされるのタイプとするり、またはセットのをループするとにします。

```

program ForLoopOnString;

{$APPTYPE CONSOLE}

var
  s : string;
  c : Char;
begin
  s := 'Example';
  for c in s do
    WriteLn(c);
  end.

```

E
 バツ
 a
 m
 p
 l
 e

ループの

`for`ループは、「カウントダウン」のとして、からまでをします。

```

program CountDown;

{$APPTYPE CONSOLE}

var
  i : Integer;
begin
  for i := 10 downto 0 do
    WriteLn(i);
  end.

```

10
 9
 8
 7
 6
 5
 4
 3
 2
 1
 0

をするループの

`for`ループは、のをします。

```

program EnumLoop;

uses
  TypInfo;

type
  TWeekdays = (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday);

var
  wd : TWeekdays;
begin

  for wd in TWeekdays do
    WriteLn(GetEnumName(TypeInfo(TWeekdays), Ord(wd)));

  end.

```

の

forループはのをします。

```

program ArrayLoop;
{$APPTYPE CONSOLE}
const a : array[1..3] of real = ( 1.1, 2.2, 3.3 );
var f : real;
begin
  for f in a do
    WriteLn( f );
  end.

```

1,1

2,2

3,3

オンラインでFor Loopsをむ <https://riptutorial.com/ja/delphi/topic/4643/for-loops>

5: GUIのをちながらスレッドをする

Examples

バックグラウンドにスレッドをするレスポンスなGUIとスレッドからってする

PostMessage

のかかるプロセスのにGUIをさせるには、GUIがメッセージキューをするためのにな「コールバック」か、バックグラウンドワーカースレッドのがです。

らかのをうためにスレッドをいくつでもオフにするのはではありません。GUIがとをしたり、をしたりするとき、しいことがまります。

GUIにかをするには、コントロールやメッセージキュー/ポンプとのやりとりがです。それはにメインスレッドのコンテキストでわれるべきです。のスレッドのコンテキストではしてありません。

これをするはたくさんあります。

このでは、あなたがそれをするのでした、GUIは、スレッドのインスタンスにアクセスすることができ、なスレッドをしてそれをうことができますをしFreeOnTerminateするfalse、そしてスレッドがして「」したときのPostMessage。

にするワーカースレッドへののは、フォームのにされます。スレッドがすると、のするはnil-edになります。

これはのなです。するのあるスレッドがまだするかどうかをにできるようにするために、"Running"ブールをします。

ロックをしてこれらのリソースをするがあるかどうかをするがあります。

このでは、それがっているので、はありません。らは2つのだけののでされてStartThreadsとHandleThreadResults。のメソッドは、メインスレッドのコンテキストでのみされます。そのようにしておき、なるスレッドのコンテキストからこれらのメソッドをびすことをしないり、をするはありません。

```
type
  TWorker = class(TThread)
  private
    FFactor: Double;
    FResult: Double;
    FReportTo: THandle;
  protected
    procedure Execute; override;
  public
    constructor Create(const aFactor: Double; const aReportTo: THandle);
```

```

property Factor: Double read FFactor;
property Result: Double read FResult;
end;

```

コンストラクタはprivateメンバーをし、FreeOnTerminateをFalseにします。これは、メインスレッドがスレッドインスタンスにそのをいわせることをにするためです。

executeメソッドは、そのをい、そのコンストラクタでつけたハンドルにメッセージをして、doneをします。

```

procedure TWorker.Execute;
const
  Max = 100000000;var
  i : Integer;
begin
  inherited;

  FResult := FFactor;
  for i := 1 to Max do
    FResult := Sqrt(FResult);

  PostMessage(FReportTo, UM_WORKERDONE, Self.Handle, 0);
end;

```

このでは、PostMessageのがです。PostMessage「ちょうど」メインスレッドのメッセージポンプのキューにメッセージをれ、それがされるのを待つことはありません。にです。あなたがSendMessageをするは、をけにするでしょう。SendMessageはメッセージをキューにれ、されるまでします。するに、それはです。

カスタムUM_WORKERDONEメッセージのは、のようにされています。

```

const
  UM_WORKERDONE = WM_APP + 1;
type
  TUMWorkerDone = packed record
    Msg: Cardinal;
    ThreadHandle: Integer;
    unused: Integer;
    Result: LRESULT;
  end;

```

UM_WORKERDONE constは、WM_APPをのとしてし、WindowsまたはDelphi VCLMicrosoftのがするにしないようにします。

のフォームをしてスレッドをできます。のメンバーをするだけです。

```

private
  FRunning: Boolean;
  FThreads: array of record
    Instance: TThread;
    Handle: THandle;
  end;
end;
procedure StartThreads(const aNumber: Integer);

```

```
procedure HandleThreadResult (var Message: TUMWorkerDone); message UM_WORKERDONE;
```

ああ、サンプルコードでは、Memo1: TMemo;をとしていMemo1: TMemo;フォームのでは、"ロギングとレポート"のためにされます。

FRunningをすると、にGUIがクリックされないようにすることができます。 FThreadsは、されたスレッドのインスタンスポインタとハンドルをするためにされます。

スレッドをするプロシージャは、かなりなです。しているスレッドのセットがすでにあるかどうかのチェックからします。そうなら、それはただする。そうでないは、フラグをtrueにし、それぞれにハンドルをするスレッドをし、「」メッセージのをります。

```
procedure TForm1.StartThreads(const aNumber: Integer);
var
  i: Integer;
begin
  if FRunning then
    Exit;

  FRunning := True;

  Memo1.Lines.Add(Format('Starting %d worker threads', [aNumber]));
  SetLength(FThreads, aNumber);
  for i := 0 to aNumber - 1 do
  begin
    FThreads[i].Instance := TWorker.Create(pi * (i+1), Self.Handle);
    FThreads[i].Handle := FThreads[i].Instance.Handle;
  end;
end;
```

スレッドのハンドルもにれられます。これは、スレッドがしたことをすメッセージでけったものであり、スレッドのインスタンスにあると、アクセスがしになります。スレッドのインスタンスでハンドルをできるようにすることで、をるためにインスタンスをとしないたとえば、データベースにされているなど、FreeOnTerminateをTrueすることもできます。その、もちろんインスタンスへのをするはありません。

そのしみはHandleThreadResultにあります

```
procedure TForm1.HandleThreadResult (var Message: TUMWorkerDone);
var
  i: Integer;
  ThreadIdx: Integer;
  Thread: TWorker;
  Done: Boolean;
begin
  // Find thread in array
  ThreadIdx := -1;
  for i := Low(FThreads) to High(FThreads) do
    if FThreads[i].Handle = Cardinal(Message.ThreadHandle) then
      begin
        ThreadIdx := i;
        Break;
      end;
  end;
```

```

// Report results and free the thread, nilling its pointer and handle
// so we can detect when all threads are done.
if ThreadIdx > -1 then
begin
    Thread := TWorker(FThreads[i].Instance);
    Memo1.Lines.Add(Format('Thread %d returned %f', [ThreadIdx, Thread.Result]));
    FreeAndNil(FThreads[i].Instance);
    FThreads[i].Handle := nil;
end;

// See whether all threads have finished.
Done := True;
for i := Low(FThreads) to High(FThreads) do
    if Assigned(FThreads[i].Instance) then
    begin
        Done := False;
        Break;
    end;
if Done then
begin
    Memo1.Lines.Add('Work done');
    FRunning := False;
end;
end;

```

このメソッドは、にメッセージでしたハンドルをしてスレッドをします。するものがつかったは、そのインスタンスをしてスレッドのをしてします `FreeOnTerminate`は`False`、えていますか、インスタンスをし、インスタンスとハンドルのを`nil`にします。よりい。

に、スレッドのいずれかがまだかどうかをします。つからないは、「すべて」がされ、 `FRunning` フラグが`False`され、しいバッチをすることができます。

オンラインでGUIのをしながらスレッドをするをむ <https://riptutorial.com/ja/delphi/topic/1796/guiのをしながらスレッドをする>

6: try、except、およびfinallyの

1. try-except[[[EExceptionType doステートメントで]] [else]の[statements]をしてください。 [ステートメント];

してみてくださいtry [statements] finally [statements] end;

Examples

メモリリークをけるためのなtry..finallyの

にがしたにリソースメモリなどがれないようfinally、 try - finallyをしてtry。

のでは、をファイルにし、 TStringList をぎます。

```
procedure SaveStringToFile(const aFilename: TFilename; const aString: string);
var
    SL: TStringList;
begin
    SL := TStringList.Create; // call outside the try
    try
        SL.Text := aString;
        SL.SaveToFile(aFilename);
    finally
        SL.Free // will be called no matter what happens above
    end;
end;
```

ファイルのにがするかどうかにかかわらず、 SLはされます。はびしにられます。

しいオブジェクトのセーフリ

がオブジェクトをすときびしがしたものをするのではなく、がオブジェクトをリークさせないようにしてください。

```
function MakeStrings: TStrings;
begin
    // Create a new object before entering the try-block.
    Result := TStringList.Create;
    try
        // Execute code that uses the new object and prepares it for the caller.
        Result.Add('One');
        MightThrow;
    except
        // If execution reaches this point, then an exception has occurred. We cannot
        // know how to handle all possible exceptions, so we merely clean up the resources
        // allocated by this function and then re-raise the exception so the caller can
        // choose what to do with it.
        Result.Free;
        raise;
    end;
```

```
// If execution reaches this point, then no exception has occurred, so the
// function will return Result normally.
end;
```

ナイーブなプログラマは、すべてのタイプをキャッチしてそのようなから `nil` をそうとするかもしれませんが、それはすべてのタイプをせずにすべてのタイプをキャッチするという、

してみてください。

`try - finally` ブロックは、`try - except` ブロックのにネストすることができます。

```
try
  AcquireResources;
  try
    UseResource;
  finally
    ReleaseResource;
  end;
except
  on E: EResourceUsageError do begin
    HandleResourceErrors;
  end;
end;
```

`UseResource` でがすると、は `ReleaseResource` ジャンプします。が `EResourceUsageError`、はハンドラにジャンプし、`HandleResourceErrors` をびし `HandleResourceErrors`。ののはのハンドラをスキップし、の `try` バブルアップします - びしスタック `except` ブロックする `except` ます。

`AcquireResource` または `ReleaseResource` をスキップして、ハンドラにくことになります `finally` ブロックする、のいずれかにするため `try` ブロックは、まだされていないかので、`finally` ブロックがすでにされています。

してみる

`try - except` ブロックは、`try - finally` ブロックにネストすることができます。

```
AcquireResource;
try
  UseResource1;
  try
    UseResource2;
  except
    on E: EResourceUsageError do begin
      HandleResourceErrors;
    end;
  end;
  UseResource3;
finally
  ReleaseResource;
end;
```

`EResourceUsageError` で `UseResource2`、はハンドラにジャンプしてびします `HandleResourceError`。は

されたとみなされるため、は `UseResource3` にいて `ReleaseResource` きます。

`UseResource2` でののがした `UseResource2`、ここにされるハンドラはされません。したがって、は `UseResource3` びしを `UseResource3`、`ReleaseResource` がびされる `finally` ブロックにします。その、がコールスタックをバブルアップするので、はのなハンドラにジャンプします。

のでのびしでがした、`HandleResourceErrors` はびされません。これは、そのハンドラにする `try` ブロックでのびしがしないためです。

2つのオブジェクトをしてみてください

```
Object1 := nil;
Object2 := nil;
try
  Object1 := TMyObject.Create;
  Object2 := TMyObject.Create;
finally
  Object1.Free;
  Object2.Free;
end;
```

`try-finally` ブロックので `nil` でオブジェクトをしない `try-finally`、そのうちの1つがされなかった、`AV` は `finally` ブロックでします。なぜなら、オブジェクトはされていないのでがします。

`Free` メソッドは、オブジェクトが `nil` であるかどうかをチェックするので、のオブジェクトを `nil` すると、されていなければするにエラーがされます。

オンラインで `try`、`except`、および `finally` のをむ <https://riptutorial.com/ja/delphi/topic/3055/try-except-およびfinallyの>

7: TStringList クラス

Examples

き

TStringListは、VCLのTStringsクラスです。TStringListは、のリストのとにできます。はをとしていましたが、このクラスをしてのタイプのオブジェクトをすることもできます。

TStringListは、のリストをするがあるに、VCLでくされています。TStringListは、なカスタマイズとのさをするなメソッドセットをサポートしています。

のは、TStringListオブジェクトの、の、べえ、およびをしています。

```
procedure StringListDemo;
var
  MyStringList: TStringList;
  i: Integer;

Begin

  //Create the object
  MyStringList := TStringList.Create();
  try
    //Add items
    MyStringList.Add('Zebra');
    MyStringList.Add('Elephant');
    MyStringList.Add('Tiger');

    //Sort in the ascending order
    MyStringList.Sort;

    //Output
    for i:=0 to MyStringList.Count - 1 do
      WriteLn(MyStringList[i]);
    finally
      //Destroy the object
      MyStringList.Free;
    end;
  end;
end;
```

TStringListには、の、ソート、け、キーとのペアリング、りのりなど、さまざまなユーザーのケースがあります。

Key-Value ペアリング

TStringListをすると、Key-Valueのペアをできます。これは、たとえばをするにです。は、キーのとでされます。キーとのペアは、Key = ValueのStringListの1にされます。

```
procedure Demo(const FileName: string = '');
```

```

var
  SL: TStringList;
  i: Integer;
begin
  SL:= TStringList.Create;
  try
    //Adding a Key-Value pair can be done this way
    SL.Values['FirstName']:= 'John';    //Key is 'FirstName', Value is 'John'
    SL.Values['LastName']:= 'Doe';    //Key is 'LastName', Value is 'Doe'

    //or this way
    SL.Add('City=Berlin');    //Key is 'City', Value is 'Berlin'

    //you can get the key of a given Index
    IF SL.Names[0] = 'FirstName' THEN
      begin
        //and change the key at an index
        SL.Names[0]:= '1stName';    //Key is now "1stName", Value remains "John"
      end;

      //you can get the value of a key
      s:= SL.Values['City']; //s now is set to 'Berlin'

      //and overwrite a value
      SL.Values['City']:= 'New York';

      //if desired, it can be saved to an file
      IF (FileName <> '') THEN
        begin
          SL.SaveToFile(FileName);
        end;
      finally
        SL.Free;
      end;
    end;
  end;
end;

```

このでは、StringListはされるにのっています。

```

1stName=John
LastName=Doe
City=New York

```

パフォーマンスにする

フードのでは、TStringListは、すべてのアイテムをにループし、アイテムのセパレータをし、のをされたキーとすることによってキーをします。パフォーマンスにきなをぼしているとはいいいので、このメカニズムはではなく、まれにりされるでのみするがあります。パフォーマンスのには、つはうべきTDictionary<TKey, TValue>からSystem.Generics.CollectionsハッシュテーブルのやソートのキーつためにTStringListとしてされたとObjectこのようにバイナリアルゴリズムを-sを。

オンラインでTStringListクラスをむ <https://riptutorial.com/ja/delphi/topic/6045/tstringlist> クラス

8: インターフェイス

インタフェースは、なのをすることなく、メソッドやクラスのなとされるをするためにされます。

クラスはインターフェイスをでき、インターフェイスはいにできます。クラスがインタフェースをしている、インタフェースでされているすべてのとプロシージャがクラスにすることをします。

デルファイのインタフェースのなは、インタフェースのインスタンスがカウントについたライフタイムをつことです。クラスインスタンスのはでするがあります。

これらのすべてののをすると、インタフェースをしてさまざまなをできます。

- のためののなるをするえば、ファイル、データベースへの、またはメールとしての、すべてをインターフェイス "SaveData"として
- をらし、デカップリングをし、コードをよりとテストにする
- にまされることなく、のユニットでインスタンスをしますただし、ここでもとしがしますが、してください。

Examples

インターフェイスのと

インタフェースはクラスのようにされていますが、アクセスはありません `public`、`private`、...。また、はされていないので、とはできません。

インターフェイスにはのがにです。これは、`Ctrl + Shift + G`をしてできます。

```
IRepository = interface
    ['{AFCFCE96-2EC2-4AE4-8E23-D4C4FF6BBD01}']
    function SaveKeyValuePair(aKey: Integer; aValue: string): Boolean;
end;
```

インタフェースをするには、インタフェースのをクラスのろにするがあります。また、クラスは `TInterfacedObject` であるがあり `TInterfacedObject` これはにとってです。

```
TDatabaseRepository = class(TInterfacedObject, IRepository)
    function SaveKeyValuePair(aKey: Integer; aValue: string): Boolean;
end;
```

クラスがインタフェースをするときは、インタフェースにされているすべてのメソッドとをむがあります。そうでなければ、コンパイルされません。

すべきの1つは、びしがインタフェースでする、アクセスはをえないということです。えば、イン

タフエースのすべてののは`strict private`メンバとしてできますが、インタフエースのインスタンスがされているは、のクラスからききびすことができます。

のインタフエースの

クラスは、**Delphi**クラスではなのクラス からするのではなく、 のインタフエースをできます。これをするには、すべてのインタフエースのをクラスのろにコンマリでするがあります。

もちろん、クラスは、インタフエースでされたもしなければなりません。

```
IInterface1 = interface
    ['{A2437023-7606-4551-8D5A-1709212254AF}']
    procedure Method1();
    function Method2(): Boolean;
end;

IInterface2 = interface
    ['{6C47FF48-3943-4B53-8D5D-537F4A0DEC0D}']
    procedure SetValue(const aValue: TObject);
    function GetValue(): TObject;

    property Value: TObject read GetValue write SetValue;
end;

TImplementer = class(TInterfacedObject, IInterface1, IInterface2)
    // IInterface1
    procedure Method1();
    function Method2(): Boolean;

    // IInterface2
    procedure SetValue(const aValue: TObject);
    function GetValue(): TObject

    property Value: TObject read GetValue write SetValue;
end;
```

インタフエースの

インタフエースは、クラスとまったくじように、いにすることができます。したがって、クラスは、インタフエースおよびすべてのインタフエースのをするがあります。しかし、このでは、コンパイラーは`implenting`クラスがインターフエースもしていることをらず、にリストされているインターフエースのみをします。そのため`as ISuperInterface`で`TImplementer`をして`as ISuperInterface`しません。これにより、すべてのインターフエースもにすることになりますこの、`TImplementer = class(TInterfacedObject, IDescendantInterface, ISuperInterface)。`

```
ISuperInterface = interface
    ['{A2437023-7606-4551-8D5A-1709212254AF}']
    procedure Method1();
    function Method2(): Boolean;
end;

IDescendantInterface = interface(ISuperInterface)
    ['{6C47FF48-3943-4B53-8D5D-537F4A0DEC0D}']
```

```

procedure SetValue(const aValue: TObject);
function  GetValue(): TObject;

property Value: TObject read GetValue write SetValue;
end;

TImplementer = class(TInterfacedObject, IDescendantInterface)
// ISuperInterface
procedure Method1();
function Method2(): Boolean;

// IDescendantInterface
procedure SetValue(const aValue: TObject);
function  GetValue(): TObject

property Value: TObject read GetValue write SetValue;
end;

```

インタフェースのプロパティ

インタフェースのものはであるため、`property Value: TObject read FValue write FValue;`をする「」な `property Value: TObject read FValue write FValue;` はできません。わりに、Getterとsetterそれぞれのみをインタフェースでするがあります。

```

IInterface = interface(IInterface)
['{6C47FF48-3943-4B53-8D5D-537F4A0DEC0D}']
procedure SetValue(const aValue: TObject);
function  GetValue(): TObject;

property Value: TObject read GetValue write SetValue;
end;

```

すべきの1つは、クラスがプロパティをするがないということです。コンパイラはこのコードをくれます

```

TImplementer = class(TInterfacedObject, IInterface)
procedure SetValue(const aValue: TObject);
function  GetValue(): TObject
end;

```

しかし、1つのは、このようなでは、クラスそのものではなく、インタフェースのインスタンスをじてのみプロパティにアクセスできることです。また、プロパティをクラスにするとみやすさがします。

オンラインでインターフェイスをむ <https://riptutorial.com/ja/delphi/topic/4885/インターフェイス>

9: ジェネリックス

Examples

TArray.Sortをしてをソートする

```
uses
  System.Generics.Collections, { TArray }
  System.Generics.Defaults; { TComparer<T> }

var StringArray: TArray<string>; { Also works with "array of string" }

...

{ Sorts the array case insensitive }
TArray.Sort<string>(StringArray, TComparer<string>.Construct(
  function (const A, B: string): Integer
  begin
    Result := string.CompareText(A, B);
  end
));
```

TListのない

```
var List: TList<Integer>;

...

List := TList<Integer>.Create; { Create List }
try
  List.Add(100); { Add Items }
  List.Add(200);

  WriteLn(List[1]); { 200 }
finally
  List.Free;
end;
```

TListからそれをにする

```
type
  TIntegerList = class(TList<Integer>)
  public
    function Sum: Integer;
  end;

...

function TIntegerList.Sum: Integer;
var
  Item: Integer;
begin
```

```
Result := 0;
for Item in Self do
    Result := Result + Item;
end;
```

TListをソートする

```
var List: TList<TDateTime>;

...

List.Sort(
    TComparer<TDateTime>.Construct(
        function(const A, B: TDateTime): Integer
        begin
            Result := CompareDateTime(A, B);
        end
    )
);
```

オンラインでジェネリックスをむ <https://riptutorial.com/ja/delphi/topic/4054/ジェネリックス>

10: バックグラウンドスレッドでされた TDataSet データをする

このFireDACのと、がしようとしているのものは、にデータセットをくためのネイティブコールのをけます。

Examples

FireDACの

のコードサンプルは、FireDACをしてバックグラウンドスレッドのMSSqlサーバーからレコードをするの1つをしています。 Delphi 10 Seattleのテストみ

いたとおり

- スレッドはのTFDConnectionとTFDQueryをしてデータをし、SynchronizeのびしでフォームのFDQueryにデータをします。
- Executeは、データを1だけします。 VCLスレッドからされたメッセージにしてクエリをしするようにすることができます。

コード

```
type
  TForm1 = class;

  TFDQueryThread = class(TThread)
  private
    FConnection: TFDConnection;
    FQuery: TFDQuery;
    FForm: TForm1;
  published
    constructor Create(AForm : TForm1);
    destructor Destroy; override;
    procedure Execute; override;
    procedure TransferData;
    property Query : TFDQuery read FQuery;
    property Connection : TFDConnection read FConnection;
    property Form : TForm1 read FForm;
  end;

  TForm1 = class(TForm)
    FConnection1: TFDConnection;
    FQuery1: TFDQuery;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    DBNavigator1: TDBNavigator;
    Button1: TButton;
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
```

```

    procedure FormCreate(Sender: TObject);
private
public
    QueryThread : TFDQueryThread;
end;

var
Form1: TForm1;

implementation

{$R *.dfm}

{ TFDQueryThread }

constructor TFDQueryThread.Create(AForm : TForm1);
begin
    inherited Create(True);
    FreeOnTerminate := False;
    FForm := AForm;
    FConnection := TFDConnection.Create(nil);
    FConnection.Params.Assign(Form.FDConnection1.Params);
    FConnection.LoginPrompt := False;

    FQuery := TFDQuery.Create(nil);
    FQuery.Connection := Connection;
    FQuery.SQL.Text := Form.FDQuery1.SQL.Text;
end;

destructor TFDQueryThread.Destroy;
begin
    FQuery.Free;
    FConnection.Free;
    inherited;
end;

procedure TFDQueryThread.Execute;
begin
    Query.Open;
    Synchronize(TransferData);
end;

procedure TFDQueryThread.TransferData;
begin
    Form.FDQuery1.DisableControls;
    try
        if Form.FDQuery1.Active then
            Form.FDQuery1.Close;
        Form.FDQuery1.Data := Query.Data;
    finally
        Form.FDQuery1.EnableControls;
    end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    QueryThread.Free;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin

```

```
if not QueryThread.Finished then
  QueryThread.Start
else
  ShowMessage('Thread already executed!');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FDQuery1.Open;
  QueryThread := TFDQueryThread.Create(Self);
end;

end.
```

オンラインでバックグラウンドスレッドでされたTDataSetデータをするをむ

<https://riptutorial.com/ja/delphi/topic/4114/バックグラウンドスレッドでされたtdatasetデータをする>

11: ループ

き

Delphiは3つのタイプのループをします

`for` - 、 、 またはのシーケンスのイテレータ

`repeat-until` - `quit`はターンののにチェックし、ループはでも1する `tmeeven`

`while do` - `do`はターンののにチェックしていますが、ループはしてできません

- `OrdinalVariable = LowerOrdinalValue`を`UpperOrdinalValue`にする `do begin {loop-body} end;`
- `OrdinalVariable = UpperOrdinalValue` `downto LowerOrdinalValue` `do begin {ループボディ} end;`
- コレクションの`EnumerableVariable`にして `do begin {loop-body} end;`
- `{break-condition}`まで`{loop-body}`をりします。
- `while {condition} do begin {ループボディ} end;`

Examples

ループでしける

```
program ForLoopWithContinueAndBreaks;

{$APPTYPE CONSOLE}

var
  i : integer;
begin
  for i := 1 to 10 do
    begin
      if i = 2 then continue; (* Skip this turn *)
      if i = 8 then break;    (* Break the loop *)
      WriteLn( i );
    end;
  WriteLn('Finish. ');
end.
```

1
3
4
5
6
7
。

りし - Until

```
program repeat_test;

{$APPTYPE CONSOLE}

var s : string;
begin
  WriteLn( 'Type a words to echo. Enter an empty string to exit.' );
  repeat
    ReadLn( s );
    WriteLn( s );
  until s = '';
end.
```

コンソールのこのいのType a words to echo. Enter an empty string to exit.ユーザのタイプをとって、それをエコーし、ループでびをつ - ユーザがををするまで。

しばらく

```
program WhileEOF;
{$APPTYPE CONSOLE}
uses SysUtils;

const cFileName = 'WhileEOF.dpr';
var F : TextFile;
s : string;
begin
  if FileExists( cFileName )
  then
    begin
      AssignFile( F, cFileName );
      Reset( F );

      while not Eof(F) do
      begin
        ReadLn(F, s);
        WriteLn(s);
      end;

      CloseFile( F );
    end
  else
    WriteLn( 'File ' + cFileName + ' not found!' );
  end.
```

このでは、While not (EOF) をして、WhileEOF.dpr ファイルのテキストをコンソールにWhileEOF.dpr ます。fileがの、ReadLn-WriteLn ループはされません。

オンラインでループをむ <https://riptutorial.com/ja/delphi/topic/9931/ループ>

12: のプログラムの

Examples

CreateProcess

のは、 `CreateProcess` Windows APIをしてのプログラムをするコードをカプセルします。

これはで、びしプロセスがするか、すぐになるまでつことができます。

パラメーター

- `FileName` - ファイルへのフルパス
- `Params` - コマンドラインパラメータまたはのをする
- `Folder` - びされたプログラムのフォルダ - のパスが`FileName`からされる
- `WaitUntilTerminated` - `true`の、はプロセスのがするまでする
- `WaitUntilIdle` - `true`の、は`WaitForInputIdle`をびし、されたプロセスがのをし、ユーザーがされなくなるまでする
- `RunMinimized` - のプロセスがされてされる
- `ErrorCode` - がすると、これにはWindowsエラーコードがまれます

```
function ExecuteProcess(const FileName, Params: string; Folder: string; WaitUntilTerminated,
WaitUntilIdle, RunMinimized: boolean;
var ErrorCode: integer): boolean;
var
  CmdLine: string;
  WorkingDirP: PChar;
  StartupInfo: TStartupInfo;
  ProcessInfo: TProcessInformation;
begin
  Result := true;
  CmdLine := '"' + FileName + ' ' + Params;
  if Folder = '' then Folder := ExcludeTrailingPathDelimiter(ExtractFilePath(FileName));
  ZeroMemory(@StartupInfo, SizeOf(StartupInfo));
  StartupInfo.cb := SizeOf(StartupInfo);
  if RunMinimized then
    begin
      StartupInfo.dwFlags := STARTF_USESHOWWINDOW;
      StartupInfo.wShowWindow := SW_SHOWMINIMIZED;
    end;
  if Folder <> '' then WorkingDirP := PChar(Folder)
  else WorkingDirP := nil;
  if not CreateProcess(nil, PChar(CmdLine), nil, nil, false, 0, nil, WorkingDirP, StartupInfo,
ProcessInfo) then
    begin
      Result := false;
      ErrorCode := GetLastError;
      exit;
    end;
  with ProcessInfo do
    begin
```

```

    CloseHandle(hThread);
    if WaitUntilIdle then WaitForInputIdle(hProcess, INFINITE);
    if WaitUntilTerminated then
        repeat
            Application.ProcessMessages;
            until MsgWaitForMultipleObjects(1, hProcess, false, INFINITE, QS_ALLINPUT) <>
WAIT_OBJECT_0 + 1;
        CloseHandle(hProcess);
    end;
end;

```

のの

```

var
    FileName, Parameters, WorkingFolder: string;
    Error: integer;
    OK: boolean;
begin
    FileName := 'C:\FullPath\myapp.exe';
    WorkingFolder := ''; // if empty function will extract path from FileName
    Parameters := '-p'; // can be empty
    OK := ExecuteProcess(FileName, Parameters, WorkingFolder, false, false, false, Error);
    if not OK then ShowMessage('Error: ' + IntToStr(Error));
end;

```

CreateProcessドキュメント

オンラインでのプログラムのをむ <https://riptutorial.com/ja/delphi/topic/5180/プログラムの>

13:

Examples

Delphiにはのがあります。

タイプ		サイ ズ	
string	2GB	16バ イト	された。 Delphi 2009をしたAnsiStringのエイリアスと、 Delphi 2009のUnicodeStringエイリアスです。
UnicodeString	2GB	16バ イト	UTF-16のされた。
AnsiString	2GB	16バ イト	ユニコードのANSIのされた。 Delphi 2009では、なコードペ ージインジケータをしています。
UTF8String	2GB	16バ イト	UTF-8のされたAnsiString -8コードページをむAnsiStringとし てされます。
ShortString	255	2バイ ト	オーバーヘッドがほとんどないののアンマネージ
WideString	2GB	4バイ ト	UTF-16のされたであるCOMをととしています。 Windows BSTR にします。

UnicodeStringとAnsiStringはカウントとCOW Copy- On -Write です。

ShortStringとWideStringはカウントではなく、COWセマンティクスをちません。

```
uses
  System.Character;

var
  S1, S2: string;
begin
  S1 := 'Foo';
  S2 := ToLower(S1); // Convert the string to lower-case
  S1 := ToUpper(S2); // Convert the string to upper-case
```

チャーズ

2009

```
uses
  Character;
```

```

var
  C1, C2: Char;
begin
  C1 := 'F';
  C2 := ToLower(C1); // Convert the char to lower-case
  C1 := ToUpper(C2); // Convert the char to upper-case

```

バージョンがXE2の、`uses`は`System.Character`なければなりません。

と

```

uses
  SysUtils;

var
  S1, S2: string;
begin
  S1 := 'Foo';
  S2 := LowerCase(S1); // S2 := 'foo';
  S1 := UpperCase(S2); // S1 := 'FOO';

```

りて

なるへののりてと、それらのにするのメモリりて、カウント、へのインデックスきアクセス、およびするはにされたコンパイラエラー。

```

var
  SS5: string[5]; {a shortstring of 5 chars + 1 length byte, no trailing `0`}
  WS: Widestring; {managed pointer, with a bit of compiler support}
  AS: ansistring; {ansistring with the default codepage of the system}
  US: unicodestring; {default string type}
  U8: UTF8string; {same as AnsiString(65001)}
  A1251: ansistring(1251); {ansistring with codepage 1251: Cyrillic set}
  RB: RawbyteString; {ansistring with codepage 0: no conversion set}
begin
  SS5:= 'test'; {S[0] = Length(SS254) = 4, S[1] = 't'...S[5] = undefined}
  SS5:= 'test1'; {S[0] = 5, S[5] = '1', S[6] is out of bounds}
  SS5:= 'test12'; {compile time error}
  WS:= 'test'; {WS now points to a constant unicodestring hard compiled into the data segment}
  US:= 'test'+IntToStr(1); {New unicode string is created with reference count = 1}
  WS:= US; {SysAllocateStr with data copied to dest, US refcount = 1 !}
  AS:= US; {the UTF16 in US is converted to "extended" ascii taking into account the codepage
in AS possibly losing data in the process}
  U8:= US; {safe copy of US to U8, all data is converted from UTF16 into UTF8}
  RB:= US; {RB = 'test1'#0 i.e. conversion into RawByteString uses system default codepage}
  A1251:= RB; {no conversion takes place, only reference copied. Ref count incremented }

```

カウント

のをえることはスレッドセーフです。プロセスをするために、ロックとハンドラがされます。コンパイラがカウントをするためにコンパイルにコードをするをすコメントとともに、のコードをしてください。

```

procedure PassWithNoModifier(S: string);
// prologue: Increase reference count of S (if non-negative),
//           and enter a try-finally block
begin
    // Create a new string to hold the contents of S and 'X'. Assign the new string to S,
    // thereby reducing the reference count of the string S originally pointed to and
    // bringing the reference count of the new string to 1.
    // The string that S originally referred to is not modified.
    S := S + 'X';
end;
// epilogue: Enter the `finally` section and decrease the reference count of S, which is
//           now the new string. That count will be zero, so the new string will be freed.

procedure PassWithConst(const S: string);
var
    TempStr: string;
// prologue: Clear TempStr and enter a try-finally block. No modification of the reference
//           count of string referred to by S.
begin
    // Compile-time error: S is const.
    S := S + 'X';
    // Create a new string to hold the contents of S and 'X'. TempStr gets a reference count
    // of 1, and reference count of S remains unchanged.
    TempStr := S + 'X';
end;
// epilogue: Enter the `finally` section and decrease the reference count of TempStr,
//           freeing TempStr because its reference count will be zero.

```

にしたように、パラメータにをするためになローカルをすることは、そのパラメータにをえるのとじオーバヘッドをいます。const するのは、パラメータがにみりであるにのみカウントをけることです。ただし、のにのがれないようにするには、にconst、var、またはout いずれかをパラメータにすることをおめします。

エンコーディング

UnicodeString、AnsiString、WideString、UTF8Stringなどのは、それぞれのエンコーディングをしてメモリにされますはを。あるタイプののをのタイプののにりてると、がわれるがあります。はエンコーディングにしないようにされています。はしてしないでください。

クラスSysutils.TEncoding メソッド GetBytes するためのstring に TBytes バイトと GetString するための TBytes するstring。クラスSysutils.TEncoding は、クラスプロパティとしてくのされたエンコーディングをします。

エンコーディングをうの1つは、アプリケーションでstringのみをし、のエンコーディングはI/O、DLLびしなどをするがあるたびにTEncodingをすることです。

```

procedure EncodingExample;
var hello,response:string;
    dataout,datain:TBytes;
    expectedLength:integer;
    stringStream:TStringStream;
    stringList:TStringList;

begin

```

```

hello := 'Hello World!Привет мир!';
dataout := SysUtils.TEncoding.UTF8.GetBytes(hello); //Conversion to UTF8
datain := SomeIOFunction(dataout); //This function expects input as TBytes in UTF8 and
returns output as UTF8 encoded TBytes.
response := SysUtils.TEncoding.UTF8.GetString(datain); //Conversion from UTF8

//In case you need to send text via pointer and length using specific encoding (used mostly
for DLL calls)
dataout := SysUtils.TEncoding.GetEncoding('ISO-8859-2').GetBytes(hello); //Conversion to ISO
8859-2
DLLCall(addr(dataout[0]),length(dataout));
//The same is for cases when you get text via pointer and length
expectedLength := DLLCallToGetDataLength();
setLength(datain,expectedLength);
DLLCall(addr(datain[0]),length(datain));
response := SysUtils.TEncoding.GetEncoding(1250).GetString(datain);

//TStringStream and TStringList can use encoding for I/O operations
stringList:TStringList.create;
stringList.text := hello;
stringList.saveToFile('file.txt',SysUtils.TEncoding.Unicode);
stringList.destroy;
stringStream := TStringStream(hello,SysUtils.TEncoding.Unicode);
stringStream.saveToFile('file2.txt');
stringStream.Destroy;
end;

```

オンラインでをむ <https://riptutorial.com/ja/delphi/topic/3957/>

14:

Examples

Windows API GetTickCountをする

Windows APIのGetTickCountは、システムコンピュータがされてからのミリをします。もなはのとおります。

```
var
  Start, Stop, ElapsedMilliseconds: cardinal;
begin
  Start := GetTickCount;
  // do something that requires measurement
  Stop := GetTickCount;
  ElapsedMilliseconds := Stop - Start;
end;
```

GetTickCountは32ビットのDWORDし、GetTickCount.7ごとにりします。りしをけるには、GetTickCount64 Windows Vistaでまたはなルーチンをしてティックのをします。

```
function TickDiff(StartTick, EndTick: DWORD): DWORD;
begin
  if EndTick >= StartTick
  then Result := EndTick - StartTick
  else Result := High(NativeUInt) - StartTick + EndTick;
end;

function TicksSince(Tick: DWORD): DWORD;
begin
  Result := TickDiff(Tick, GetTickCount);
end;
```

とにかく、これらのルーチンは、GetTickCount 2のびしのがGetTickCount.7のをえると、なをします。

ミリをにする

```
var
  Start, Stop, ElapsedMilliseconds: cardinal;
begin
  Start := GetTickCount;
  sleep(4000); // sleep for 4 seconds
  Stop := GetTickCount;
  ElapsedMilliseconds := Stop - Start;
  ShowMessage('Total Seconds: '
    + IntToStr(round(ElapsedMilliseconds/SysUtils.MSecsPerSec))); // 4 seconds
end;
```

TStopwatchレコードをする

Delphiのバージョンには、のTStopwatchレコードがしています。

```
uses
  System.Diagnostics;

var
  StopWatch: TStopwatch;
  ElapsedMilliseconds: Int64;
begin
  StopWatch := TStopwatch.StartNew;
  // do something that requires measurement
  ElapsedMilliseconds := StopWatch.ElapsedMilliseconds;
end;
```

オンラインでをむ <https://riptutorial.com/ja/delphi/topic/2425/>

15: にりしなランタイムエラーチェックをする

き

これは、のったエラーチェックルーチンをどのようにしてにみむことができるのかをしています。これにより、コードオフにコードオーバーヘッドがしません。

Examples

な

```
{ $DEFINE MyRuntimeCheck } // Comment out this directive when the check is no-longer required!
                           // You can also put MyRuntimeCheck in the project defines instead.

function MyRuntimeCheck: Boolean; { $IFDEF MyRuntimeCheck } inline; { $ENDIF }
begin
    result := TRUE;
    { $IFDEF MyRuntimeCheck }
        // .. the code for your check goes here
    { $ENDIF }
end;
```

はにこれです

されたシンボルは、コードのをにするためにされます。また、にインラインされているコードをします。つまり、チェックルーチンにブレークポイントをするがです。

しかし、このののしさは、もうチェックしたくないときです。 **\$DEFINE** そのに `//` をくをコメントアウトすることによって、チェックコードをするだけでなく、ルーチンのインラインをオンにして、びされたすべてのからオーバーヘッドをりきますルーチンコンパイラは、チェックのすべてのトレースをにしますもちろん、インラインは "On" または "Auto" にされているものとしします。

のはに "アサーション" のにています。のは、にじてを TRUE または FALSE にできます。

しかし、トレースロギング、メトリックなどをうコードにこのようなをにできるようになりました。例えば

```
procedure MyTrace(const what: string); { $IFDEF MyTrace } inline; { $ENDIF }
begin
    { $IFDEF MyTrace }
        // .. the code for your trace-logging goes here
    { $ENDIF }
end;

...
MyTrace('I was here'); // This code overhead will vanish if 'MyTrace' is not defined.
MyTrace( SomeString ); // So will this.
```

オンラインでにりしなランタイムエラーチェックをするをむ

<https://riptutorial.com/ja/delphi/topic/10541/>にりしなランタイムエラーチェックをする

クレジット

S. No		Contributors
1	Embarcadero Delphi	Charlie H , Community , Dalija Prasnikar , Florian Koch , Jeroen Wiert Pluimers , René Hoffmann , RepeatUntil , Rob Kennedy , Vadim Shakun , w5m , Y.N , Zam
2	DelphiでのRTTIの	Petzy , René Hoffmann
3	Firemonkeyでのアニメーションの	Alexander Petrosyan
4	For Loops	Filipe Martins , Jeroen Wiert Pluimers , John Easley , René Hoffmann , Rob Kennedy , Siendor , Y.N
5	GUIのをしながらスレッドをする	Fr0sT , Jerry Dodge , Johan , kami , LU RD , Marjan Venema
6	try、except、およびfinallyの	EMBarbosa , Fabio Gomes , Johan , MrE , Nick Hodges , Rob Kennedy , Shadow
7	TStringListクラス	Charlie H , Fabricio Araujo , Fr0sT , KaiW
8	インターフェイス	Florian Koch , Willo van der Merwe
9	ジェネリックス	Rob Kennedy , Steffen Binas , Uli Gerhardt
10	バックグラウンドスレッドでされたTDataSetデータをす	MartynA
11	ループ	Y.N
12	のプログラムの	Dalija Prasnikar
13		AleKXL , Dalija Prasnikar , EMBarbosa , Fabricio Araujo , Johan , Radek Hladík , René Hoffmann , RepeatUntil , Rob Kennedy , Rudy Velthuis
14		Fr0sT , John Easley , kludg , Rob Kennedy , Victoria , Wolf
15	にりしなランタイムエラーチェックをす	Alex T

	る	
--	---	--