

Kustomisasi MessageDlg

Bayu Prasetyo

bprasetio@gmail.com

http://www.bprasetio.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.

Setiap aplikasi yang mempunyai rutin interaksi dengan pengguna umumnya tidak lepas dengan jendela pesan dialog. Jenis pesan yang disampaikan dapat berupa informasi (*information*), konfirmasi (*confirmation*), peringatan (*warning*) dan kesalahan (*error*).

Dalam pemrograman Delphi, ada beberapa fungsi yang berkaitan dengan jendela pesan dialog, antara lain ShowMessage, MessageDlg, MessageBox dan sebagainya.

Pada artikel ini akan dibahas bagaimana melakukan kustomisasi terhadap fungsi MessageDlg. Kustomisasi yang akan dibahas meliputi penggantian teks tombol dan pengaturan perataan teks pesan.

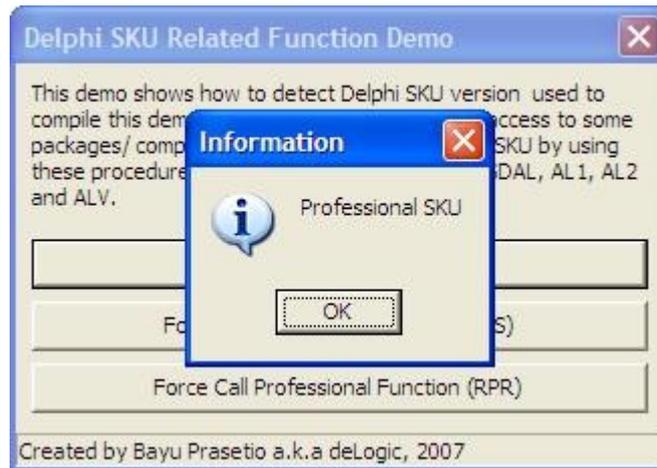
1. Mengganti Teks Judul dan Tombol

MessageDlg Standar dan Terkustomisasi

Apabila terdapat cuplikan kode program:

```
MessageDlg('Professional SKU', mtInformation, [mbOK], 0);
```

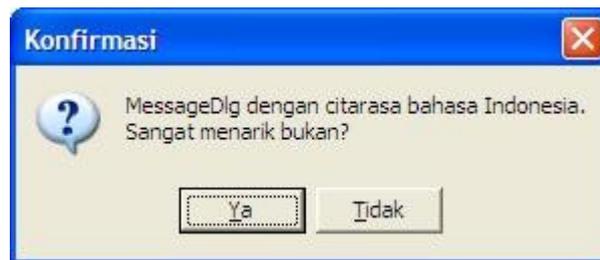
Maka akan ditayangkan jendela dialog ber-tipe informasi dengan teks pesan “Professional SKU” dan tombol “OK” seperti pada gambar berikut:



Jika terdapat cuplikan kode:

```
MessageDlg('MessageDlg dengan citarasa bahasa Indonesia.' + #13#10 +  
          'sangat menarik bukan?', mtConfirmation,  
          [mbYes, mbNo], 0);
```

Lalu bagaimana caranya agar tayangan kode di atas menjadi seperti gambar berikut:



Perhatikan jendela dialog, yang secara standar berbahasa Inggris menjadi berbahasa Indonesia. Judul yang seharusnya bertuliskan “Confirmation”, berubah menjadi “Konfirmasi”. Kemudian teks tombol “Yes” berubah menjadi “Ya”, dan teks tombol “No” berubah menjadi “Tidak”.

Dan perhatikan bahwa penayangan jendela dialog tetap menggunakan pemanggilan fungsi MessageDlg, bukan pemanggilan fungsi lainnya, apalagi menggunakan kelas turunan !

Pilihan Teknik Kustomisasi

Ada beberapa teknik yang dapat dilakukan untuk melakukan kustomisasi MessageDlg seperti pada gambar di atas, antara lain:

- Membuat ulang fungsi MessageDlg (*built from scratch*)
Alternatif pertama adalah dengan membuat sendiri pustaka untuk menampilkan jendela dialog dengan bahasa Indonesia. Disini, diperlukan form dan konstanta tombol berbahasa Indonesia, membuat rutin yang berkaitan dengan antar muka, membuat *event handler* yang dibutuhkan dan sebagainya. Ah.. sudah terbayang betapa merepotkannya!

- Modifikasi *Source Code*
Alternatif kedua, memodifikasi bahasa / teks yang digunakan oleh fungsi `MessageDlg` yang disimpan pada unit *Consts*. Namun pilihan ini menuntut Anda untuk mengkompilasi ulang kode sumber unit Delphi, sangat banyak dan beresiko terutama jika terdapat unit yang Anda tidak memiliki *source-code* nya.
- Memanfaatkan *facade pattern* fungsi `CreateMessageDialog`
Alternatif yang lebih elegan adalah dengan menggunakan *facade pattern* dari fungsi `CreateMessageDialog` yang boleh jadi merupakan inti dari pembuatan jendela dialog. Jendela dialog dapat lebih variatif. Berdasarkan form hasil pemanggilan fungsi `CreateMessageDialog`, lakukan beberapa perubahan teks judul jendela dan tombol yang ada sesuai dengan yang dikehendaki. Namun cara ini tidak dapat dilakukan karena pemanggilan jendela dialog tidak dilakukan melalui fungsi `MessageDlg`, namun melalui fungsi lain yang memanfaatkan `CreateMessageDialog` tersebut.
- Modifikasi alamat memori yang memuat teks tombol dan Judul
Teknik ini dilakukan dengan memodifikasi alamat memori yang memuat teks tombol dan judul kotak dialog pada saat aplikasi dalam proses inisialisasi. Teknik inilah yang akan dibahas pada artikel ini. Mengapa melalui teknik ini? Tentu saja alasannya adalah tidak perlu susah payah membangun dari awal, memodifikasi *source code* dan mengkompilasi ulang dan tidak perlu membuat *facade pattern* fungsi `CreateMessageDlg`. Dan tentu saja pemanggilannya masih tetap menggunakan `MessageDlg`. Cukup mengganti teks yang digunakan oleh `MessageDlg` pada saat *runtime*. Dan hasilnya cukup efektif.

Teknik Modifikasi

Seperti yang telah disebutkan sebelumnya, bahwa teks yang digunakan oleh fungsi `MessageDlg` disimpan pada unit *Consts*. Teks tersebut dideklarasikan sebagai *resource-string*, seperti berikut:

```
resourcestring
  SMsgDlgWarning = 'Warning';
  SMsgDlgError = 'Error';
  SMsgDlgInformation = 'Information';
  SMsgDlgConfirm = 'Confirm';
  SMsgDlgYes = '&Yes';
  SMsgDlgNo = '&No';
  SMsgDlgOK = 'OK';
  SMsgDlgCancel = 'Cancel';
  SMsgDlgHelp = '&Help';
  SMsgDlgHelpNone = 'No help available';
  SMsgDlgHelpHelp = 'Help';
  SMsgDlgAbort = '&Abort';
  SMsgDlgRetry = '&Retry';
  SMsgDlgIgnore = '&Ignore';
  SMsgDlgAll = '&All';
  SMsgDlgNoToAll = 'N&o to All';
  SMsgDlgYesToAll = 'Yes to &All';
```

Karena dideklarasikan sebagai *resource-string*, maka perlakuan pengubahannya tidak bisa sembarangan, yaitu menggunakan *pointer* `PResStringRec`. Yang mengarah pada *pointer* dimana *resource-string* tersebut dideklarasikan.

Deklarasi `PResStringRec` sendiri adalah sebagai berikut:

```
type
  PResStringRec = ^TResStringRec;
  TResStringRec = packed record
    Module: ^Cardinal;
    Identifier: Integer;
  end;
```

Dimana `Identifier` merupakan pengenalan suatu *resource-string*. Untuk teks dengan tipe data `PChar`, `Identifier` merupakan alamat memori dari `PChar` tersebut.

Untuk mengetahui alamat memori dari suatu `PChar`, dapat ditentukan melalui fungsi berikut:

```
AnAddress := Integer(APChar: PChar);
```

Sehingga pengubahan *resource-string* dapat dilakukan dengan:

```
APResStringRec^.Identifier := Integer(APChar);
```

Karena *resource-string* yang akan diubah cukup banyak, maka sebaiknya dibuat sebuah fungsi / prosedur yang mengaturnya:

```
procedure ReplaceResourceString(RStringRec: PResStringRec;
  AString: PChar);
begin
  if RStringRec = nil then Exit;
  RStringRec^.Identifier := Integer(AString);
end;
```

Berikut contoh untuk menggunakan prosedur di atas untuk mengubah teks judul jendela dari “Warning” menjadi “Peringatan”:

```
const _NewSMsgDlgWarning = 'Peringatan';
ReplaceResourceString(@SMsgDlgWarning, _NewSMsgDlg);
```

Apabila kode di atas dijalankan, pesan kesalahan akses tulis pada suatu alamat memori akan muncul, dengan teks seperti :

```
“Access violation at address xxxxxxxx in module 'nama_file.exe'.
Write of address yyyyyyyy.”
```

dimana `xxxxxxx` dan `yyyyyyy` merupakan suatu alamat di memori.

Hal ini terjadi karena adanya perintah untuk menulis di suatu alamat memori yang telah diproteksi terhadap akses tulis. Dan perintah akses tulis itu terletak pada saat pengubahan teks, yaitu pada *statement* `RStringRec^.Identifier := Integer(AString)`.

Untuk itu proteksi akses tulis ini harus dibuka terlebih dahulu, yaitu dengan API `VirtualProtect`. Berikut penjelasan dari dokumentasi Windows API:

```
The VirtualProtect function changes the access protection on a region of committed
```

pages in the virtual address space of the calling process.

```
BOOL VirtualProtect(  
    LPVOID lpAddress,    // address of region of committed pages  
    DWORD dwSize,       // size of the region  
    DWORD flNewProtect, // desired access protection  
    PDWORD lpflOldProtect // address of variable to get old protection  
);
```

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Sehingga prosedur `ReplaceResourceString` diubah menjadi:

```
procedure ReplaceResourceString(RStringRec: PResStringRec;  
    AString: PChar);  
var  
    OldProtect: Cardinal;  
begin  
    if RStringRec = nil then Exit;  
    if VirtualProtect(RStringRec, SizeOf(RStringRec^),  
        PAGE_EXECUTE_READWRITE, OldProtect) then  
    begin  
        RStringRec^.Identifier := Integer(AString);  
        VirtualProtect(RStringRec, SizeOf(RStringRec^),  
            OldProtect, @OldProtect);  
    end;  
end;
```

Secara ringkas, alur kerja kode diatas adalah sebagai berikut:

- Jika pointer `RStringRec` bernilai nil, maka proses tidak dilanjutkan, keluar dari prosedur;
- Lakukan operasi pengubahan proteksi akses alamat memori `RStringRec` menjadi dapat dieksekusi, baca dan tulis. Keadaan proteksi akses sebelumnya disimpan di variabel `OldProtect`;
- Lakukan pengubahan teks;
- Kembalikan proteksi akses alamat memori yang disimpan pada variabel `OldProtect`;

Dan jika kode dijalankan, maka kesalahan terkait dengan hak akses tulis tidak terjadi lagi.

Permanenkan Perubahan

Langkah selanjutnya adalah bagaimana menerapkan perubahan sejak awal, sejak aplikasi mulai diinisialisasi, sehingga ketika muncul jendela dialog, teks yang muncul sudah bukan standar dari Delphi, melainkan sudah dalam bentuk yang telah ditentukan. Perubahan ini bersifat permanen sampai aplikasi ditutup dan melingkupi semua form yang ada.

Salah satu cara adalah dengan meletakkannya pada bagian inisialisasi, misalnya:

```
initialization  
ReplaceResourceString(@SMsgDlgWarning, _NewSMsgDlgWarning);  
ReplaceResourceString(@SMsgDlgError, _NewSMsgDlgError);
```

Selengkapnya mengenai kode sumber demo mengubah teks judul dan tombol MessageDlg menjadi berbahasa Indonesia dapat disimak pada kode sumber *MessageDlgIndonesia.7z* terlampir.

Teks Judul dan Tombol yang Dinamis

Sekarang bagaimana bila perubahan teks judul dan tombol dapat dilakukan secara dinamis, yaitu teks dapat diganti sewaktu – waktu tergantung kebutuhan? Sebagai contoh adalah dua gambar berikut:



Sangat mudah sekali, tinggal memindahkan rutin penggantian teks, jika sebelumnya diletakkan pada bagian inisialisasi, kali ini diletakkan pada bagian dimana penggantian teks tersebut diperlukan, misalnya ketika menu cetak dipilih:

```
procedure TfrmCustomMessageDlg.SetCustomMessageContextPrint;  
begin  
    // sebagai contoh, ubah resource string untuk MessageDlg berdasarkan  
    konteks kejadian  
    // dalam hal ini adalah proses pencetakan  
    // mbOK disetarakan tayang ke layar (preview)  
    // mbYes disetarakan cetak ke printer  
    // mbNo disetarakan cetak ke dokumen PDF  
    ReplaceResourceString(@SMsgDlgConfirm, _SMsgDlgConfirmContextPrint);  
    ReplaceResourceString(@SMsgDlgYes, _SMsgDlgYesContextPrint);  
    ReplaceResourceString(@SMsgDlgNo, _SMsgDlgNoContextPrint);  
    ReplaceResourceString(@SMsgDlgOK, _SMsgDlgOKContextPrint);  
  
    // gunakan ModalResult dari MessageDlg untuk menentukan tindakan  
    selanjutnya  
    // hati - hati, Anda tidak dapat menggunakan ShowMessage sekehendak  
    hati  
    // karena ShowMessage sebenarnya MessageDlg dengan parameter  
    MessageType mtInformation
```

```
// dan Buttons [mbOK]. Pahami mengapa tombol 'OK' berubah menjadi 'Layar' ?
case MessageDlg('Tentukan tujuan pencetakan dokumen ?',
mtConfirmation, [mbOK, mbYes, mbNo], 0) of
  mrOK : ShowMessage('Dokumen ditayangkan ke layar');
  mrYes : ShowMessage('Dokumen dicetak ke printer');
  mrNo : ShowMessage('Dokumen disimpan dalam format .PDF');
end;
end;
```

Pada kesempatan lain, menu simpan dipilih:

```
procedure TfrmCustomMessageDlg.SetCustomMessageContextSave;
begin
  // sebagai contoh, ubah resource string untuk MessageDlg berdasarkan
  konteks kejadian
  // dalam hal ini adalah proses simpan
  ReplaceResourceString(@SMsgDlgConfirm, _SMsgDlgConfirmContextSave);
  ReplaceResourceString(@SMsgDlgYes, _SMsgDlgYesContextSave);
  ReplaceResourceString(@SMsgDlgNo, _SMsgDlgNoContextSave);
  ReplaceResourceString(@SMsgDlgCancel, _SMsgDlgCancelContextSave);

  MessageDlg('Anda Yakin akan menyimpan dokumen ini ?', mtConfirmation,
mbYesNoCancel, 0);
end;
```

Cukup mudah dan sederhana sekali bukan ?

Namun demikian metode ini mempunyai kelemahan, walaupun tidak tergolong fatal, yaitu ukuran tombol hanya berubah pada saat *ReplaceResourceString* pertama, pemanggilan berikutnya tidak mengubah ukuran tombol. Jadi, tugas Anda lah untuk melakukan eksplorasi dan pendalaman materi.

Selengkapnya mengenai kode sumber demo mengubah teks judul dan tombol MessageDlg secara dinamis dapat disimak pada kode sumber *CustomMessageDlg.7z* terlampir.

2. Mengatur Perataan Teks

Kustomisasi berikutnya adalah mengenai perataan teks. Secara default, teks pesan yang disajikan pada jendela dialog adalah rata kiri (*left justified*). Seperti pada perubahan teks jendela dan tombol, terdapat beberapa teknik modifikasi, namun yang saya pilih adalah teknik yang berkaitan dengan modifikasi alamat memori, yaitu *CodeRedirect*.

Secara singkat, *CodeRedirect* merupakan teknik membelokkan (mengganti alamat tujuan) pemanggilan suatu blok kode program menuju ke blok kode yang dikehendaki. Dengan demikian blok kode yang dibelokkan tidak pernah dieksekusi.

Blok Kode yang Diblokkan dan Penggantinya

Karena tujuan kustomisasi ini adalah mengubah perataan teks pada MessageDlg, maka kode yang dibelokkan tentu saja adalah pemanggilan fungsi MessageDlg. Untuk itu perlu dibuat sebuah fungsi yang akan dijadikan sebagai pengganti fungsi MessageDlg yang baru. Dan yang terpenting adalah fungsi tersebut harus mempunyai jumlah dan tipe parameter

yang sama. Berikut deklarasinya:

```
function MessageDlgCentered(const Msg: string; DlgType: TMsgDlgType;  
    Buttons: TMsgDlgButtons; HelpCtx: Integer): Integer;
```

Kemudian bagian implementasi dari fungsi tersebut adalah:

```
function MessageDlgCentered(const Msg: string; DlgType: TMsgDlgType;  
    Buttons: TMsgDlgButtons; HelpCtx: Integer): Integer;  
var  
    ALabel : TComponent;  
begin  
    with Dialogs.CreateMessageDialog(Msg, DlgType, Buttons) do  
        try  
            ALabel := FindComponent('Message');  
            if ALabel <> nil then  
                TLabel(ALabel).Alignment := taCenter;  
            Result := ShowModal;  
        finally  
            Free;  
        end  
    end;  
end;
```

Berikut penjelasan singkat mengenai alur kerja fungsi MessageDlgCentered:

- Buat jendela dialog dengan memanfaatkan fungsi CreateMessageDialog. Fungsi ini merupakan inti dari pembuatan jendela dialog. Pemanggilan fungsi MessageDlg standar berujung pada fungsi ini;
- Berdasarkan jendela dialog yang berhasil di buat, cari komponen pesan teks yang dimunculkan pada jendela, komponen ini mempunyai nama 'Message';
- Jika komponen Message ini tersedia, *cast* sebagai TLabel dan ubah properti yang mengatur perataan teks, yaitu Alignment sesuai dengan yang diinginkan, dalam hal ini adalah rata tengah, yaitu taCenter;
- Tampilkan jendela dialog secara modal;
- Akhirnya, bebaskan memori yang digunakan ketika jendela dialog ditutup.

Pemblokkan Blok Kode

Setelah mengetahui blok kode yang akan diblokkan dan blok kode penggantinya, langkah berikutnya adalah merumuskan rutin pemblokkan kode.

Pemblokkan pemanggilan blok kode dilakukan dengan perintah assembly JUMP dengan kode *mnemonic* \$E9 yang kemudian diikuti dengan informasi lokasi blok kode yang akan dituju di memori.

```
procedure CodeRedirect(Proc: Pointer; NewProc: Pointer);  
var  
    OldProtect: Cardinal;  
begin  
    if Proc = nil then Exit;  
    Proc := GetActualAddr(Proc);  
    if VirtualProtectEx(GetCurrentProcess, Proc, SizeOf(TInjectRec),  
        PAGE_EXECUTE_READWRITE, OldProtect) then  
        begin  
            TInjectRec(Proc^).Jump := $E9;  
            TInjectRec(Proc^).Offset := Integer(NewProc) - (Integer(Proc) +
```

```
        SizeOf(TInjectRec));  
    VirtualProtectEx(GetCurrentProcess, Proc, SizeOf(TInjectRec),  
                    OldProtect, @OldProtect);  
end;  
end;
```

Perhitungan lokasi blok kode yang akan dituju harus tepat, tidak boleh meleset karena apabila hal tersebut terjadi, maka dapat menimbulkan kesalahan, eksepsi, pelanggaran akses bahkan tidak responsifnya sistem operasi (*hang*).

Pada potongan kode di atas, instruksi *mnemonic* JUMP \$E9 dilakukan pada kode:

```
InjectRec(Proc^).Jump := $E9;
```

Sedangkan perhitungan alamat blok yang dituju dilakukan pada kode:

```
TInjectRec(Proc^).Offset := Integer(NewProc) - (Integer(Proc) +  
        SizeOf(TInjectRec)).
```

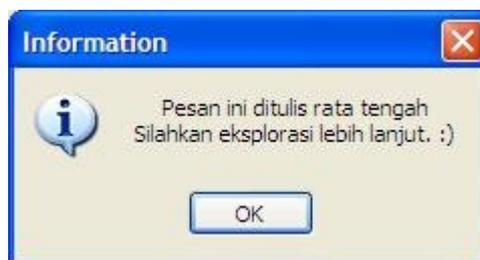
Kemudian cara menggunakan prosedur *CodeRedirect* adalah sebagai berikut:

```
CodeRedirect(@Dialogs.MessageDlg, @MessageDlgCentered);
```

Sehingga apabila terdapat *statement* pemanggilan *MessageDlg* seperti:

```
MessageDlg('Pesan ini ditulis rata tengah' + #13#10 +  
        'Silahkan eksplorasi lebih lanjut. :)',  
        mtInformation, [mbOK], 0);
```

Maka berikut tayangan jendela dialognya:



Selengkapnya mengenai kode sumber demo mengubah perataan teks pesan *MessageDlg* dapat disimak pada kode sumber *CustomMessageDlgCentered.7z* terlampir.

Penutup

Pada artikel ini telah dibahas bagaimana melakukan kustomisasi terhadap teks judul dan tombol *MessageDlg* serta pengubahan perataan teks pesan. Apa yang telah dipaparkan 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. Kode sumber RtlVclOptimize, Andreas Hausladen, <http://andy.jgknet.de/blog>
2. Meng-Indonesia-kan MessageDlg, Bayu Prasetyo, <http://blog.bprasetyo.or.id/2007/10/22/meng-indonesia-kan-messagedlg/>
3. Custom MessageDlg, Bayu Prasetyo, <http://blog.bprasetyo.or.id/2008/08/07/custom-messagedlg/>
4. Custom MessageDlg: Centered Message, Bayu Prasetyo, <http://blog.bprasetyo.or.id/2008/08/12/custom-messagedlg-centered-message/>

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>.