

# Mengukur Waktu Tayang Form

**Bayu Prasetyo**

*bprasetyo@gmail.com*

*http://www.bprasetyo.or.id*

## ***Lisensi Dokumen:***

*Copyright © 2003-2008 IlmuKomputer.Com*

*Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarkan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.*

Salah satu syarat suatu aplikasi dikatakan tangguh adalah apabila aplikasi sangat responsif terhadap interaksi dengan pengguna. Salah satu interaksi yang dilakukan oleh aplikasi adalah penayangan suatu jendela (form).

Permasalahan yang sering timbul adalah, bahwa kita tidak dapat mengukur waktu tayang suatu form dengan pasti, yang dapat dilakukan adalah berdasarkan perkiraan, misalnya, penayangan form tampak lebih cepat dan tanpa kedipan (*flicker*). Namun kemudian jika diajukan pertanyaan seperti: “Berapa selisih kecepatannya, apakah signifikan dan apa buktinya?”. Kita tidak dapat menyertakan bukti yang kuat, yaitu metode pengukuran yang dilakukan serta hasil pengukuran yang terukur.

Untuk itulah, dalam kesempatan ini, akan dibahas bagaimana membuat pengukur lama proses (*profiler*) sederhana untuk mengukur waktu tayang (*load time*) suatu form dari form tersebut di buat hingga benar - benar tampil dan siap digunakan. Tentunya dengan menggunakan Delphi.

## **1. Siklus Hidup Form**

Sebelum melangkah lebih jauh, perlu diketahui terlebih dahulu siklus hidup suatu form, dari form tersebut dibuat, ditampilkan hingga ditutup dan dibebaskan dari memori. Berikut penjelasan singkat / ringkasan mengenai siklus hidup form:

### **1. Pembuatan Form (*Form Creation*)**

Terdapat 2 (dua) cara pembuatan form, yaitu melalui pemanggilan konstruktor `Create`, misalnya

```
AForm := TForm.Create(nil);
```

dan melalui obyek `Application.CreateForm`. Ketika suatu form dibuat, beberapa instruksi inisialisasi dijalankan.

Jika form yang dimaksud sudah dirancang pada saat disain (*design-time*), yang ditandai dengan disimpan pada *resourcefile* yaitu DFM, maka kode – kode yang terdapat pada *resourcefile* tersebut dibaca (*parse*) dan diterjemahkan menjadi informasi / keterangan suatu obyek. Langkah selanjutnya adalah pembuatan obyek dan pemberian properti dan *event* yang diperlukan berdasarkan informasi obyek yang telah diterjemahkan tersebut.

Jika form yang dimaksud tidak dirancang pada saat disain, dengan kata lain form tersebut dibuat pada saat itu juga (*on the fly / run time creation*), maka pembuatan obyek dan pengaturan properti dan *event* form tersebut berdasarkan nilai default yang telah ditentukan. Pemberian nilai properti dan *event* selanjutnya diserahkan ke pemrogram.

Setelah pembuatan obyek dan inisialisasi terhadap form selesai dikerjakan, method *DoCreate* dijalankan. Salah satu isi method *DoCreate* adalah menjalankan *event* *OnCreate*, jika *event* tersebut tersedia.

## 2. Penayangan Form (*Form Showing*)

Setelah form selesai dibuat, jika terdapat instruksi untuk menampilkan form, baik itu melalui properti *Visible := True* atau *method Show* atau *method ShowModal*, maka form akan tertayang di layar. Penayangan form tidak lepas dari penggambaran form, yaitu dijalankannya method *Paint* yang bertugas menjalankan *event* *OnPaint* jika tersedia. Selain itu juga dijalankan *method DoShow* yang akan menjalankan instruksi yang diberikan pada *event* *OnShow*, jika tersedia.

## 3. Aktivasi Form (*Form Activation*)

Setelah form ditayangkan, langkah selanjutnya adalah aktivasi form tersebut, yaitu pemberian fokus ke form tersebut sehingga form yang dimaksud tampil di atas jendela yang lain dan dalam posisi aktif siap menerima respon dari pengguna.

Ada beberapa *method* yang dijalankan pada saat aktivasi form, yaitu *Resizing*, *Paint* dan *Activate*. *Method Activate* mengaktifkan semua kontrol yang ada didalam form tersebut yaitu dengan mengirimkan pesan *CM\_ACTIVATE*. Selain itu *method Activate* akan menjalankan instruksi yang diberikan pada *event* *OnActivate*, jika tersedia.

## 4. Penggunaan Form (*Form Usage*)

Pada tahap ini form bertanggung jawab terhadap interaksi pengguna berdasarkan kode yang telah dirancang. Banyak sekali *event* dan *method* yang dijalankan, baik itu yang telah disediakan maupun *method / event* buatan pemrogram.

## 5. Konfirmasi Penutupan Form (*Form Close Query*)

Ketika form akan ditutup, yaitu ketika menerima method *Close* atau *CloseModal*, *method CloseQuery* dijalankan, salah satu instruksinya adalah menjalankan *event* *OnCloseQuery* jika tersedia.

## 6. Penutupan Form (*Form Closing*)

Jika konfirmasi penutupan form menghasilkan nilai *True*, yaitu instruksi agar form

ditutup, maka *method* `DoClose` akan dijalankan. Salah satu isinya adalah menjalankan *event* `OnClose` jika tersedia.

#### 7. Penyembunyian Form (*Form Hiding*)

Tahapan selanjutnya adalah penyembunyian form dari layar. Tahapan ini ditandai dengan deaktivasi form dan kontrol yang ada didalamnya melalui *method* `Deactivate` atau pesan `CM_DEACTIVATE`. Selanjutnya form disembunyikan dengan memberikan properti `Visible := False` atau menjalankan *method* `DoHide` yang isinya menjalankan *event* `OnHide` jika tersedia.

#### 8. Penghancuran Form (*Form Destruction*)

Setelah form disembunyikan, langkah selanjutnya adalah menghancurkan form. Tahapan ini ditandai dengan pemberian perintah `Free` yang menjalankan *method* `DoDestroy`. *Method* ini bertugas menjalankan *event* `OnDestroy` jika tersedia.

Pada tahap ini semua alokasi obyek dan memori baik yang menjadi tanggung jawab sistem maupun tanggung jawab pemrogram dibebaskan agar memori yang tersedia dapat digunakan oleh aplikasi lain yang lebih membutuhkan.

## 2. Metode Pengukuran

Setelah mengetahui siklus hidup form, dari pembuatan, penayangan, penggunaan hingga penghancuran, langkah selanjutnya adalah mengetahui metode pengukuran yang akan digunakan.

### Memulai Pengukuran

Kode pengukuran disisipkan pada saat pemanggilan konstruktor suatu form. Yaitu di dalam *method* `create`. Pada awal baris, sisipkan kode untuk menghitung waktu mulai proses, yaitu dengan menggunakan fungsi `GetTickCount()` yang hasilnya disimpan pada suatu variabel, sebagai contoh adalah `FStart` yang bertipe `Cardinal`.

### Mengakhiri Pengukuran

Pengukuran diletakkan pada bagian paling akhir aktivasi form (form activation), yaitu dengan menggunakan fungsi `GetTickCount()` yang hasilnya disimpan pada variabel `FEnd` yang bertipe `Cardinal`.

### Teknik Penyisipan Kode

Pasti ada yang berpendapat bahwa kode disisipkan pada *event* `OnCreate` (awal pengukuran) dan `OnActivate` (akhir pengukuran). Tidak. Seperti yang telah dijelaskan pada siklus hidup form, peletakan pada *event* `OnCreate` tidak mengukur rutin untuk

membaca form DFM dan pembuatan komponen yang ada didalamnya. Dan penyisipan pada *event* OnActivate juga demikian, tidak menjamin bahwa semua rutin sudah dilakukan. Lalu dimana?

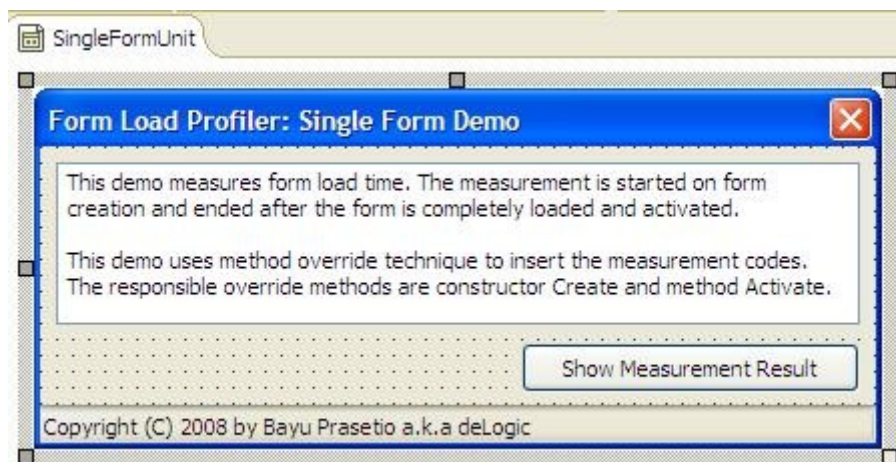
Tentu saja di constructor *Create*. Untuk mewujudkannya, perlu dilakukan penimpaan (*override*) constructor *Create*, menyisipkan kode pengukuran diatas segalanya, kemudian memanggil *constructor* yang sebenarnya (*inherited*).

Demikian pula untuk menyisipkan pengukuran akhir. *Method* *Activate* harus ditimpa (*override*) dengan meletakkan kode pengukuran di bawah segalanya, dengan sebelumnya memanggil *method* *Activate* yang sebenarnya (*inherited*).

### 3. Menerapkan Teknik Pengukuran

Berdasarkan metode pengukuran yang telah ditentukan, penerapannya dalam kode sumber dicontohkan pada bagian berikut.

Rancanglah sebuah proyek aplikasi yang terdiri dari 1 (satu) form. Inti dari aplikasi ini adalah untuk menampilkan waktu tayang yang telah diukur melalui sebuah tombol. Dalam form tersebut terdapat TMemo, TButton dan TStatusBar seperti yang digambarkan pada disain form berikut:



Kemudian lengkapi kode sumber seperti di bawah ini:

```
unit SingleFormUnit;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
  Dialogs, StdCtrls, ComCtrls;  
  
type  
  TSingleForm = class(TForm)  
    btnShowResult: TButton;  
    mmoLegend: TMemo;  
    stbMain: TStatusBar;  
    procedure btnShowResultClick(Sender: TObject);  
  private
```

```
{ Private declarations }
FStart, FEnd : Cardinal;
public
{ Public declarations }
constructor Create(AOwner: TComponent); override;
procedure Activate; override;
end;

var
  SingleForm: TSingleForm;

implementation

{$R *.dfm}

{ TSingleForm }

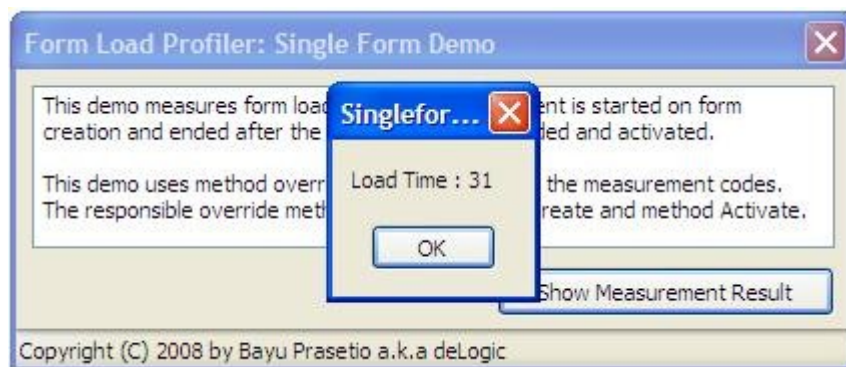
procedure TSingleForm.Activate;
begin
  inherited;
  FEnd := GetTickCount;
end;

procedure TSingleForm.btnShowResultClick(Sender: TObject);
begin
  ShowMessage('Load Time : ' + IntToStr(FEnd - FStart));
end;

constructor TSingleForm.Create(AOwner: TComponent);
begin
  FStart := GetTickCount;
  inherited;
end;

end.
```

Pengukuran secara otomatis akan dilakukan ketika aplikasi dijalankan. Klik tombol "Show Measurement Result" untuk melihat hasilnya.



Nah selesai, cukup mudah bukan ?

Selengkapnya mengenai kode sumber demo mengukur waktu tayang dapat disimak pada kode sumber *FormLoadProfiler-SingleForm.7z* terlampir.

## 4. Teknik Pengukuran Pada Banyak Form

Setelah berhasil menerapkan teknik pengukuran pada sebuah form, langkah selanjutnya adalah bagaimana melakukan pengukuran pada banyak form. Sebuah aplikasi tentunya

tidak hanya terdiri dari 1 (satu) form saja bukan?

Menerapkan kode pengukuran pada setiap form tentu saja bisa dilakukan, namun bagaimana apabila jumlah form nya sangat banyak? Cukup menyita waktu dan tenaga bukan ?

Nah bagaimana teknik menambahkan kode pengukuran tanpa harus menuliskannya ke setiap form yang diinginkan secara berulang – ulang ?

Salah satu teknik yang dapat digunakan adalah *interposer class*. Berikut pengertian *interposer class* yang dicuplik dari catatan ke 67 dari *handbook* Marco Cantu:

*An interposer class is a class with the same name of the class it wants to change (using inheritance). By adding its unit after the unit of its base class in a uses statement, a unit will use the modified version of the class rather than the original. Interposer classes are not a terribly neat technique, rather a hack. But they can be very handy, indeed!*

## Membuat Interposer Class

Lalu bagaimana membuat *interposer class*? Pertama, buat kelas TForm baru yang merupakan turunan dari kelas TForm yang asli, yaitu kelas yang dideklarasikan pada unit Forms. Pendeklarasian dari kelas induk harus menyertakan unit asal dimana kelas TForm yang sebenarnya berasal, yaitu Forms.TForm.

```
TForm = class(Forms.TForm);
```

Kedua, tambahkan variable FStart dan FEnd. Untuk lingkup aksesnya, bisa dipilih apakah *Private*, *Protected* atau *Public*. Karena kita ingin mengakses variabel FStart dan FEnd, maka lingkup aksesnya minimal adalah *protected*. Anda bisa saja membuatnya sebagai *private*, namun nantinya harus menambahkan properti untuk mengakses kedua variabel tersebut.

```
protected  
  FStart, FEnd : Cardinal;
```

Ketiga, tidak lupa deklarasikan konstruktor Create dan *method* Activate yang akan ditimpa (*override*).

```
public  
  constructor Create(AOwner: TComponent); override;  
  procedure Activate; override;
```

Sehingga deklarasi kelas *interposer* menjadi:

```
type  
  TForm = class(Forms.TForm)  
    protected  
      FStart, FEnd : Cardinal;  
    public  
      { Public declarations }  
      constructor Create(AOwner: TComponent); override;  
      procedure Activate; override;  
  end;
```

Rangkaian kode selanjutnya sama dengan pada kode untuk form tunggal, sehingga kode lengkapnya menjadi sebagai berikut:

```
{-----  
The contents of this file are subject to the Mozilla Public License  
Version 1.1 (the "License"); you may not use this file except in compliance  
with the License. You may obtain a copy of the License at  
http://www.mozilla.org/MPL/MPL-1.1.html  
  
Software distributed under the License is distributed on an "AS IS" basis,  
WITHOUT WARRANTY OF ANY KIND, either expressed or implied. See the License for  
the specific language governing rights and limitations under the License.  
  
The Original Code is: MultipleInterposerClass.pas, released on 2008-09-15  
  
The Initial Developer of the Original Code is Bayu Prasetio  
Portions created by Bayu Prasetio are Copyright (C) 2008 Bayu Prasetio.  
All Rights Reserved.  
-----}  
unit MultipleInterposerClass;  
  
interface  
  
uses  
  Classes, Forms;  
  
type  
  TForm = class(Forms.TForm)  
  protected  
    FStart, FEnd : Cardinal;  
  public  
    { Public declarations }  
    constructor Create(AOwner: TComponent); override;  
    procedure Activate; override;  
  end;  
  
implementation  
  
{ TForm }  
  
uses  
  Windows;  
  
procedure TForm.Activate;  
begin  
  inherited;  
  FEnd := GetTickCount;  
end;  
  
constructor TForm.Create(AOwner: TComponent);  
begin  
  FStart := GetTickCount;  
  inherited;  
end;  
  
end.
```

## Menggunakan Interposer Class

Langkah selanjutnya adalah bagaimana menggunakan *interposer class* tersebut. Caranya cukup mudah, yaitu tambahkan unit dimana *interposer class* tersebut dideklarasikan pada bagian klausul `uses`.

Yang perlu diperhatikan adalah, urutan deklarasi unit tersebut harus terletak **setelah** unit `Forms` karena jika dideklarasikan sebelum unit `Forms`, maka yang akan dijadikan sebagai rujukan adalah kelas `TForm` yang dideklarasikan pada unit `Forms`, bukan kelas `TForm`



dari unit `MultipleInterposerClass`. Dan setelah deklarasi ditambahkan, selesai sudah.

Berikut contoh bagaimana menggunakan *interposer class*:

```
unit MultipleMainFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, MultipleInterposerClass;

type
  TMainForm = class(TForm)
    btnShowResult: TButton;
    mmoLegend: TMemo;
    stbMain: TStatusBar;
    btnLoadSecondary: TButton;
    lblLoadTime: TLabel;
    btnLoadAnotherForm: TButton;
    procedure btnShowResultClick(Sender: TObject);
    procedure btnLoadSecondaryClick(Sender: TObject);
    procedure btnLoadAnotherFormClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.dfm}

{ TSingleForm }

uses
  MultipleSecondaryFormUnit, MultipleAnotherFormUnit;

procedure TMainForm.btnLoadAnotherFormClick(Sender: TObject);
begin
  AnotherForm := TAnotherForm.Create(nil);
  AnotherForm.ShowModal;
  AnotherForm.Free;
end;

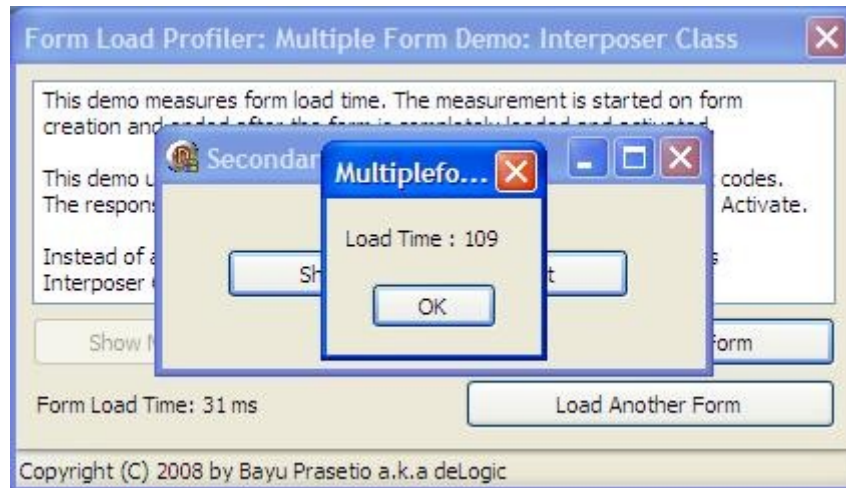
procedure TMainForm.btnLoadSecondaryClick(Sender: TObject);
begin
  SecondaryForm := TSecondaryForm.Create(nil);
  SecondaryForm.ShowModal;
  SecondaryForm.Free;
end;

procedure TMainForm.btnShowResultClick(Sender: TObject);
begin
  lblLoadTime.Caption := ' Form Load Time: ' + IntToStr(FEnd - FStart) + ' ms';
  btnShowResult.Enabled := False;
end;

end.
```

Berikut contoh demo hasil pengukuran dengan banyak form:





Selengkapnya mengenai kode sumber demo mengukur waktu tayang untuk banyak form dapat disimak pada kode sumber *FormLoadProfiler-InterposerClass.7z* terlampir.

## 5. Teknik Pengukuran Pada Banyak Form Dengan Class Helper

Selain menggunakan *interposer class* untuk melakukan pengukuran pada banyak form, teknik lain yang dapat digunakan adalah *class helper*.

Berikut cuplikan pengertian class helper menurut dokumentasi Delphi:

A class helper is a type that - when associated with another class - introduces additional method names and properties which may be used in the context of the associated class (or its descendants). Class helpers are a way to extend a class without using inheritance. A class helper simply introduces a wider scope for the compiler to use when resolving identifiers. When you declare a class helper, you state the helper name, and the name of the class you are going to extend with the helper. You can use the class helper any place where you can legally use the extended class. The compiler's resolution scope then becomes the original class, plus the class helper.

Class helpers provide a way to extend a class, but they should not be viewed as a design tool to be used when developing new code. They should be used solely for their intended purpose, which is language and platform RTL binding.

Perbedaan mendasar antara *interposer class* dan *class helper* adalah bahwa *interposer class* dilakukan dengan cara menurunkan kelas yang akan dikembangkan, sedangkan *class helper* tanpa perlu menurunkannya.

### Membuat Class Helper

Lalu bagaimana membuat *class helper*? Pertama, deklarasikan helper untuk kelas `TForm`.

```
TGlobalFormHelper = class helper for TForm  
end;
```

Kemudian untuk mendeklarasikan variabel `FStart` dan `FEnd`, tidak bisa dilakukan secara sembarangan mengingat *class helper* memberikan batasan tidak dapat menambah *instance data*, namun dapat menambah *class fields*. Nah untuk itu harus memanfaatkan **class var**, sehingga deklarasi variabelnya menjadi:

```
TGlobalFormHelper = class helper for TForm
  class var
    FStart, FEnd : Cardinal;
end;
```

langkah selanjutnya adalah mendeklarasikan konstruktor `Create`, namun kali ini deklarasinya sedikit berbeda. Jika pada *interposer class*, kita harus menimpa (*override*) konstruktor `Create`, maka pada *class helper* tidak perlu. Secara umum, isi dari konstruktor adalah sebagai berikut:

```
TGlobalFormHelper = class helper for TForm
  class var
    FStart, FEnd : Cardinal;
  public
    constructor Create(AOwner: TComponent);
end;
//
// ....
//
constructor TGlobalFormHelper.Create(AOwner: TComponent);
begin
  FStart := GetTickCount;
  inherited;
end;
```

Sampai di sini kita sudah mulai menghitung pengukuran, langkah berikutnya tentu saja adalah meletakkan rutin akhir pengukuran pada *method* `Activate`. Namun sayang sekali, dalam *class helper* ini, kita tidak dapat mendeklarasikan dan atau menimpa (*override*) *method* `Activate` karena **sudah dideklarasikan**.

Sebagai alternatif solusinya, kita akan **membelokkan event `OnActivate`** untuk menjalankan rutin pengukuran. Untuk itu kita harus menyimpan *event* `OnActivate` yang sebenarnya. Kemudian di dalam rutin pengukuran tersebut, jalankan *event* `OnActivate` yang sebenarnya sebelum mengakhiri perhitungan. Selengkapnya menjadi:

```
TGlobalFormHelper = class helper for TForm
  class var
    FStart, FEnd : Cardinal;
    FOnActivateTemporary : TNotifyEvent;
  public
    constructor Create(AOwner: TComponent);
    procedure CalculateLoadTime(Sender: TObject);
end;
//
// ....
//
procedure TGlobalFormHelper.CalculateLoadTime(Sender: TObject);
begin
  if Assigned(FOnActivateTemporary) then FOnActivateTemporary(Sender);
  FEnd := GetTickCount;
end;

constructor TGlobalFormHelper.Create(AOwner: TComponent);
begin
  FStart := GetTickCount;
  inherited;

  FOnActivateTemporary := Self.OnActivate;
  Self.OnActivate := CalculateLoadTime;
```

```
end;
```

Sampai disini *class helper* sudah selesai, sekarang bagaimana dengan kelas utama nya? Cukup dengan menambahkan konstruktor *Create* dan tambahkan *directive* override. Di dalam konstruktor tersebut, tambahkan kode **inherited** untuk memanggil konstruktor *Create* yang sebenarnya, yaitu yang dideklarasikan pada *class helper*.

```
TMainForm = class(TForm)
//
// ...
//
public
{ Public declarations }
constructor Create(AOwner: TComponent); override;
end;
//
// ...
//
constructor TMainForm.Create(AOwner: TComponent);
begin
inherited;
end;
```

## Menggunakan Class Helper

Langkah selanjutnya adalah bagaimana memanfaatkan *class helper* tersebut pada form. Untuk form utama, tidak ada masalah, tidak ada tambahan apapun karena *class helper* dideklarasikan dalam unit yang sama.

```
{-----
The contents of this file are subject to the Mozilla Public License
Version 1.1 (the "License"); you may not use this file except in compliance
with the License. You may obtain a copy of the License at
http://www.mozilla.org/MPL/MPL-1.1.html

Software distributed under the License is distributed on an "AS IS" basis,
WITHOUT WARRANTY OF ANY KIND, either expressed or implied. See the License for
the specific language governing rights and limitations under the License.

The Original Code is: MultipleMainFormUnit.pas, released on 2008-09-25

The Initial Developer of the Original Code is Bayu Prasetio
Portions created by Bayu Prasetio are Copyright (C) 2008 Bayu Prasetio.
All Rights Reserved.
-----}
unit MultipleMainFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls;

type
  TMainForm = class(TForm)
    btnShowResult: TButton;
    mmoLegend: TMemo;
    stbMain: TStatusBar;
    btnLoadSecondary: TButton;
    lblLoadTime: TLabel;
    btnLoadAnotherForm: TButton;
    procedure btnShowResultClick(Sender: TObject);
    procedure btnLoadSecondaryClick(Sender: TObject);
    procedure btnLoadAnotherFormClick(Sender: TObject);
  private
    { Private declarations }
  public
```

```

    { Public declarations }
    constructor Create(AOwner: TComponent); override;
end;

TGlobalFormHelper = class helper for TForm
    class var
        FStart, FEnd : Cardinal;
        FOnActivateTemporary : TNotifyEvent;
    public
        constructor Create(AOwner: TComponent);
        procedure CalculateLoadTime(Sender: TObject);
    end;

var
    MainForm: TMainForm;

implementation

{$R *.dfm}

uses
    MultipleSecondaryFormUnit, MultipleAnotherFormUnit;

procedure TMainForm.btnLoadAnotherFormClick(Sender: TObject);
begin
    AnotherForm := TAnotherForm.Create(nil);
    AnotherForm.ShowModal;
    AnotherForm.Free;
end;

procedure TMainForm.btnLoadSecondaryClick(Sender: TObject);
begin
    SecondaryForm := TSecondaryForm.Create(nil);
    SecondaryForm.ShowModal;
    SecondaryForm.Free;
end;

procedure TMainForm.btnShowResultClick(Sender: TObject);
begin
    lblLoadTime.Caption := ' Form Load Time: ' + IntToStr(FEnd - FStart) + ' ms';
    btnShowResult.Enabled := False;
end;

constructor TMainForm.Create(AOwner: TComponent);
begin
    inherited;
end;

procedure TGlobalFormHelper.CalculateLoadTime(Sender: TObject);
begin
    if Assigned(FOnActivateTemporary) then FOnActivateTemporary(Sender);
    FEnd := GetTickCount;
end;

constructor TGlobalFormHelper.Create(AOwner: TComponent);
begin
    FStart := GetTickCount;
    inherited;

    FOnActivateTemporary := Self.OnActivate;
    Self.OnActivate := CalculateLoadTime;
end;

end.

```

Kemudian untuk form yang lainnya, deklarasi mirip dengan menggunakan *interposer class*, yaitu cukup dengan menambahkan pada klausul *uses* dengan nama unit dimana *class helper* dideklarasikan, seperti contoh kode berikut:

```

{-----
The contents of this file are subject to the Mozilla Public License
Version 1.1 (the "License"); you may not use this file except in compliance

```

with the License. You may obtain a copy of the License at  
<http://www.mozilla.org/MPL/MPL-1.1.html>

Software distributed under the License is distributed on an "AS IS" basis,  
WITHOUT WARRANTY OF ANY KIND, either expressed or implied. See the License for  
the specific language governing rights and limitations under the License.

The Original Code is: MultipleAnotherFormUnit.pas, released on 2008-09-25

The Initial Developer of the Original Code is Bayu Prasetio  
Portions created by Bayu Prasetio are Copyright (C) 2008 Bayu Prasetio.  
All Rights Reserved.

```
-----}
unit MultipleAnotherFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TAnotherForm = class(TForm)
    btnShowResult: TButton;
    procedure btnShowResultClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AnotherForm: TAnotherForm;

implementation

{$R *.dfm}

uses
  MultipleMainFormUnit;

procedure TAnotherForm.btnShowResultClick(Sender: TObject);
begin
  ShowMessage('Load Time : ' + IntToStr(FEnd - FStart));
end;

procedure TAnotherForm.FormCreate(Sender: TObject);
begin
  Sleep(200);
end;

end.
```

Selengkapnya mengenai kode sumber demo mengukur waktu tayang untuk banyak form dengan menggunakan *class helper* dapat disimak pada kode sumber *FormLoadProfiler-ClassHelper.7z* terlampir.

## 6. Meningkatkan Presisi Perhitungan Waktu

Dari beberapa variasi teknik pengukuran yang telah dibahas, semuanya menggunakan metode pengukuran waktu yang sama, yaitu menggunakan fungsi `GetTickCount`. Fungsi `GetTickCount` memiliki ketelitian hingga hitungan mili-detik (ms). Untuk kebutuhan secara umum, `GetTickCount` sudah mencukupi, namun adakalanya pengukuran yang dilakukan membutuhkan ketelitian yang lebih, misalnya dalam nano-

detik. Disinilah GetTickCount tidak bisa memenuhinya.

## QueryPerformanceCounter dan QueryPerformanceFrequency

Windows menyediakan 2 (dua) API yang berkaitan dengan pengukuran dengan ketelitian tinggi, yaitu QueryPerformanceCounter dan QueryPerformanceFrequency. Kedua fungsi ini digunakan saling melengkapi.

Berikut definisi API QueryPerformanceCounter yang diambil dari dokumentasi Delphi:

```
BOOL QueryPerformanceCounter(  
    LARGE_INTEGER *lpPerformanceCount    // address of current counter  
    value  
);
```

*Parameters*

*lpPerformanceCount*

*Points to a variable that the function sets, in counts, to the current performance-counter value. If the installed hardware does not support a high-resolution performance counter, this parameter can be to zero.*

*Return Values*

*If the installed hardware supports a high-resolution performance counter, the return value is nonzero.  
If the installed hardware does not support a high-resolution performance counter, the return value is zero.*

Berikut definisi API QueryPerformanceFrequency yang diambil dari dokumentasi Delphi:

```
BOOL QueryPerformanceFrequency(  
    LARGE_INTEGER *lpFrequency    // address of current frequency  
);
```

*Parameters*

*lpFrequency*

*Points to a variable that the function sets, in counts per second, to the current performance-counter frequency. If the installed hardware does not support a high-resolution performance counter, this parameter can be to zero.*

*Return Values*

*If the installed hardware supports a high-resolution performance counter, the return value is nonzero.  
If the installed hardware does not support a high-resolution performance counter, the return value is zero.*

Dari kutipan diatas, sudah jelas bahwa QueryPerformanceCounter digunakan untuk menampung nilai pencacah / penghitung pada saat API tersebut dipanggil, sedangkan QueryPerformanceFrequency digunakan untuk mendapatkan besaran frekuensi dari pencacah tersebut, berapa nilai cacah yang dihasilkan dalam 1 (satu) detik.

Sehingga untuk mendapatkan nilai pengukuran adalah dengan menghitung nilai cacah setelah pengukuran dikurangi dengan nilai cacah sebelum pengukuran. Nilai tersebut kemudian dibagi dengan frekuensi. Nilai ukur yang didapat adalah dalam satuan detik, untuk mengkonversi dalam satuan lain tinggal disesuaikan. Misalnya untuk dihitung dalam satuan mili-detik maka nilai tersebut dikalikan dengan 1000.

```
FElapsedTime := (FEnd - FStart) / FFrequency * 1000;
```

## Integrasi ke Kode

Sekarang saatnya melakukan konversi dari penggunaan API `GetTickCount` ke `QueryPerformanceCounter` dan `QueryPerformanceFrequency`. Disini dasar kode sumber yang akan digunakan adalah teknik *interposer class*. Untuk penggunaan teknik *class helper* silahkan disesuaikan sendiri sebagai latihan.

Pertama, deklarasi variabel `FStart` dan `FEnd` harus diubah tipe datanya dari `Cardinal` menjadi `Int64`. Mengapa `Int64`? Karena nilai yang ditampung adalah sangat besar mengingat ketelitian presisi yang dihasilkan sangat tinggi, sehingga dibutuhkan tipe data yang lebih besar dari `Cardinal` (0..4294967295), yaitu `Int64` atau `LargeInt` ( $-2^{63}..2^{63}-1$ ). Kemudian tambahkan variabel untuk menyimpan nilai frekuensi dari pencacah tersebut, yaitu `FFrequency`, dengan tipe data `Int64`. Selanjutnya tambahkan variabel untuk menampung nilai hasil pengukuran, yaitu `FElapsedTime`, dengan tipe data `Double`.

```
protected
  FStart, FEnd : Int64;
  FFrequency   : Int64;
  FElapsedTime : Double;
```

Perubahan kedua adalah pada konstruktor `Create`. Disini perlu dilakukan pemanggilan fungsi `QueryPerformanceFrequency` untuk mengetahui frekuensi nilai pencacah.

```
constructor TForm.Create(AOwner: TComponent);
begin
  QueryPerformanceFrequency(FFrequency);
  QueryPerformanceCounter(FStart);
  inherited;
end;
```

Ketiga, modifikasi *method* `Activate`, dan menambahkan perhitungan waktu ukur dalam 1 (satu) mili-detik.

```
procedure TForm.Activate;
begin
  inherited;
  QueryPerformanceCounter(FEnd);
  FElapsedTime := (FEnd - FStart) / FFrequency * 1000;
end;
```

Modifikasi keempat adalah modifikasi pada unit - unit lain yang menggunakan unit `MultipleInterposerClassHP.pas`. Bagian yang diubah adalah cara penyampaian pesan informasi besarnya waktu tayang.

Jika sebelumnya menggunakan fungsi `IntToStr` karena waktu ukur berbasis pada tipe



data Cardinal, maka sekarang menjadi FloatToStr karena waktu ukur, yaitu variabel FEelapsedTime bertipe data Double. Sebagai contoh unit yang dimodifikasi adalah unit MultipleMainFormUnit.pas.

```
procedure TMainForm.btnShowResultClick(Sender: TObject);
begin
  lblLoadTime.Caption := ' Form Load Time: ' + FloatToStr(FElapsedTime) + ' ms';
  btnShowResult.Enabled := False;
end;
```

Sehingga rangkaian kode MultipleInterposerClassHP.pas menjadi sebagai berikut:

```
{-----}
The contents of this file are subject to the Mozilla Public License
Version 1.1 (the "License"); you may not use this file except in compliance
with the License. You may obtain a copy of the License at
http://www.mozilla.org/MPL/MPL-1.1.html

Software distributed under the License is distributed on an "AS IS" basis,
WITHOUT WARRANTY OF ANY KIND, either expressed or implied. See the License for
the specific language governing rights and limitations under the License.

The Original Code is: MultipleInterposerClassHP.pas, released on 2008-09-30

The Initial Developer of the Original Code is Bayu Prasetio
Portions created by Bayu Prasetio are Copyright (C) 2008 Bayu Prasetio.
All Rights Reserved.
-----}
unit MultipleInterposerClassHP;

interface

uses
  Classes, Forms;

type
  TForm = class(Forms.TForm)
  protected
    FStart, FEnd : Int64;
    FFrequency   : Int64;
    FElapsedTime : Double;
  public
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
    procedure Activate; override;
  end;

implementation

{ TForm }

uses
  Windows;

procedure TForm.Activate;
begin
  inherited;
  QueryPerformanceCounter(FEnd);
  FElapsedTime := (FEnd - FStart) / FFrequency * 1000;
end;

constructor TForm.Create(AOwner: TComponent);
begin
  QueryPerformanceFrequency(FFrequency);
  QueryPerformanceCounter(FStart);
  inherited;
end;

end.
```

Dan sebagai contoh MultipleMainFormUnit.pas menjadi:

```
{-----
The contents of this file are subject to the Mozilla Public License
Version 1.1 (the "License"); you may not use this file except in compliance
with the License. You may obtain a copy of the License at
http://www.mozilla.org/MPL/MPL-1.1.html

Software distributed under the License is distributed on an "AS IS" basis,
WITHOUT WARRANTY OF ANY KIND, either expressed or implied. See the License for
the specific language governing rights and limitations under the License.

The Original Code is: MultipleMainFormUnit.pas, released on 2008-09-30

The Initial Developer of the Original Code is Bayu Prasetio
Portions created by Bayu Prasetio are Copyright (C) 2008 Bayu Prasetio.
All Rights Reserved.
-----}
unit MultipleMainFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, MultipleInterposerClassHP;

type
  TMainForm = class(TForm)
    btnShowResult: TButton;
    mmoLegend: TMemo;
    stbMain: TStatusBar;
    btnLoadSecondary: TButton;
    lblLoadTime: TLabel;
    btnLoadAnotherForm: TButton;
    procedure btnShowResultClick(Sender: TObject);
    procedure btnLoadSecondaryClick(Sender: TObject);
    procedure btnLoadAnotherFormClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.dfm}

{ TSingleForm }

uses
  MultipleSecondaryFormUnit, MultipleAnotherFormUnit;

procedure TMainForm.btnLoadAnotherFormClick(Sender: TObject);
begin
  AnotherForm := TAnotherForm.Create(nil);
  AnotherForm.ShowModal;
  AnotherForm.Free;
end;

procedure TMainForm.btnLoadSecondaryClick(Sender: TObject);
begin
  SecondaryForm := TSecondaryForm.Create(nil);
  SecondaryForm.ShowModal;
  SecondaryForm.Free;
end;

procedure TMainForm.btnShowResultClick(Sender: TObject);
begin
```

```
lblLoadTime.Caption := ' Form Load Time: ' + FloatToStr(FElapsedTime) + ' ms';  
btnShowResult.Enabled := False;  
end;  
end.
```

Selengkapnya mengenai kode sumber demo mengukur waktu tayang dengan presisi tinggi dapat disimak pada kode sumber *FormLoadProfiler-HighPrecision-InterposerClass.7z* terlampir.

## Penutup

Pada artikel ini telah dibahas bagaimana melakukan pengukuran sederhana waktu tayang suatu form dari pembuatan hingga siap digunakan oleh pengguna. Berbagai implementasi teknik pengukuran juga telah dipaparkan, mulai dari implementasi pada form tunggal hingga banyak form, baik dengan metode *interposer class* maupun *class helper*. Kemudian juga telah dipaparkan penggunaan teknik penghitung waktu yang lebih akurat dengan presisi tinggi dengan menggunakan fungsi *QueryPerformanceFrequency* dan *QueryPerformanceCounter*.

Apa yang telah dipaparkan dalam artikel ini hanyalah sebagai pintu gerbang bagi Anda untuk melakukan eksplorasi lebih mendalam dan mengembangkannya jauh lebih baik. Silahkan gunakan logika, kreativitas dan imajinasi Anda. Tetap Semangat !

## Referensi

1. Form Load Profiler: Single Form, Bayu Prasetyo, <http://blog.bprasetyo.or.id/2008/09/01/form-load-profiler-single-form/>
2. Form Load Profiler: Multiple Form, Bayu Prasetyo, <http://blog.bprasetyo.or.id/2008/09/15/form-load-profiler-multiple-form/>
3. Handbook Note 67/113: Interposer Classes, Marco Cantu, [http://blog.marcocantu.com/blog/handbook\\_note\\_67.html](http://blog.marcocantu.com/blog/handbook_note_67.html)
4. Delphi Help, CodeGear
5. Form Load Profiler: Multiple Form: Class Helper, Bayu Prasetyo, <http://blog.bprasetyo.or.id/2008/09/26/form-load-profiler-multi-form-class-helper/>
6. Form Load Profiler: High Precision Timer, Bayu Prasetyo, <http://blog.bprasetyo.or.id/2008/10/06/form-load-profiler-high-precision-timer/>

## Biografi Penulis



**Bayu Prasetyo.** Menyelesaikan studi S1 di STIMIK Muhammadiyah Jakarta tahun 2004. Bekerja di sebuah instansi pemerintah sejak 1999. Waktu luangnya antara lain diisi dengan berbagi ilmu di forum Delphi Indonesia (<http://delphi-id.org>) dan milis lain yang berkaitan dengan Delphi, menulis buku, artikel baik di media cetak maupun elektronik, termasuk *blog* serta *programmer* lepas (*part-time programmer*). Tulisan terkait dengan pemrograman, khususnya Delphi tersedia di <http://blog.bprasetyo.or.id>.