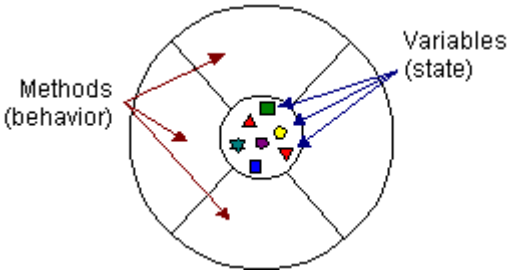
# 2. TEORI PENUNJANG

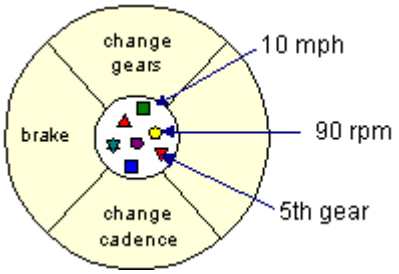
## 2.1. *Object Oriented Programming* (OOP)

*Object Oriented Programming* (OOP) adalah sebuah konsep revolusioner yang mengubah cara pandang pemrograman sebuah aplikasi. OOP menitikberatkan *programming* pada konsep *objects* daripada konsep *actions*, atau data daripada *logic*. Dengan menggunakan OOP, struktur data dalam sebuah program dipandang sebagai *object-object* yang memiliki data, *variable*, dan *method*-nya sendiri yang unik.




Gambar 2.1. Representasi *visual* sebuah *software object*

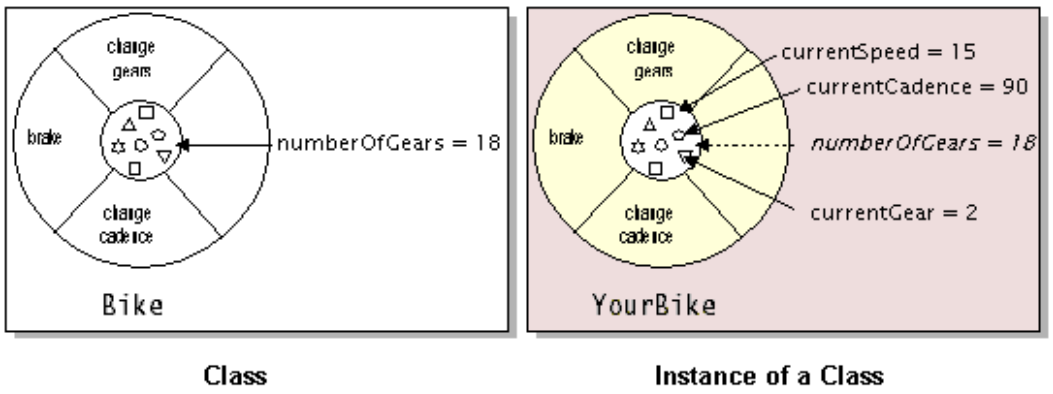
Langkah pertama dalam penggunaan OOP adalah bagaimana cara mengidentifikasi semua *object-object* yang ingin dimanipulasi, dan bagaimana *object-object* tersebut saling berhubungan satu sama lain. Langkah tersebut sering juga disebut dengan *data modeling*. Setelah semua *object-object* telah diidentifikasi, maka langkah selanjutnya adalah membuat generalisasi dari *object-object* tersebut menjadi sebuah *class*.




Gambar 2.2. Representasi *visual* sebuah *object* dengan model sepeda

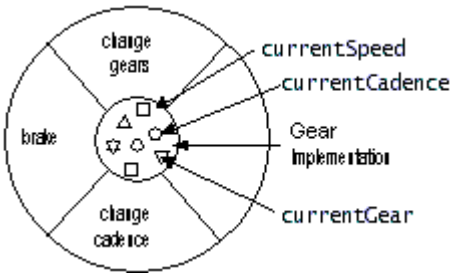
*Class* adalah *prototype* dari sebuah *object*, yang berfungsi untuk mendefinisikan tipe-tipe data yang dibutuhkan dan instruksi-instruksi yang

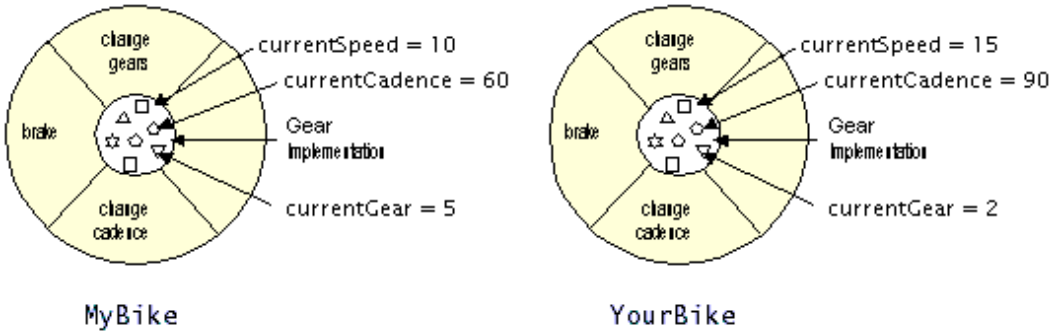
diperlukan untuk memanipulasi data-data yang terdapat pada *class* tersebut. Sedangkan o*bject* adalah sebuah *instance* (hasil produksi) dari *class*.




Gambar 2.3. Representasi *visual* dari c*lass* dan o*bject*

Untuk memudahkan pengertian tentang *class* dan *object*, bayangkan analogi berikut ini: bila sebuah pabrik sepeda ingin memproduksi sepeda jenis tertentu dengan jumlah banyak, tentu antara sepeda yang satu dan sepeda yang lain tidak akan memiliki karakteristik yang sama persis, maka tidak efisien bila dibuat satu *blueprint* tiap satu sepeda yang akan diproduksi. Tentu akan dibuat hanya satu *blueprint*, dan tiap-tiap sepeda yang diproduksi akan didasarkan pada *blueprint* tersebut (disebut *instance*). Hubungan antara *class* dan *object* mirip dengan hubungan antara *blueprint* dan sepeda. Yang berfungsi sebagai *class* adalah rancangan *blueprint*, sedangkan sepeda yang merupakan hasil fisik merupakan analogi dari *object*.





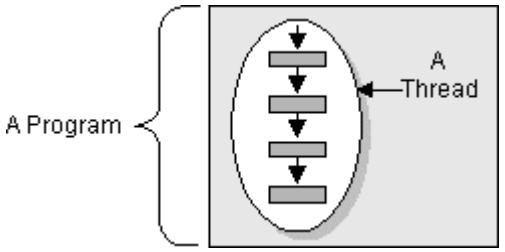

Gambar 2.4. Representasi *visual* sebuah *class* dan *object* hasil *instance*-nya

Beberapa keuntungan yang diperoleh dengan menggunakan *Object Oriented Programming*:

a. Adanya konsep *class* memungkinkan untuk mendefinisikan *sub-classes* dari *object-object* data yang memiliki sebagian atau semua persamaan data dengan *class* utama. Metode ini dinamakan *inheritance*.
b. Karena sebuah *class* hanya mendefinisikan data-data yang dibutuhkannya, maka ketika satu *instance* dari *class* tersebut (sebuah *object*) dibuat, maka instruksi-instruksi didalamnya tidak akan bisa diakses oleh program lainnya secara tidak sengaja. Karakteristik penyembunyian data dalam *class* ini memberikan pengamanan sistem yang lebih baik dan dapat menghindari *corruption* data yang tidak diinginkan.
c. Definisi dari sebuah *class* tidak hanya bisa digunakan oleh program induknya sendiri, namun juga bagi program berbasis OOP lainnya. Misalnya dalam penggunaan *library*.
d. Konsep *inheritance* mengefisienkan kerja *programmer* karena mampu membuat *class* baru sesuai kebutuhan dengan cepat.

## 2.2. *Thread*

Pada jaman awal perkembangan komputer, semua pemrograman komputer pada dasarnya hanya memiliki satu *thread* saja. Semua *processing* bersifat *sequential* dan seorang *user* memperoleh semua *processing time* milik komputer secara eksklusif.




Gambar 2.5. Program yang hanya memiliki satu *thread*

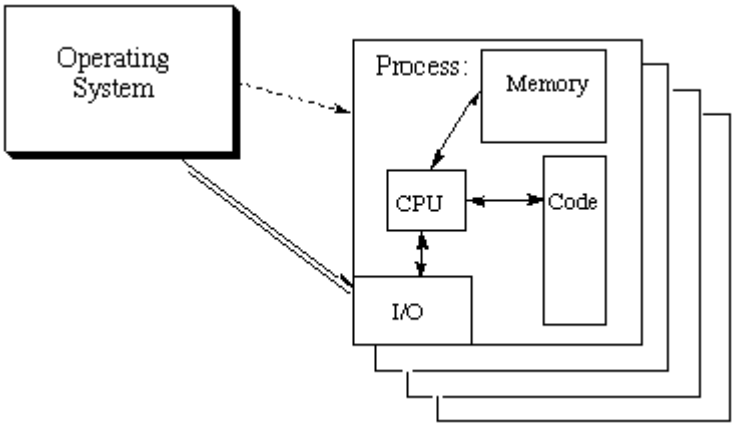
Dengan berkembangnya teknologi komputer, konsep *multiple threads of execution* atau *multithreading* pertama kali muncul pada sistem yang berbasis *time sharing*, dimana lebih dari satu *user* bisa *log on* ke satu *mainframe* pusat pada satu waktu. Hal yang penting untuk diperhatikan pada sistem seperti itu adalah bagaimana cara untuk memastikan bahwa *processing time* bisa dibagi secara rata antara semua *user*. Untuk mengatasinya, *operating system* pada jaman itu mengembangkan dan mengimplementasikan konsep *process*, *thread* dan *multithreading.*

Dalam 15 tahun terakhir ini PC juga mengalami perkembangan ke arah yang sama. Disk Operating System (DOS) dan Microsoft Windows sebagai o*perating system* populer pada jaman-jaman awal PC masih bersifat *single tasking*. Program milik *user* jalan secara eksklusif pada suatu komputer, atau tidak sama sekali. Dengan berkembangnya aplikasi-aplikasi modern yang semakin rumit, dan adanya *demand* yang tinggi atas performa PC, *operating system* yang berbasis *multiprocess* dan *multithreading* kini umum dijumpai di area PC. Perkembangan tersebut dimotori oleh kebutuhan lebih akan performa dan *usability*.

Supaya lebih mudah dalam memahami konsep *thread*, maka konsep pertama yang perlu didefinisikan adalah *process.* Pada operating system seperti Windows 95, 98, dan NT atau *Nix (versi-versi *variant* Unix), sebuah *process* adalah sebuah program yang berjalan secara bersamaan dan dieksekusi secara paralel dengan program-program lainnya pada satu komputer.

Dari sudut pandang seorang *programmer*, sebuah *process* merupakan satu eksekusi dari code, dimana code tersebut memiliki satu eksistensi unik dalam sebuah sistem, dan instruksi-intruksi yang dieksekusi oleh *process* tersebut dilakukan secara berurutan dan teratur. Pada keseluruhan, tiap-tiap *process* dieksekusi secara terisolasi dari *process* lainnya. Tiap-tiap *process* memiliki satu

set *virtual resource* yang hanya milik *process* tersebut sendiri, tidak tersentuh oleh *process-process* lainnya. Virtualisasi *resource* tersebut diatur oleh operating system.



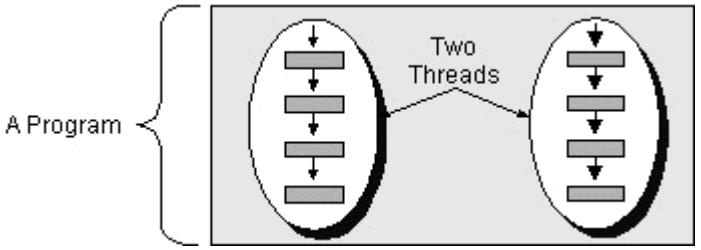

Gambar 2.6. Virtualisasi *resource* sebuah *process* dalam *operating system*

Pada sebagian besar sistem yang mendukung *multiprocessing* (sering disebut *multitasking*) dan *multithreading*, jumlah fisik *hardware* prosesor lebih sedikit daripada jumlah *process* atau *thread* yang berjalan secara paralel. Karena itu sebagian besar sistem mendukung metode *time-slicing* / *preemptive multitasking* untuk mengatasi hal ini. Yang dimaksud dengan *time-slicing* tersebut adalah setiap *process* / *thread* dieksekusi hanya sebentar saja per satuan waktu *time-slice*, kemudian *operating system* akan mengevaluasi *process* / *thread* mana yang akan dilanjutkan eksekusinya untuk *time-slice* berikutnya. Dengan metode ini sebuah *processor* bisa menjalankan banyak *thread* secara bergantian, namun dari sisi pandang *user*, semua *thread* berjalan secara bersamaan. Pada sistem PC, biasanya panjang sebuah *time-slice* adalah 55 milidetik.

Konsep *thread* dan *multithreading* dikembangkan karena kebutuhan *programmer* untuk mengembangkan aplikasi yang bisa melakukan berbagai fungsinya dengan lebih bebas dan tidak terikat oleh satu jalur eksekusi. Pada situasi dimana satu fungsi akan menimbulkan *delay* yang cukup lama (misal: menunggu *user* melakukan sesuatu), akan lebih efisien bila aplikasi bisa melakukan suatu fungsi lainnya secara paralel (misal: *background spell checking*, memproses *packet-packet* dari jaringan). Namun bila paralelisme tersebut diimplementasikan dengan menggunakan konsep *process*, *overhead* pembuatan

satu *process* baru untuk tiap fungsi dan mengatur tiap-tiap *process* untuk bisa saling berkomunikasi sering dirasa terlalu besar.

*Thread* merupakan bagian dari program yang bisa dieksekusi secara terpisah dan berjalan paralel dari program utama. Dengan menggunakan teknik *multithreading* suatu program bisa memiliki lebih dari satu alur *processing* sehingga bisa mengoptimalkan performa sebuah aplikasi komputer.




Gambar 2.7. Program yang menggunakan *multithreading*

Di sisi negatifnya, penggunaan *multithreading* akan menambah kompleksitas dari pembuatan sebuah aplikasi, karena dengan adanya dua atau lebih *thread* dalam satu aplikasi berarti akan muncul problema baru, yaitu proses sinkronisasi dan komunikasi antar *thread* yang sedang berjalan.

Konsep *multithreading* ini digunakan oleh aplikasi *Automated File Distributor* (AFD) untuk mengimplementasikan fitur *Simultaneous Transfers*, dimana aplikasi memerlukan banyak *thread* untuk bisa menjalankan banyak proses *transfer file* secara sekaligus.

## 2.3. Internet Protocol Helper API

Internet Protocol (IP) Helper API adalah suatu *application programming interface* (API) yang dibuat oleh Microsoft. Biasa disebut IPHLPAPI, API ini disertakan dalam versi Microsoft Windows NT4, 2000 dan XP. Sayangnya IP Helper API ini tidak disertakan di Windows 98 kebawah.

IP Helper API menyediakan banyak fungsi-fungsi yang berhubungan dengan informasi dan konfigurasi jaringan komputer yang menggunakan protokol TCP/IP, di antaranya sebagai berikut:

a. Mengambil informasi mengenai konfigurasi jaringan
b. Mengatur *Network Adapters*
c. Mengatur *Interfaces*
d. Mengatur alamat-alamat IP

e. Penggunaan *Address Resolution Protocol*
f. Mengambil informasi mengenai Internet Protocol (IP) dan Internet Control Message Protocol (ICMP)
g. Mengatur jalur *routing*
h. Menerima notifikasi akan *event-event* jaringan
i. Menerima informasi akan Transmission Control Protocol (TCP) dan User Datagram Protocol (UDP)

Aplikasi *Automated File Distributor* (AFD) ini menggunakan IP Helper API untuk sebagai media untuk fitur *Network Traffic Monitoring*, yaitu memperoleh informasi mengenai kondisi *thoughput* jaringan, baik jumlah *packet-packet* yang masuk maupun keluar.

*Function-function* penting dari IP Helper API yang dipakai oleh aplikasi AFD adalah (*function* dalam bahasa Delphi):

a. *Function* Get_IPAddrTableMIB(IPAddrTable: TMibIPAddrArray)
   Berfungsi untuk mengambil data IP *address* dari *network interface* yang sedang aktif di-*monitor* oleh aplikasi.
b. *Function* Get_IfTableMIB (MIBIfArray: TMIBIfArray)
   Berfungsi untuk mengambil data mengenai koneksi pada saat itu.

## 2.4. Win32 Application Programming Interface (API)

Win32 API adalah sekumpulan *function*, *structures*, *messages*, *macros*, dan *interfaces* yang bisa digunakan oleh *programmer* dalam membuat aplikasi yang berbasis pada semua *platform* 32-bit dari Microsoft.

Win32 API bisa dikelompok-kelompokkan menjadi beberapa kategori fungsi, yaitu:

a. *Window Management*
b. *Window Controls*
c. *Shell Features*
d. *Graphics Device Interface*
e. *System Services*
f. *International Features*
g. *Network Services*

Beberapa fungsi Win32 API digunakan dalam pembuatan aplikasi *Automated File Distributor* (AFD) yang dibahas pada laporan ini. Beberapa fungsi penting yang digunakan adalah (*function* dalam bahasa C++):

a. *Function* SHBrowseForFolder(LPBROWSEINFO lpbi)
   Berfungsi untuk menampilkan dialog "*Browse for folder*" yang digunakan dalam implementasi fitur *Explorer* dari AFD.

b. *Structure* _browseinfo{HWND hwndOwner; LPCITEMIDLIST pidlRoot; LPSTR pszDisplayName; LPCSTR lpszTitle; UINT ulFlags; BFFCALLBACK lpfn; LPARAM lParam; int iImage;
   *Structure* (*record* pada Delphi) yang berfungsi sebagai parameter eksekusi. Digunakan oleh *function* SHBrowseForFolder.

Tabel 2.1. Struktur dari LPBROWSEINFO

| **Nama variabel** | **Deskripsi** |
|---|---|
| hwndOwner | Handle of the owner window for the dialog box |
| pidlRoot | Pointer to an item identifier list (an ITEMIDLIST structure) specifying the location of the "root" folder to browse from. Only the specified folder and its subfolders appear in the dialog box. This member can be NULL, and in that case, the name space root (the desktop folder) is used |
| pszDisplayName | Pointer to a buffer that receives the display name of the folder selected by the user. The size of this buffer is assumed to be MAX_PATH bytes. |
| lpszTitle | Pointer to a null-terminated string that is displayed above the tree view control in the dialog box. This string can be used to specify instructions to |

| | |
|---|---|
| | the user. |
| ulFlags | Value specifying the types of folders to be listed in the dialog box as well as other options. This member can include zero or more of the following values:<br>BIF_BROWSEFORCOMPUTER<br>BIF_BROWSEFORPRINTER<br>BIF_DONTGOBELOWDOMAIN<br>BIF_RETURNFSANCESTORS<br>BIF_RETURNONLYFSDIRS<br>BIF_STATUSTEXT |
| lpfn | Address an application-defined function that the dialog box calls when events occur. For more information, see the description of the BrowseCallbackProc function. This member can be NULL. |
| lParam | Application-defined value that the dialog box passes to the callback function (if one is specified). |
| iImage | Variable that receives the image associated with the selected folder. The image is specified as an index to the system image list. |

c. *Function* sndPlaySound(LPCSTR lpszSound, UINT fuSound)

   Berfungsi untuk menjalankan suatu file *audio* yang digunakan dalam implementasi fitur *Audio Notification* dari AFD.

d. *Function* GetFileVersionInfo(LPTSTR lptstrFilename, DWORD dwHandle, DWORD dwLen, LPVOID lpData)

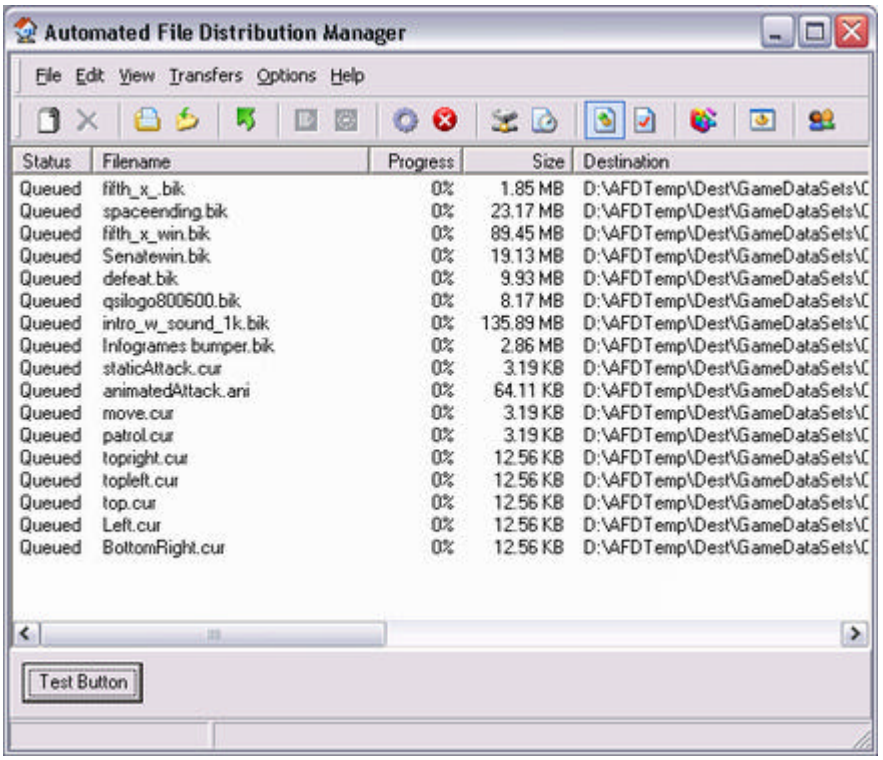
   Berfungsi untuk mengambil informasi versi sebuah aplikasi yang digunakan

*Function-function* diatas akan dibahas dengan lebih lengkap pada bab 3 dari laporan ini.

### 2.5. Microsoft Windows XP Theme Support

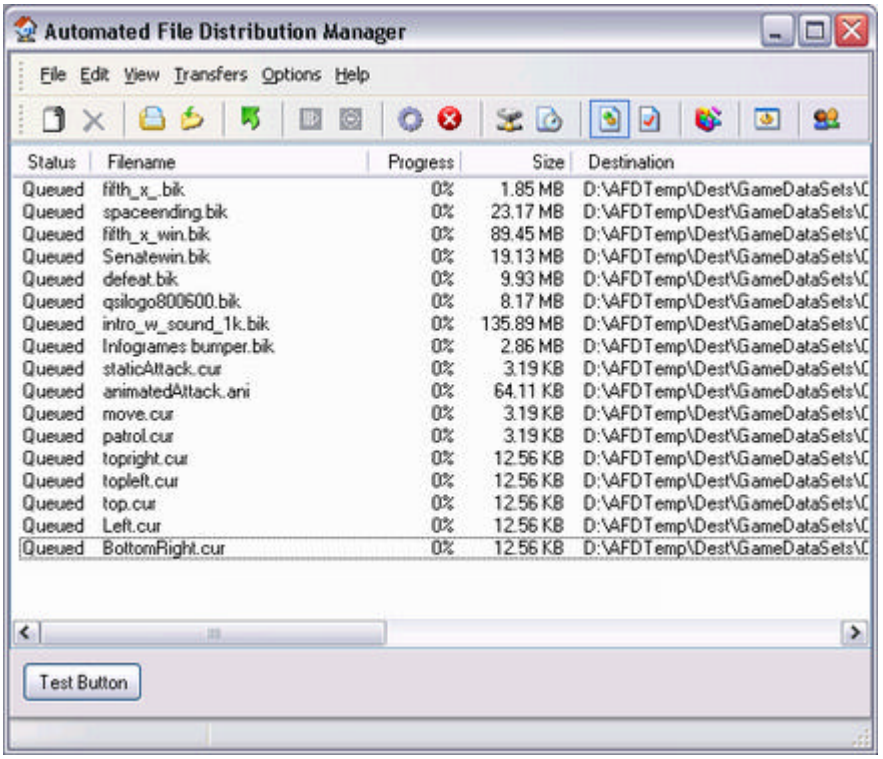
Microsoft membagi Windows Common Controls (juga sering disebut COMCTL) menjadi dua versi yang berbeda. *Controls* adalah *object-object visual* dalam Windows, contohnya *Window*, *Button*, *Title Bar*, *Dialogs*, *Main menu*, dan lain-lain.

Common Controls versi 5 adalah versi lama yang bisa dijumpai di semua versi Windows mulai dari Windows 95, yang menampilkan *controls* dengan menggunakan tampilan "*3D chiseled*".




Gambar 2.8. Tampilan aplikasi yang menggunakan Common Controls versi 5

Sedangkan *Common Controls* versi 6 mulai dikenalkan oleh Microsoft pada Windows XP. Pada versi 6 ini, tampilan *controls* ditampilkan dengan menggunakan sebuah *theme engine* sesuai dengan yang *theme* yang dipilih oleh *user*. Bila *user* mengganti *theme* yang aktif, maka secara otomatis semua tampilan *controls* akan diubah sesuai dengan *theme* yang baru dipilih.




Gambar 2.9. Tampilan aplikasi yang menggunakan Common Controls versi 6

Sebagai *default*, aplikasi yang dikembangkan dengan menggunakan Delphi akan menampilkan *controls* sesuai *common controls* versi 5 karena tingkat kompatibilitasnya yang lebih tinggi. Supaya aplikasi bisa menggunakan Common Controls versi 6 tanpa kompilasi ulang, maka harus ditambahkan sebuah *manifest file* yang sesuai dengan spesifikasi dari Microsoft.

Sebuah *manifest file* adalah sebuah *file* berformat XML yang berisi *dependencies* dari aplikasi yang hendak dirubah. *Manifest file* harus ditaruh di satu *directory* yang sama dengan aplikasi, dan nama *manifest file* sendiri harus sama dengan nama *executable* dari aplikasi, namun dengan menambahkan *extension* ".manifest" pada akhir nama *file*. Sebagai contoh, bila nama aplikasinya adalah "AFD.exe", maka *manifest file* harus diberi nama "AFD.exe.manifest"

Berikut ini adalah contoh sebuah *manifest file* untuk menginstruksikan aplikasi supaya menggunakan Common Controls versi 6:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
  version="1.0.0.0"
  processorArchitecture="X86"
  name="CompanyName.ProductName.YourApp"
  type="win32"
/>
<description>Your application description here.</description>
<dependency>
  <dependentAssembly>
    <assemblyIdentity
      type="win32"
      name="Microsoft.Windows.Common-Controls"
      version="6.0.0.0"
      processorArchitecture="X86"
      publicKeyToken="6595b64144ccf1df"
      language="*"
    />
  </dependentAssembly>
</dependency>
</assembly>
```

## 2.6. Borland Delphi

Delphi adalah suatu aplikasi dan bahasa pemrograman berbasis bahasa Pascal yang dikembangkan oleh Borland. Pada saat penulisan laporan ini, Delphi 7.0 for .NET adalah versi yang paling baru dirilis.

Delphi memiliki keunggulan-keunggulan dibanding bahasa pemrograman lainnya, diantaranya adalah:

a. Menggunakan bahasa pemrograman yang bersifat *high-level language.* Terstruktur dengan jelas dan mudah dimengerti.
b. Merupakan aplikasi pemrograman yang bersifat *Rapid Application Deployment* (RAD) yang berarti Delphi difokuskan untuk bisa membuat aplikasi-aplikasi komputer dengan cepat dengan menggunakan *Object Oriented Programming*.
c. Penggunaan yang *user friendly*, dan fasilitas dokumentasi yang tersedia dengan lengkap dan terstruktur dengan rapi.
d. Banyaknya *component-component* yang tersedia di *internet* mampu memperluas kemampuan Delphi.

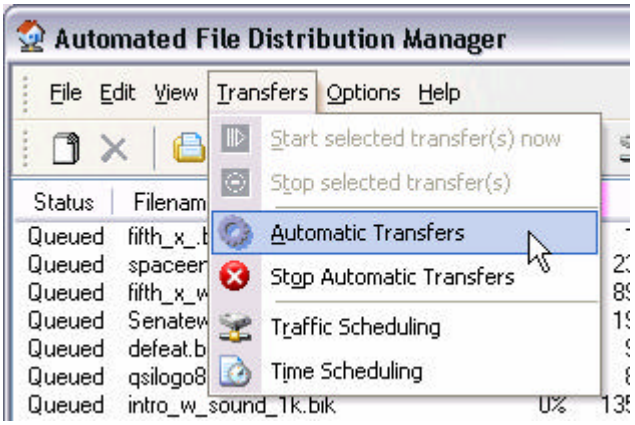
Delphi 7.0 for .NET menambahkan fitur-fitur dan *component-component* baru yang digunakan dalam pengembangan aplikasi *Automated File Distributor* (AFD), yaitu:

a. *Action Manager* (TActionManager)

*Component* ini didesain untuk menggantikan *component Action List* (TActionList) yang sudah mulai kadaluwarsa. Fungsi utama *component* ini adalah untuk menyimpan satu daftar *action-action* secara sentral agar *action-action* tersebut lebih mudah untuk diatur. Fungsi lainnya dari *component* ini adalah untuk memberi tampilan *menu* dan *toolbar* yang lebih baru. *Menu* dan *toolbar* baru tersebut menggunakan tampilan *button* yang datar, mirip seperti yang digunakan Microsoft pertama kali pada Microsoft Office 2000. Untuk itu, c*omponent* yang ini didesain untuk digunakan bersama *Action Main Menu Bar* dan *Action Toolbar*. Selain itu, *Action Manager* bisa melacak seberapa sering penggunaan *user interface* (UI) dan bisa menyembunyikan UI yang jarang digunakan oleh *user*.

b. *Action Main Menu Bar* (TActionMainMenuBar)

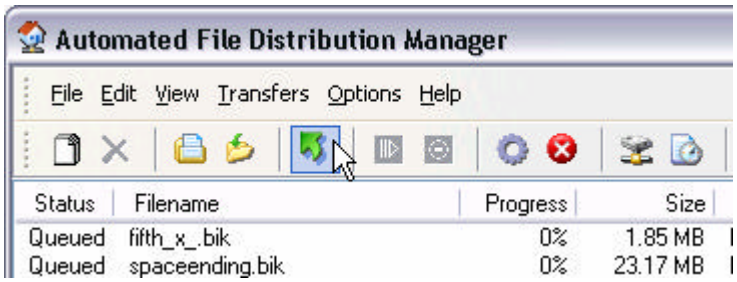
Biasa digunakan bekerjasama dengan *Action Manager*, perbedaan utama *component* ini dengan *Main Menu* (TMainMenu) biasa adalah tampilannya yang mirip seperti tampilan aplikasi Microsoft Office 2000 keatas. Dengan bekerjsama dengan *Action Manager*, tampilan *menu* atau *user interface* yang jarang digunakan dapat disembunyikan secara otomatis oleh aplikasi dengan melacak frekuensi penggunaan *menu* atau *user interface* tersebut.




Gambar 2.10. Contoh tampilan *Action Main Menu Bar*

c. *Action Toolbar* (TActionToolbar)

Biasa digunakan bekerjasama dengan *Action Manager*, perbedaan utama *component* ini dengan *Toolbar* (TToolbar) biasa adalah tampilannya yang mirip seperti tampilan aplikasi Microsoft Office 2000 keatas.




Gambar 2.11. Contoh tampilan *Action Toolbar*