Windows API Kurs Notları

Kaan ASLAN

C ve Sistem Programcıları Derneği

Güncelleme Tarihi: 04/10/2003

WINDOWS SİSTEMLERİNİN TARİHSEL GELİŞİMİ

Windows'un ilk versiyonu 1985 yılında çıkmıştır. Bu 1.0 versiyonu çok kısıtlı ölçüde kullanılmıştır. Daha sonra 1987 yılında 2.0, 1988 yılında da 3.0 versiyonları piyasaya sürülmüştür. O yıllarda Microsoft'un DOS işletim sistemi aktif olarak kullanılıyordu. Windows, DOS altında çalışan bir utility program gibiydi. Windows API programlamanın temelleri ağırlıklı olarak 3.0 versiyonu ile oluşturulmuştur. Fakat Windows'un en fazla kullanılan versiyonu 80'li yılların sonlarına doğru piyasaya sürülmüş olan 3.1 versiyonudur. Windows 3.1 de bir işletim sistemi değildi. Fakat 286 ve 386 işlemcilerde korumalı moda geçerek çalışıyordu. Windows 3.1, temel olarak DOS'u kullanan fakat DOS'un olanaklarını korumalı mod yoluyla arttıran işletim sistemi denilmese bile işletim sistemi olmaya aday bir programdı. Windows 3.1, 16 bit bir çalışma sunuyordu.

Microsoft, 1992-93 yıllarında Windows NT (**N**ew **T**echnology) adı altında ilk 32 bit gerçek Windows işletim sistemini piyasaya sürmüştür. Windows NT, DOS uyumu zayıf olan bir işletim sistemiydi. Oysa o yıllarda DOS programları oldukça yaygın bir biçimde kullanılıyordu. DOS programlarını çalıştıramayan yani DOS uyumluluğu yüksek olmayan işletim sistemlerinin tutulması zor gözüküyordu. Bu nedenle Microsoft, Windows'un DOS uyumu yüksek yeni bir 32 bitlik versiyonunu 95'te piyasaya sürmüştür. Windows 95 DOS'u neredeyse tamamen destekleyen 32 bit bir işletim sistemidir. Windows 95'ten sonra 98 yılında Windows 98 ve 2000 yılında Windows ME ve 2000 sistemleri çıkmıştır. Bunu yaygın son sürümü olan XP izlemiştir.

Windows sistemleri 16 bit ve 32 bit sistemler olmak üzere 2'ye ayrılır. Bugün 16 bit Windows sistemleri tamamen kullanımdan kalkmıştır. Win16 sistemleri Windows 3.x sistemleridir. Win32 sistemleri ise Windows NT, Windows 95, Windows 98, Windows ME, Windows 2000 ve Windows XP sistemleridir.

Win32 sistemleri koruma mekanizmasının katılığına göre ve çekirdek yapısına göre 95 grubu sistemler ve NT grubu sistemler olmak üzere 2'ye ayrılır. 95 grubu sistemler Windows 95 98 ve ME sistemleridir. Bu sistemlerin DOS uyumu çok yüksektir yani DOS programlarını tamamen çalıştırabilirler. Bu sistemlerde katı bir koruma uygulanmamıştır. Oysa NT grubu sistemlerin DOS uyumu zayıftır ve bu sistemlerde katı bir koruma uygulanmıştır. Genel olarak NT grubu sistemlerin 95 grubu sistemlere göre daha kararlı olduğu söylenebilir.

API programlama modeli Win32 sistemlerin hepsinde büyük ölçüde aynıdır. Fakat bazı sistem özellikleri 95 grubu sistemlerde olmadığı halde NT grubu sistemlerde söz konusu olmaktadır.

WIN32 SİSTEMLERİNİN TEMEL ÖZELLİKLERİ

1. **Çok İşlemlilik:** Win32 sistemleri aynı anda birden fazla programın çalıştırılabildiği çok işlemli sistemlerdir.

Anahtar Notlar:

Çalışabilen dosyanın diskteki durumuna program denilmektedir. Bir program çalıştırıldığında işletim sistemi pek çok düzenlemeler yapar. Çalışmakta olan programlara process yada task denilmektedir.

Çok işlemli sistemlerde prosesler parçalı olarak zaman paylaşımlı bir biçimde çalıştırılmaktadır. Yani bir proses bir süre çalıştırılır, sonra çalışmasına ara verilir, başka bir proses çalıştırılır. Sonra sıra ilk prosese geldiğinde çalışma kalınan yerden devam eder. Bir prosesin parçalı çalıştırılma süresine "quanta süresi" denilmektedir. Win32 sistemlerinde quanta süresi tipik olarak 20ms'dir.

Çok işlemli sistemlerde programın iki noktası arasında geçen gerçek zaman, sistemin o anki durumuna göre farklı olabilir. Kabaca sistemde ne kadar çok proses çalışıyor durumda ise programımızın çalışması yavaşlamaktadır.

Bazı makinelerde birden fazla işlemci bulunabilir. Bu mimariye **SMP** (Switch Mode) denilmektedir. Bu durumda prosesleri işlemciler paylaşarak daha hızlı çalıştırırlar. Windows 95, 98 ve ME versiyonları çok işlemcili sistemleri desteklememektedir. Ama NT, 2000 ve XP; çok işlemcili bilgisayarlarda çalışabilmektedir.

2. **Grafik Tabanlı Çalışma:** Windows sistemleri çekirdekle entegre edilmiş grafik bir ara birime sahiptir. Yani işletim sistemi bir pencere sistemi göz önünde bulundurularak tasarlanmıştır. Oysa örneğin UNIX/LINUX sistemlerinde grafik çalışma pencere yöneticileri denilen çekirdekle entegre edilmemiş programlar tarafından sağlanmaktadır. Genel olarak Windows sistemlerinin grafik işlemleri UNIX/LINUX sistemlerinin grafik işlemlerinden daha hızlıdır.

3. **GUI ve Console Uygulamaları:** GUI (Graphical User Interface) uygulaması, pencereye sahip olan gerçek Windows uygulamasıdır. GUI uygulamalarında mesaj tabanlı bir programlama modeli kullanılmaktadır. Oysa Console uygulamaları siyah ekranda çalışan klasik programlardır. Yani console uygulamaları için yine DOS'ta ve UNIX/LINUX sistemlerindeki programlama modeli kullanılır. Console uygulamaları görünüşü ve ara birimi önemli olmayan fakat hızlı çalışması istenen programlarda tercih edilir.

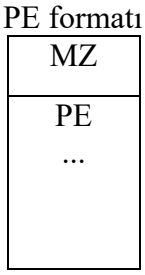
4. **DOS Uyumu:** Win32 sistemleri Intel mimarisinde DOS programlarını çalıştıracak biçimde tasarlanmışlardır. Windows 95, 98 ve ME sistemleri hemen hemen tüm DOS programlarını çalıştırırken Windows NT, 2000 ve XP sistemleri yalnızca bazı programları çalıştırabilmektedir.

Windows 98 sisteminde bir DOS programın çalıştırdığımızı düşünelim. Bu program da zaman paylaşımlı olarak diğer Windows programları ile birlikte çalıştırılacaktır. Fakat çalışma zamanı DOS programına geldiğinde mikroişlemcinin çalışma modu geçici süre korumalı moddan V86 (Virtual 86) moduna geçirilir. V86 modu, Pentium işlemcilerinin 8086 işlemcisi gibi çalıştığı bir moddur. Bilindiği gibi 8086 işlemcisi 1MB belleği kullanabilen 16 bit bir işlemcidir. DOS programları, 8086 mikroişlemcisi ile çalışabilecek yapıya sahiptir. Yani eğer mikroişlemcinin modu geçici bir süre V86'ya geçemeseydi DOS programları ile Windows programları birlikte çalışamazdı. Bu sistemlerde farklı DOS programları adeta aynı anda farklı DOS makinelerinde çalışıyor gibi bir durum olmaktadır.

En çok karıştırılan konu DOS prompt'u kavramı ile ilgilidir. 95 grubu sistemlerde DOS prompt'una geçtiğimizde DOS işletim sistemi, işlemcinin modunu değiştirerek ayrı bir proses biçiminde çalıştırmaktadır. Yani bu sistemlerde DOS prompt'u bir DOS programıdır. Halbuki NT grubu sistemlerinin prompt'unun DOS ile ilgisi yoktur.

DOS prompt'unda bir Windows console programı çalıştırırsak ne olur? İşte Windows içerisindeki DOS .exe formatına bakar; bu bir Windows console programı ise yeni bir console ekranı açmadan işlemcinin modunu değiştirerek programı çalıştırır. Yani özetle 95 grubu sistemlerin DOS prompt'unda biz Windows programlarını da çalıştırabiliriz.

5. **Çalışabilir ve Amaç(Object) Dosya Formatları:** DOS'ta kullanılan .exe formatına "MZ" formatı denilmektedir (MZ, bu formatı tasarlayan Mark Zibikovski'den yapılmış bir kısaltmadır). Windows 3.1 sistemlerine geçildiğinde NE (New Executable) denilen başka bir format kullanılmaya başlamıştır. Gerçekten de Win16.exe dosyalarının başı NE harfleriyle başlamaktadır. Nihayet Win32 sistemlerine geçildiğinde .exe dosya formatı da yenilenerek PE (Portable Executable) formatına geçilmiştir. Bir PE formatı isteğe bağlı olarak küçük bir MZ programına da sahip olabilir.



DOS'ta kullanılan .obj dosya formatına OMF (Object Module Format) deniliyordu. Win16 sistemlerinde de 16 bit OMF formatı kullanılmıştır. fakat Win32 sistemlerine geçildiğinde .obj dosya formatı değiştirilerek COFF (Common Object File Format) formatına geçilmiştir. COFF formatının orijini UNIX sistemleridir. Win32 formatı UNIX COFF formatından oldukça farklıdır.

Anahtar Notlar:

Intel tabanlı PC'lerde çalışan UNIX türevi sistemler eskiden çalışabilen dosya formatı olarak a.out formatı kullanıyordu. Yine bu sistemlerin çoğu amaç dosya formatı olarak 32 bit OMF formatını kullanıyordu. Daha sonraları ELF (Executable and Linkable Format) formatı yaygınlaşmıştır. Örneğin bugün LINUX sistemlerinde amaç dosya formatı olarak ve çalışabilen dosya formatı olarak ELF formatı kullanılmaktadır.

---

6. **Mesaj Tabanlı Programlama Modeli:** Klasik console tabanlı uygulamalarda fare kullanımı yaygın değildir ve girdi oluşturan pek az olay vardır. Programcı bu sistemlerde bir olayın gerçekleşip gerçekleşmediğini bir fonksiyonu çağırarak tespit eder. Olay gerçekleşmişse uygun işlemleri yapar. Örneğin hem fareyi hem klavyeyi izleyebilmek için bir döngü içerisinde hiç beklemeden bu olayların gerçekleşip geçekleşmediğine bakmak gerekir. Bu da çok yorucu bir işlem gerektirmektedir.

Mesaj tabanlı programlama modelinde programcı girdi olaylarını kendisi izlemez. Bu olaylar ilk elden işletim sistemi tarafından alınır ve hangi programa ilişkinse o programa mesaj olarak verilir. Mesaj, teknik olarak bir yapıdır. Örneğin yapının bir elemanında girdi olayının ne olduğu; başka bir elemanında, o olaya ilişkin bilgiler bulunmaktadır. Olay oldukça sistem, mesajı programcıya nasıl iletmektedir? Sistemin yaptığı tek şey mesajı bir FIFO (First In First Out) kuyruk sistemine eklemektir. Programcı, bir döngü içerisinde bu kuyruk sistemine bakmalı; eğer bir olay oluşmuşsa o olayın bilgilerini kuyruk sisteminden alarak işleme

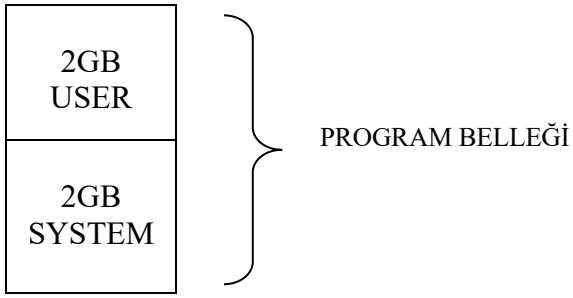
sokmalıdır. Yani program bütün yaşamını kuyruktan mesajı alan ve onu işleyen bir döngüde geçirir. Bu döngüye mesaj döngüsü denilmektedir.

7. **API Fonksiyonları:** İşletim sistemleri fonksiyonel bir biçimde yazılmış programlardır. Bazı fonksiyonlar hem işletim sisteminin çalışması sırasında sistem tarafından hem de programcı tarafından kullanılabilmektedir. Böyle fonksiyonlara UNIX dünyasında sistem fonksiyonları; Windows dünyasında ise API (Application Programming Interface) fonksiyonları denilmektedir. API fonksiyonları, işletim sisteminin bir parçası olarak bulunmaktadır.

Win32 sistemlerinde temelde 4 grup API fonksiyonu vardır.

1) ***Kernel API Fonksiyonları:*** Bu fonksiyonlar KERNEL32.DLL dosyası içerisindedir ve sistemin çekirdek kısmı ile ilgili işlemlere yöneliktir.
2) ***User API Fonksiyonları:*** Bu fonksiyonlar USER32.DLL dosyası içerisindedir ve temel olarak pencere işlemleri ile ilgili fonksiyonlardır.
3) ***GDI API Fonksiyonları:*** Bu fonksiyonlar GDI32.DLL içerisindedir ve grafik işlemleri ile ilgilidir.
4) ***Diğer API Fonksiyonları:*** Yukarıdakilerin dışında çeşitli değişik konularda birtakım sistem DLL'leri içerisinde bulunan fonksiyonlardır.

8. **Bellek Yönetimi:** Win32 sistemleri sanal bellek (virtual memory) kullanabilen sistemlerdir. Sanal bellek RAM'in yetmediği durumlarda programın disk ile RAM arasında yer değiştirmeli bir biçimde çalıştırılmasına yönelik bir mekanizmadır. Programcı programını sanki tek parça belleğe yüklenecekmiş gibi yazar. Sanal bellek, işletim sisteminin kendisinin arka planda sağladığı bir mekanizmadır.



Win32 sistemlerinde proseslerin bellek alanları birbirinden tamamen izole edilmiştir. Her proses, çalışma sırası kendisine geldiğinde 4GB'lik belleğin düşük anlamlı 2GB'lik alanına yüklenerek çalıştırılır. (2GB user, 2GB system)

Yüksek anlamlı 2GB'lik alan, işletim sisteminin bulunduğu sistem alanıdır. Bu alan prosesler arası geçişten etkilenmez. Görüldüğü gibi bir proses çalışırken diğer prosesler bellekte değildir. Her proses 2GB'ye kadar büyüyebilir. Örneğin bir prosesin 2GB'lik alanında bir adres bölgesi düşünelim. Başka bir prosesin aynı bölgesinde başka değerler bulunacaktır. Yani biz, bir gösterici hatası yaparak başka bir prosesin bellek alanına erişemeyiz.

Win32 sistemlerinde tüm göstericiler 4 byte uzunluğundadır. Bu sistemler Flat model kullandıkları için segment kavramına gereksinim duymazlar. 4 byte'lık göstericilerle 4GB'lik bellek alanının her yerine erişilebilmektedir.

Win32 sistemlerinde doğal olarak heap alanı da çok geniştir. Örneğin programcı malloc fonksiyonuyla 100'lerce MegaByte'lık alanı tahsis edebilir. Bu sistemlerde ciddi bir bellek sıkıntısı yaşanmamaktadır.

9. **Koruma Mekanizması:** Win32 sistemleri Intel işlemcilerinin korumalı modunda çalışırlar. Koruma mekanizması bir programın, sistemin geneline ve başka programlara zarar vermesini engelleyen bir mekanizmadır.

Koruma mekanizması temel olarak mikroişlemci tarafından sağlanan bir mekanizmadır. Yani bir koruma ihlali olduğunda bu daha ilk elden mikroişlemci tarafından tespit edilir ve işletim sistemine bildirilir.

Koruma mekanizması şu konularda bir koruma faaliyeti oluşturmaktadır.

1) ***Bellek Koruması:*** Bir proses kendi bellek alanı dışına erişmeye çalışırsa sonlandırılır.
2) ***I/O Port Koruması:*** Örneğin proses IRQ kesmelerini kapatmak istesin. Bunun için 21H portuna OUT işlemi yaptığı anda proses sonlandırılacaktır.
3) ***Makine Kodu Koruması:*** Bazı makine komutları sistemin tamamen çökmesine neden olabilmektedir. İşte koruma mekanizması altında bu makine komutlarının da kullanımı tamamen yasaklanmıştır.

Intel işlemcilerinde bir prosese 0 ile 3 arasında bir öncelik derecesi verilmektedir. Gerek Win32 sistemleri gerekse UNIX/LINUX sistemleri yalnızca 0 ve 3 öncelik derecelerini kullanırlar. 0 öncelik derecesinde çalışan bir proses hiçbir koruma engeline takılmaz, her şeyi yapabilir. 3 öncelikli prosesler ise çalıştırdığımız sıradan programlardır. Bunlar, yukarıda sözü edilen koruma duvarlarına çarparlar. Şimdi 3 öncelikli bir prosesin (ring 3'te çalışan bir prosesin) bir API fonksiyonunu çağırdığını düşünelim. Bu fonksiyon sistemin kendi içerisindeki birtakım dataların üzerinde güncelleme yapabilir. Peki bu durumda koruma hatası ortaya çıkmaz mı? İşte Intel işlemcilerinde Kapı (Gate) denilen bir mekanizma vardır. Bir fonksiyon kapı yoluyla çağırılırsa otomatik olarak fonksiyon sonlanana kadar prosesin önceliği 0'a (ring 0'a) çekilir. Görüldüğü gibi sistem fonksiyonlarının çalışması süresinde proses ring 0'da kalmaktadır. Şüphesiz kapılar, işletim sistemi tarafından güvenli fonksiyonlara yerleştirilmiştir.

Şüphesiz yine de çeşitli gerekçelerle birtakım özel işlemlerin yapılması gerekebilir. Örneğin bir program bir kartın üzerindeki işlemcileri programlamak isteyebilir. İşte ring 0'da çalışacak olan programlar, aygıt sürücü (device driver) denilen özel bir formatta yazılmalıdır. Aygıt sürücüler adeta işletim sisteminin bir parçasıymış gibi yüklenerek çalışırlar.

10. **Birden Fazla Thread İle Çalışma:** Win32 sistemleri aynı zamanda çok thread ile çalışmaya olanak sağlayan sistemlerdir. Thread, bir programın, sanki başka bir programmış gibi çizelgelenen akışıdır. Bir proses, ana bir thread ile çalışmaya başlatılır. Bu C'deki main fonksiyonunu çalıştıran thread'dir. Diğer thread'ler bir thread akışı içerisinde CreateThread API fonksiyonu ile herhangi bir zaman yaratılabilir. CreateThread fonksiyonunda thread'i yaratırken akışın başlatılacağı fonksiyon adresi verilir. Artık o thread de sanki ana thread'miş gibi diğerinden bağımsız olarak çalışır.

Thread'ler prosesin aynı data bölgesini kullanır. Yani tüm Thread'ler statik ömürlü nesneleri ortak olarak kullanmaktadır. Fakat her thread ayrı bir stack'e sahiptir. Yani thread'lerin stack'leri birbirinden ayrılmıştır. Bu durumda örneğin iki thread aynı fonksiyon üzerinde ilerlese fonksiyonun yerel değişkenlerinin farklı kopyalarını kullanır. Yani yerel değişkenler birbirine karışmazlar.

Win32 sistemlerinin proses yönetimi thread'lere göre çizelgeleme yapmaktadır. Yani her quanta geçişinde bir thread bırakılır, başka bir thread çalıştırılır. Thread'ler arası geçiş aynı iki prosesin thread'leri arasında olabileceği gibi farklı proseslerin thread'leri arasında da olabilir. Thread'ler aynı prosesin bellek alanını kullandığına göre aynı prosesin thread'leri arasındaki geçişte belleğin düşük anlamlı 2GB'lik bölümü değişmez. Fakat farklı prosesin thread'leri arasındaki geçişte bellek üzerinde de geçiş yapılmaktadır.

Thread'ler özellikle arka plan işlemlerin gerçekleştirilmesinde kullanılmaktadır. Örneğin bir program çalışırken aynı zamanda saati ekranda gösterecek olsun. Saatin gösterilmesi için akışın hiçbir yerde takılmaması gerekir. Bu tür işlemleri thread'siz işlemlerde yapmak mümkün olsa bile çok zordur. Oysa

thread'li sistemlerde programcı bir thread yaratır, saatin gösterilmesi işlemini o thread'e yaptırır. Artık gerçek thread, bir yerde beklese bile problem oluşmaz.

11. **Unicode Kullanımı:** (www.unicode.org) Unicode, her bir karakterin 2 Byte ile ifade edildiği yeni bir karakter tablolama sistemidir. Bir metin, unicode olarak yazılırsa metin içerisinde binlerce farklı karakter kullanılabilir. Unicode karakter sisteminde tüm ülkelerin kullandığı karakterler ve birimlerde kullanılan semboller, tabloya çakışmayacak biçimde yerleştirilmiştir. Tabi, metnin unicode yazılmış olması, bunun pürüzsüz bir biçimde görüntüleneceği anlamına gelmez. Görüntüleme işlemi ayrı bir konudur. Görüntüleyecek olan program, karakteri anlamlandırabildiği halde görüntülemeyebilir.

Win32 sistemleri, unicode çalışmayı destekleyen sistemlerdir. Sistemin API fonksiyonları çeşitli yazıları ASCII yada unicode olarak alabilmektedir.

MACAR NOTASYONU

Macar notasyonu, değişkenlerin isimlendirilmesi ile ilgili kurallar topluluğudur. Microsoft, Windows programlamada ağırlıklı olarak bu notasyonu kullanmaktadır. Macar notasyonu, Microsoft çalışanlarından Charles Simongi tarafından tasarlanmıştır.

Macar notasyonunun temel özellikleri şunlardır.

— Fonksiyon isimlerinin her sözcüğünün ilk harfi büyük yazılır. Önce bir eylem, sonra eylemin nesnesi getirilir. Örneğin: CreateWindow, SetWindowText gibi...

— Değişkenler, onların türlerini belirten küçük harf ön eklerle başlatılır. Sonra her sözcüğün ilk harfi büyük harf olacak şekilde devam edilir. Kullanılan önekler şunlardır:

Önek	Tür
l	long
u	unsigned int
c	char
ul	unsigned long int
f	float
d	double
s	short
n	adet, sayı
b	BOOL
f	flag
h	HANDLE
p	gösterici
pl	long *
pi	int *
pv	void *
sz	yazı (zero terminated string)
psz	char *
lp	uzak gösterici
b	BYTE
w	WORD
dw	DWORD

Örneğin: long lNumberOfRecord;

```
float   fWeight;
BOOL bExist;
int     *piCount;
```

Genellikle int türü çok kullanıldığı için bir önek getirilmez. Deve notasyonu (camel casting)'na göre yazılır. Yani ilk sözcüğün tamamı küçük harflerle yazılır; sonraki sözcüklerin yalnızca ilk harfleri büyük yazılır.

Örneğin: int recordFormat;
int count;

Yapılarda önek olarak genellikle yapının typedef ismi yada buna ilişkin kısaltmalar kullanılır.
Örneğin: RECT rectWindow;
POINT ptLeft;
Göstericinin gösterdiği yerde tek bir karakter yok fakat bir yazı varsa psz öneki kullanılır.

Örneğin: char *strcpy(char *pszDest, char *pszSource);

WINDOWS.H DOSYASI

Windows'ta C yada C++ ile program yazarken windows.h dosyası include edilmelidir. Bu dosyanın içeriği şöyledir:

— API fonksiyonlarının prototipleri
— Çeşitli yapıların bildirimleri
— Çeşitli typedef isimleri
— Sembolik sabitler
— Makrolar
— Diğer bildirimler...

windows.h dosyasının kendisi küçük olmasına karşın pek çok başka başlık dosyalarını include ettiği için büyük bir include işlemine yol açmaktadır. Kursumuzda bir bildirimin windows.h dosyası içerisinde olduğu belirtildiğinde bu bildirimin windows.h dosyasının include edilmesiyle bir biçimde kaynak koda dahil edildiği anlaşılacaktır.

Modern derleyicilerde çok büyük başlık dosyalarının önişlemci tarafından hızlı bir biçimde ele alınabilmesi için "precompiled header" denilen bir yöntem kullanılmaktadır. Bu yönteme göre derleyici, ilk kez başlık dosyasını ele aldığında başlık dosyası içindeki bildirimleri indeksli (binary) olarak bir dosyanın içerisine yazar. Sonraki derleme işlemlerinde yeniden aynı text dosyayı işleme sokmaz; indeksli dosyayı kullanır. Bu dosya, debug dizini içindeki pch dosyasıdır.

windows.h Dosyasında Kullanılan Typedef İsimleri

windows.h içerisinde API fonksiyonlarında kullanılan pek çok typedef ismi vardır. Bu typedef isimleri, kolay yazım sağlamasının yanında taşınabilirliği de arttırmaktadır.

— BYTE, 1 byte'lık işaretsiz tamsayı türünü; WORD, 2 byte'lık işaretsiz tamsayı türünü; DWORD, 4 byte'lık işaretsiz tam sayı türünü ifade eder. Win32 sistemlerinde tipik olarak aşağıdaki gibi typedef edilmişlerdir:

```
typedef unsigned char   BYTE;
```

```
typedef  unsigned  short  int  WORD;
typedef  unsigned  long   int  DWORD;
```

— BOOL, int olarak typedef edilmiştir.

```
typedef  int   BOOL;
```

— Genel olarak adres türleri için P öneki; gösterdiği yer const olan adres türleri için PC öneki getirilir.
— PSTR, char* olarak; PCSTR, const char * olarak typedef edilmiştir.

```
typedef  char   *PSTR;
typedef  const char   *PCSTR;
```

— Win16 sistemlerinde uzak gösterici kavramı vardı. Uzak göstericilere ilişkin tür isimleri, LP ve LPC öneki ile başlatılıyorlardı. Fakat Win32 sistemlerinde uzak gösterici kavramı yoktur. Ancak hala LP ve LPC önekleri kullanılmaya devam etmektedir. LP öneki ile P önekinin; LPC önekiyle PC önekinin bir farkı yoktur.
— Bütün doğal türler tamamen büyük harflerle typedef edilmişlerdir. Örneğin:

```
typedef  int     INT;
typedef  long    LONG;
...
```

— Bütün doğal türlerin adres karşılığı P önekli biçiminde; const adres karşılığı PC öneki biçiminde bulunmaktadır. Örneğin:

```
typedef  int     *PINT;
typedef  long    *PLONG;
...
```

— void * için PVOID yada LPVOID; const void * için LPCVOID kullanılmaktadır (her nedense PCVOID yoktur).
— Genel olarak yapı isimleri “tag” sözcüğü ile başlatılmıştır fakat typedef isimlerinden bu tag öneki kaldırılmıştır. Yapı isimlerinin de adres biçimleri ve const adres biçimleri PV, PC önekleri yada LP ve LPC önekleri ile belirtilmiştir. Örneğin:

```
typedef  struct  tagRECT {
      ...
} RECT, *PRECT;

typedef   const  RECT  *PRECT;
```

— windows.h içerisindeki sembolik sabitler önce xx_ biçiminde bir önekle başlatılmıştır. Bu xx, sembolik sabitin hangi konuya ilişkin olduğunu belirtmektedir. Örneğin:

```
CS_REDRAW
WS_CHILD
```

— Bütün H ile başlayan tür isimleri void * olarak typedef edilmiştir. Örneğin:

```
HCURSOR   hcursor;
HWIND     hwind
```

```
    typedef   void *HWIND;
    typedef   void *HICON;
...
```

Anahtar Notlar:

Genel olarak bir API fonksiyonunun geri dönüş değerinde BOOL yazıyorsa geri dönüş değeri, fonksiyonun başarısını belirtir. Fonksiyon başarılıysa 0 dışı bir değere, başarısızsa 0 değerine geri döner. windows.h içerisinde FALSE sembolik sabiti 0 olarak; TRUE sembolik sabiti 1 olarak define edilmiştir. Func bir API fonksiyonu olmak üzere, aşağıdaki işlem hatalıdır:

```
if ( Func() == TRUE ){
        ...
};
```

Burada fonksiyon başarılı ise 1 değerine geri dönmek zorunda değildir. Fakat aşağıda yapılanlar doğrudur:

```
if ( Func()){
        ...
};
if ( Func() == FALSE) {
        ...
};
if (!Func() ) {
        ...
};
```

İSKELET WINDOWS GUI PROGRAMI

Ekrana bir pencere çıkartan bir GUI programı bile onlarca satır sürmektedir. Böyle temel programa "iskelet GUI programı" diyeceğiz. Bir Windows programı yazarken hemen her programda gereken bu temel işlemleri yeniden yazmak yerine iskelet programı kullanmak daha uygundur. İskelet program, generic.c ismi ile dağıtılmıştır.

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
```

```
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_QUESTION);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}
	hWnd = CreateWindow("Generic", "Sample Windows",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);
	if (!hWnd)
		return -1;
	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
	switch (message) {
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

İskelet GUI programı, 5 bölümden oluşmaktadır.

1. WinMain fonksiyonu
2. Programın ana penceresine ilişkin sınıfın register ettirilmesi
3. Programın ana penceresinin yaratılması
4. Mesaj döngüsü
5. Pencere fonksiyonu

1. WinMain Fonksiyonu

Bir Windows GUI programının başlangıç fonksiyonu WinMain fonksiyonudur. WinMain fonksiyonunun parametrik yapısı aşağıdaki gibi olmak zorundadır:

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
```

WinMain fonksiyonunun geri dönüş değeri int olmak zorundadır. Fonksiyonun geri dönüş değeri tamamen main fonksiyonunda olduğu gibi prosesin sonlanma kodudur. Geleneksel olarak başarı durumunda 0; başarısızlık durumunda 0 dışı bir değere geri dönülür.

WINAPI, __stdcall anlamına gelen bir makrodur.

```
#define   WINAPI   __stdcall
```

WinMain fonksiyonunu çağırma biçimi __stdcall olmalıdır.

Anahtar Notlar:

Fonksiyonun geri dönüş değerinin türü ile fonksiyon ismi arasına yazılan anahtar sözcüklere "fonksiyonun çağırma biçimi (calling convention)" denir. Fonksiyonun çağırma biçimi kavramı, standartlarda olan bir kavram değildir. Daha çok, kullanılan mikroişlemci sistemine özgü bir kavramdır. Fonksiyon çağırma biçimi __cdecl, __pascal, __stdcall biçiminde olabilir. Fonksiyonun çağırma biçimi, parametrelerin stack'e atılma sırasını, stack'in temizlenme biçimini ve fonksiyon isimlerinin dekore edilme biçimini belirleyen bir kavramdır. C'de eğer çağırma biçimi yazılmazsa __cdecl (C decleration) yazılmış gibi işlem yapılır. Win16 sistemleri __pascal çağırma biçimini kullanıyordu. Oysa Win32 sistemleri bu iki çağırma biçiminin karışımı olan __stdcall çağırma biçimini kullanmaktadır.

Fonksiyonun birinci parametresi olan **hInstance**, PE formatının belleğe yüklenme adresini belirtir. Yükleme adresi, NT grubu sistemlerde minimum 64K; 95 grubu sistemlerde ise 4MB'dir. Aslında yükleme adresi, PE formatının içerisinde yazmaktadır ve linker seçenekleri ile değiştirilebilir. Yükleyici, yükleme adresini buradan alarak WinMain fonksiyonuna geçirir. WinMain fonksiyonuna geçirilen bu hInstance değeri, bazı API fonksiyonları için gerekmektedir. Farklı programların hInstance değerleri farklı olabilmektedir.

Fonksiyonun ikinci parametresi olan **hPrevInstance**, Win16 sistemlerinde anlamlıydı. Win32 sistemlerinde bir anlamı kalmamıştır. Fakat geriye doğru uyumu korumak için bu değer, hala muhafaza edilmektedir. Win32 sistemlerinde bu parametre her zaman NULL olarak geçilmektedir.

Anahtar Notlar:

Win16 sistemlerinde hInstance parametresi programın belleğe yüklenme adresini belirtmiyordu. Bu değer, programın modül database adresini belirtiyordu ve sistem genelinde tekti. O sistemlerde (Win16) hPrevInstance ise program ilk kez çalışıyorsa NULL; aksi durumda daha önceki çalışan kopyanın hInstance değeriydi. Dolayısıyla bu sistemlerde bir programdan birden fazla kopya çalıştırması istenmiyorsa aşağıdaki gibi işlem yapılıyordu:

```
if (hPrevInstance != NULL)
```

exit(0);

Oysa artık Win32 sistemlerinde bu durum anlamsız hale gelmiştir.

WinMain fonksiyonunun 3. parametresi olan **lpszCmdParam**, char * türündendir. Programın komut satırı argümanlarını belirtir. Fakat Windows'ta komut satırı argümanları main fonksiyonunda olduğu gibi bir gösterici dizisi biçiminde WinMain fonksiyonuna geçirilmez. Program isminden sonraki tüm satır, argümanlar arasındaki boşluklar muhafaza edilerek tek bir yazıymış gibi geçirilir. Bunun parse edilmesi, programcının sorumluluğundadır.

WinMain fonksiyonunun son (4.) parametresi olan **nCmdShow**, programın ana penceresinin nasıl görüntüleneceğine ilişkin bir tavsiye değeridir. Bu değer, programcı tarafından kullanılabilir yada kullanılmayabilir.

2. Programın Ana Penceresine İlişkin Sınıfın Register Ettirilmesi

Programının ana penceresi CreateWindow yada CreateWindowEx API fonksiyonları ile yaratılır. Fakat bir pencereyi yaratmadan önce o pencerenin bazı genel özelliklerinin, sisteme tanıtılması gerekmektedir. Bu sisteme tanıtma işlemine pencere sınıfının register ettirilmesi denilmektedir.

Buradaki pencere sınıfı kavramının, nesne yönelimli programlamadaki sınıf kavramı ile bir ilgisi yoktur. Pencere sınıfı, pencerenin çeşitli özelliklerini belirten bilgi topluluğudur.

Pencere sınıfının register ettirilmesi için, önce WNDCLASS isimli bir yapı değişkeni tanımlanır; sonra bu değişkenin içi doldurularak RegisterClass yada RegisterClassEx fonksiyonları çağırılır. Bu fonksiyonlar, WNDCLASS içerisindeki bilgileri, sistem alanına kopyalarlar.

Anahtar Notlar:

Windows işletim sisteminin temelleri Win16 sistemleri zamanında atılmıştır. Gerçekten de API fonksiyonlarının büyük bölümü, o sistemlerden aktarılmıştır. Fakat zaman geçtikçe bu fonksiyonların bazılarında yetersizlikler görülmüş ve düzeltilme yoluna gidilmiştir. Microsoft, eskiden yazılan kodların bu işlemden etkilenmemesi için hem eski fonksiyonları korumuş, hem de bunların genişletilmiş ve iyileştirilmiş yeni biçimlerini tanımlamıştır. Bu yeni biçimler, eskilerinin sonuna "Ex" eki getirilerek isimlendirilmiştir.

RegisterClass fonksiyonunun parametrik yapısı şöyledir:

```
ATOM RegisterClass(CONST WNDCLASS *lpWndClass);
```

Fonksiyon, parametre olarak WNDCLASS türünden nesnenin adresini alır. Fonksiyon başarılı ise register ettirilen sınıfa ilişkin ID değerine; başarısızsa 0 değerine geri döner. ATOM, WORD türünü belirten bir typedef ismidir.

WNDCLASS yapısının elemanları ve anlamları şöyledir:

UINT style: Bu eleman, pencere sınıfının bazı temel özelliklerini belirlemek için kullanılır. İskelet programda bu elemana şöyle değer atanmıştır:

```
wndClass.style = CS_HREDRAW | CS_VREDRAW ;
```

Böylece pencere, yatayda yada dikeyde boyut değiştirdiğinde pencereye WM_PAINT mesajı gönderilecektir.

Anahtar Notlar:

Bazı API fonksiyonlarının parametreleri, bir grup özellikten hangilerinin seçildiğini belirlemekte kullanılır. Bunun için, tüm bitleri 0 olan fakat yalnızca bir biti 1 olan çeşitli sembolik sabitler tanımlanmıştır. Programcı, bu sembolik sabitlere "bitsel or (|)" işlemi uygular ve böylece bazı bitleri 1 olan bir sayı elde eder. Fonksiyon, bu sayının 1 olan bitlerini araştıracak ve o özelliklerin belirtildiğini anlayacaktır.

HICON hIcon: Bu parametre, pencere başlığında görüntülenecek simgeyi (icon) belirlemekte kullanılır. İskelet programda bu elemana şöyle değer atanmıştır:

```
wndClass.hIcon = LoadIcon(NULL, IDI_QUESTION);
```

Bu ifade ile pencerede simge olarak "?" biçiminde bir simge görülür.

HBRUSH hbrBackground: Bu eleman, pencerenin zemin rengini belirlemekte kullanılır. İskelet programda bu elemana şöyle değer atanmıştır:

```
wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
```

Bu ifade, pencerenin zemin renginin beyaz olmasını sağlamaktadır.

HCURSOR hCursor: Windows'ta cursor, fare oku için kullanılan bir değimdir. Bu eleman farenin pencere sınırları içerisine sokulduğunda hangi şekilde gözükeceğini belirlemekte kullanılır. İskelet programda bu elemana şöyle değer verilmiştir:

```
wndClass.hCursor = LoadCursor(NULL, ID_ARROW);
```

Bu ifade ile fare oku, pencere sınırlarına sokulduğunda → biçiminde görüntülenir.

LPCTSTR lpszMenuName: Bu eleman, programın ana menüsüne ilişkin menü kaynağının ismini almaktadır. İskelet programda bu elemana şöyle değer verilmiştir:

```
wndClass.lpszMenuName = NULL;
```

Bu durum, programın, menüye sahip olmadığı anlamına gelir.

int cbClsExtra: Bu eleman, register ettirilen sınıf için ayrılacak ekstra byte sayısını belirtmektedir. İskelet programda bu elemana 0 değeri verilmiştir:

```
wndClass.cbClsExtra = 0;
```

int cbWndExtra: Bu eleman, pencerenin handle alanında ayrılacak ekstra byte sayısını belirtmektedir. İskelet programda bu elemana 0 değer, verilmiştir:

```
wndClass.cbWndExtra = 0;
```

LPCTSTR lpszClassName: Bu eleman, pencere sınıfının ismini belirlemekte kullanılır. Windows'ta pencere sınıfları bir isim ve ATOM numarasıyla belirlenmektedir. Genellikle isim, daha yaygın kullanılmaktadır. İskelet programda bu elemana "Generic" yazısının adresi atanmıştır. Bu sınıf ismi, CreateWindow fonksiyonunda kullanılacaktır.

```
wndClass.lpszClassName = "Generic";
```

WNDPROC lpfnWndProc: WNDPROC, bir fonksiyon göstericisine ilişkin tür ismidir. Bu eleman, pencerenin pencere fonksiyonunun başlangıç adresini tutar. İskelet programda bu elemana şöyle değer verilmiştir:

```
wndClass.lpfnWndProc = (WNDPROC) WndProc;
```

HINSTANCE hInstance: Bu eleman, programın sanal belleğe yüklenme adresini içerir. İskelet programda bu elemana WinMain fonksiyonuna geçirilen hInstance değeri verilmiştir.

```
wndClass.hInstance = hInstance;
```

Sınıfın register ettirilmesi işlemi, Win16 sistemlerinde programın ilk kopyasının çalıştırıldığında yapılması gereken, sonraki kopyaların çalıştırılmasında yapılmaması gereken bir işlemdi. Bu nedenle register işlemi şöyle yapılmaktaydı:

```
if (!hPrevInstance) {
        ...
        ...
}
```

Halbuki Win32 sistemlerinde register ettirme işlemi, çalıştırılan her kopya için ayrıca yapılmak zorundadır. Bu sistemlerde hPrevInstance her zaman NULL olarak geçirildiğine göre yukarıdaki if deyimi de bu sistemlerde her zaman doğrudan sapacaktır. Aslında bugün bu if deyimine hiç gerek yoktur. Fakat geleneksel olarak programcıların çoğu hala bu yapıyı kullanırlar. İskelet programda gerekmediği halde bu if deyimi kullanılmıştır.

3. Programın Ana Penceresinin Yaratılması

Windows'ta tüm pencereler, CreateWindow yada CreateWindowEx fonksiyonlarıyla yaratılmaktadır. İskelet programda da programın ana penceresi CreateWindow fonksiyonuyla yaratılmıştır.

```
hWND CreateWindow(
        LPCTSTR lpClassName,     // pointer to registered class name
        LPCTSTR lpWindowName,// pointer to window name
        DWORD dwStyle,           // window style
        int x,                   // horizontal position of window
        int y,                   // vertical position of window
        int nWidth,              // window width
```

```
    int nHeight,            // window height
    HWND hWndParent,        // handle to parent or owner window
    HMENU hMenu,            // handle to menu or child-window
    HANDLE hInstance,       // handle to application instance
    LPVOID lpParam,         // pointer to window-creation data
);
```

Pencereler, pek çok özelliğe sahip nesnelerdir. Bu özelliklerin bir bölümü, sınıfın register ettirilmesi sırasında belirlenmekte, bir bölümü ise CreateWindow fonksiyonunda belirlenmektedir. Fonksiyonun parametreleri ve anlamları şöyledir:

LPCTSTR lpClassName: Fonksiyonun 1. parametresi olan bu parametreye pencerenin yaratılacağı register ettirilmiş olan pencere sınıfının ismi girilmiştir. İskelet programda bu parametre "Generic" olarak girilmiştir.

LPCTSTR lpWindowName: Fonksiyonun 2. parametresine, pencere başlığında çıkacak yazının adresi girilir.

DWORD dwStyle: Bu parametrede, pencerenin çeşitli biçimsel özellikleri belirtilmektedir. Bu parametre, WS_XXX biçiminde çeşitli sembolik sabitlerin bit or işlemine sokulmasıyla elde edilmektedir. Her sembolik sabit, bir özelliğin olup olmadığını belirtir. bu parametreye genel olarak "Pencere biçimi" denilmektedir.

Önemli pencere biçimleri şunlardır:

WS_BORDER: Pencerenin sınır çizgilerinin bulunmasını sağlar.

WS_CAPTION: Pencere başlığının bulunmasını sağlar.

WS_SYSMENU: Pencerenin sol üst köşesine tıklandığındaki sistem menüsünün çıkmasını sağlar.

WS_MINIMIZEBOX, WS_MAXIMIZEBOX: Bu pencere biçimleri, pencerenin sağ üst köşesindeki minimize, maximize/restore tuşlarının çıkmasını sağlar. Maximize tuşu ile restore tuşları, aynı tuşlardır. Pencerenin sistem menüsü varsa her zaman X tuşu vardır. Ayrıca, bu tuşların çıkması için WS_SYSMENU pencere biçiminin eklenmiş olması gerekmektedir.

WS_THICKFRAME: Bu pencere biçimi, pencere sınır çizgilerinin kalın bir çizgi olmasını sağlar. Fakat asıl önemlisi, bu pencere biçiminin pencerenin boyutunun değiştirilmesi mümkün hale getirmesidir.

WS_HSCROLL, WS_VSCROLL: Bu pencere biçimleri, pencerenin yatay ve düşey kaydırma çubuklarına sahip olmasını sağlar.

WS_OVERLAPPED, WS_POPUP: Bir ana pencere ya overlapped olur yada popup olur. Normal pencereler overlapped'tir. POPUP pencereler konusu ileride ele alınacaktır. Normal bir pencere yaratıyorsak WS_OVERLAPPED pencere biçimini eklemeliyiz.

WS_CHILD: Bu pencere biçimi, alt pencere yaratmak için kullanılmaktadır.

İskelet programda pencere biçimi parametresi, WS_OVERLAPPEDWINDOW biçiminde girilmiştir. WS_OVERLAPPEDWINDOW, aslında kendi başına bir pencere biçimi değil; aşağıdaki gibi bir makrodur:

```
#define   WS_OVERLAPPED ( WS_OVERLAPPED |
                          WS_CAPTION |
                          WS_SYSMENU |
                          WS_THICKFRAME |
                          WS_MAXIMIZEBOX |
                          WS_MINIMIZEBOX)
```

---

int x, int y, int nwidth, int nheight: Bu parametreler, pencerenin ilk ara boyutunun konumunu belirtir. x ve y, sol üst köşe koordinatları; nwidth ve nheight, yatay ve düşey uzunluklarıdır. Ana pencereler için bu orijin, sol üst köşesidir. Eğer x parametresi CW_USEDEFAULT geçilirse y parametresi dikkate alınmaz. Bu durumda x ve y değerleri, sistem tarafından otomatik alınır. Benzer biçimde eğer nwidth parametresi CW_USEDEFAULT geçilirse, nheight parametresi dikkate alınmaz. Bu durumda pencerenin genişlik ve yüksekliği sistem tarafından otomatik olarak alınır. İskelet programda bu parametreler:

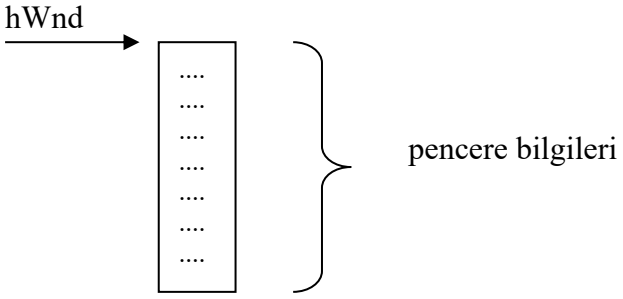
CW_USEDEFAULT, 0,
CW_USEDEFAULT, 0, biçiminde alınmıştır.

HWND hWndParent: Bu parametre, eğer yaratılacak pencere ana pencere ise NULL geçilmelidir. Alt pencere ise üst pencerenin handle değeri olarak girilmelidir. İskelet programda ana pencere yaratıldığı için bu parametre NULL değeri biçiminde girilmiştir.

HMENU hMenu: Bu parametre, iki anlamlıdır. Ana pencere için , ana pencere menüsünün handle değerini, alt pencere için, alt pencere ID değerini belirtir. İskelet programda bu parametre NULL olarak geçilmiştir; bu da ana pencerenin menüye sahip olmayacağı anlamına gelir.
HANDLE hInstance: Bu parametre, programın belleğe yüklenme adresini alır. WinMain'e geçirilen hInstance değeri, bu parametreye aktarılmalıdır.

LPVOID lpParam: Bu parametre, WM_CREATE mesajına geçirilecek değeri belirlemekte kullanılır. İskelet programda bu parametre NULL olarak geçirilmiştir.

CreateWindow fonksiyonun geri dönüş değeri başarı durumunda, yaratılan pencereye ilişkin handle değeridir. Yani fonksiyon, yaratılan pencereye ilişkin bütün bilgileri bir bölgede saklar; o bölgenin başlangıç adresi ile geri döner. Fonksiyon başarısızsa NULL değerine geri döner.



CreateWindow fonksiyonundan elde edilen bu handle değeri, pencereler üzerinde işlem yapan fonksiyonlara parametre olarak geçirilir. Pencereler üzerinde işlem yapan fonksiyonlar, birinci parametre olarak bu handle değerini alırlar.

ShowWindow ve UpdateWindow Fonksiyonları:

İskelet programda, programın ana penceresi yaratıldıktan sonra, ShowWindow ve UpdateWindow fonksiyonları çağırılmıştır. Bir pencere, CreateWindow fonksiyonu ile yaratıldığında hemen görüntülenebilir

yada görüntülenmeyebilir. Bu durum, WS_VISIBLE pencere biçimine bağlıdır. Eğer WS_VISIBLE pencere biçimi belirtildiyse pencere yaratılır yaratılmaz görüntülenir. Eğer bu pencere biçimi belirtilmemişse, pencere yaratılır yaratılmaz görüntülenmez; istenilen bir zaman ShowWindow fonksiyonu kullanılarak görüntülenir. ShowWindow fonksiyonunun prototipi şöyledir:

```
BOOL ShowWindow( HWND   hWnd, int   nCmdShow );
```

Fonksiyonun 1. parametresi, görüntülenecek pencerenin handle değeridir. Bu fonksiyon aslında yalnızca pencereyi görünür yapmak için değil, görünmez yapmak için de kullanılır.

Fonksiyonun 2. parametresi, ne yapılacağını belirtmektedir. Bu parametre, SW_XXX biçimindeki sembolik sabitlerden oluşturulur. Önemli değerler şunlardır:

SW_MAXIMIZE, SW_MINIMIZE, SW_RESTORE: Bu değerler pencerenin hangi boyutta görüntüleneceğini belirtir.

SW_HIDE: Bu, pencereyi görünmez yapmakta kullanılır.

Anahtar Notlar:

ShowWindow fonksiyonu, istenildiği zaman kullanılabilir. Örneğin penceremizi minimize yapmak istiyorsak bunun en basit yolu ShowWindow fonksiyonunu çağırmaktır.

Anahtar Notlar:

Sıklıkla görev çubuğu (task bar) ve görev yöneticisi (task manager) kavramları birbiriyle karıştırılmaktadır. Görev çubuğu, o anda açılmış bulunan ana pencereleri görüntülemekte kullanılır. Ctrl+Alt+Del tuşlarıyla açılan görev yöneticisi ise, çalışmakta olan programları yani prosesleri görüntülemekte kullanılır. Bir program, birden fazla ana pencere yaratabilir. Bu durumda görev çubuğunda birden fazla pencere görüntülenir. Ayrıca bir program hiç ana pencere de yaratmayabilir. Bu durumda görev çubuğunda bir şey görülmez fakat görev yöneticisinde proses görüntülenir. O halde, pencere bakımından 3 çeşit program olabilir.

1. ***Console Programları:*** *Bu tür programlarda program yüklenir yüklenmez sistem, siyah bir pencere açar. Programın girişi main fonksiyonudur.*
2. ***GUI Programları:*** *Bu tür programlarda programın en az 1 ana penceresi vardır. Programın giriş noktası WinMain fonksiyonudur.*
3. ***Penceresiz Programlar:*** *Burada programın giriş noktası WinMain fonksiyonudur. Ama programcı CreateWindow fonksiyonu ile hiçbir pencere yaratmamıştır.*

UpdateWindow Fonksiyonu

Bu fonksiyon, iskelet programda ShowWindow fonksiyonundan sonra kullanılmıştır. Pencereye doğrudan WM_PAINT mesajını gönderir.

4. Mesaj Döngüsü

Programın ana penceresi yaratıldıktan sonra artık gelen mesajların işlenmesi için bir mesaj döngüsünün oluşturulması gerekir. Windows, pencereye ilişkin bir olay gerçekleştiğinde bu olayı MSG isimli bir yapı ile ifade eder ve MSG yapılarından oluşan bir kuyruk sistemine yerleştirir. Programcı da kuyruktan mesajları alarak işleyecek bir döngü oluşturmalıdır. Bu döngüye "mesaj döngüsü" (message loop) denilmektedir. Mesaj döngüsü içerisinde programcı, sıradaki mesajı alarak mesajın ne amaçla gönderildiğini inceler ve uygun işlemleri yapar.

Windows programının sonlanması için programın akışının mesaj döngüsünden çıkması gerekir. Akış, mesaj döngüsünden çıkınca WinMain fonksiyonu biter ve program da sonlanır.
MSG yapısı şöyledir:

```
typedef struct tagMSG (
        HWND        hwnd;
        UINT        message;
        WPARAM      wParam;
        LPARAM      lParam;
        DWORD       time;
        POINT       pt;
) MSG;
```

Yapının **hwnd** elemanı, olaya konu olan pencerenin handle değerini içerir. Mesaj kuyruğunda birden fazla pencereye ilişkin mesaj bir arada bulunabilir. Mesajın hangi penceredeki olay nedeniyle gönderildiği, bu eleman ile anlaşılabilir.

Yapının **message** elemanı, gerçekleşen olayı anlatan bir sayı belirtir. Yani Windows'ta mesaja konu olabilecek her olay, bir sayı ile ifade edilmiştir. Fakat bütün bu sayılar windows.h içerisinde WM_XXX biçimindeki sembolik sabitlerle ifade edilmiştir. Böylece programcı, konuşurken yada kodlama yaparken mesaj numaraları ile değil, bu sembolik sabit isimleri ile mesajları belirtir.

Yapının **wParam** ve **lParam** elemanları sırasıyla WPARAM ve LPARAM türlerindendir. WPARAM , Win16 sistemlerinde 16 bit bir tamsayı türü idi. Fakat Win32 sistemlerinde 32 bite yükseltilmiştir. Bugün Win32 sistemlerinde WPARAM ve LPARAM türlerinin her ikisi de DWORD olarak typedef edilmiştir. wParam ve lParam elemanları tek başına anlamlı elemanlar değildir. Mesaj numarasına göre o mesaja özgü anlamları vardır. Gerçekleşen olaya ilişkin diğer bilgiler, bu elemanlara yazılırlar. Programcı, hangi mesajlarda bu elemanlara hangi bilgilerin yerleştirilmiş olduğunu bilmelidir.

Yapının **time** elemanı, mesajın kuyruğa yazıldığı zamanı belirtmektedir. Bu zaman, Windows boot edildikten sonra geçen göreli bir zamandır. Nihayet yapının son elemanı olan **pt**, yalnızca bazı mesajlarda kullanılan koordinat bilgisini tutar.

Mesaj kuyruğu, Win16 sistemlerinde programa özgüydü. Yani her programın mesaj döngüsü birbirinden farklıydı. Win32 sistemlerinde mesaj kuyrukları thread başına oluşturulmaktadır. Yani her thread'in ayrı bir mesaj kuyruğu vardır. Bir thread'in yarattığı tüm pencerelere ilişkin mesajlar, o thread'in mesaj kuyruğuna bırakılır. Yani Windows, bir olay gerçekleştiğinde olaya konu olan pencerenin hangi thread tarafından yaratıldığını belirlemekte ve mesajı, o thread'in kuyruğuna bırakmaktadır. Bazı istisnaları olsa da genel olarak mesajlar kuyruğun sonuna eklenir, başından alınır. Mesaj alındığında kuyruktan silinmektedir.

Win16 sistemlerinde mesaj kuyruğu, bir dizi biçiminde oluşturulmuştu ve en fazla 8 mesajın biriktirilmesine izin veriliyordu. Oysa Win32 sistemlerinde mesaj kuyruğu bağlı liste tekniğiyle oluşturulmuştur ve kuyruk uzunluğu konusunda bir sınırlama yoktur.

Anahtar Notlar:

Çok işlemli ve çok thread'li sistemlerde dışsal olayları bekleyen thread'ler, etkin bir bekleme sağlamak için işletim sistemi tarafından olay gerçekleşene kadar geçici süre çizelge dışına çıkarılırlar. Bu işleme "thread'in bloke edilmesi" denilmektedir. İlgili olay gerçekleştiğinde, thread yeniden çizelgeye alınarak işleme sokulur. Bloke işleminde amaç, olay gerçekleşene kadar thread'in, gereksiz bir biçimde CPU zamanı harcamamasını sağlamaktır.

GetMessage Fonksiyonu

GetMessage, mesaj kuyruğundan sıradaki mesajı alan bir API fonksiyonudur. Fonksiyon, kuyruktan mesajı alarak MSG yapısı biçiminde programcıya iletir. Eğer kuyrukta mesaj yoksa, thread'i bloke ederek bekler. Prototipi aşağıdaki gibidir:

```
BOOL GetMessage(
 LPMSG     lpmsg,          // address of structure with message
 HWND      hWnd,           // handle of window
 UINT      wMsgFilterMin,  // first message
 UINT      wMsgFilterMax   // last message
);
```

Fonksiyonun birinci parametresi, MSG türünden bir yapı nesnesinin adresini alır. Fonksiyon, kuyruktan aldığı mesaj bilgilerini, bu yapıya yerleştirecektir.

Fonksiyonun ikinci parametresi, hangi pencereye ilişkin mesajların alınacağını belirtir. GetMessage aslında, kuyruktaki belirli bir pencereye ilişkin mesajları alabilmektedir. Bu parametre NULL geçilirse, tüm pencerelere ilişkin mesajların alınacağı anlamına gelir.

Fonksiyonun 3. ve 4. parametreleri belirli numaralar arasındaki mesajların kuyruktan alınmasını sağlar. Örneğin biz bu parametreleri 100, 150 biçiminde belirlersek GetMessage kuyrukta ilerler; bu aralıktaki ilk mesajı alır. Veya bu parametreye 0, 0 biçiminde girilirse kuyruktan her türlü mesajın alınacağı anlamına gelir.

GetMessage fonksiyonu, eğer kuyruktan WM_QUIT isimli özel mesajı aldıysa 0 ile geri döner. GetMessage, diğer tüm mesajlar için 0 dışı bir değerle geri dönmektedir. Mesaj döngüsünün yapısı şöyledir:

```
while (GetMessage(...)){
        ...
}
```

Görüldüğü gibi mesaj döngüsünden çıkmak için GetMessage fonksiyonunun 0 ile geri dönmesi gerekmektedir. GetMessage fonksiyonunun 0 ile dönmesi için ise WM_QUIT mesajının kuyruktan alınması gerekir. O halde bir Windows programının sonlanması için WM_QUIT mesajı gerekmektedir.

5. Pencere Fonksiyonu

Kuyruktan alınan mesajlar, pencere fonksiyonu denilen bir fonksiyon tarafından işlenir. Pencere fonksiyonunun parametrik yapısı aşağıdaki gibi olmak zorundadır:

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT message,
                          WPARAM wParam, LPARAM lParam)
{
        ...
}
```

LRESULT, long olarak typedef edilen bir tür ismidir. CALLBACK, __stdcall anlamına gelen bir makrodur.

Pencere fonksiyonu, programcı tarafından değil, DispatchMessage fonksiyonu tarafından çağırılır. Programcı GetMessage ile mesajı alıp DispatchMessage fonksiyonuna verir. DispatchMessage ise mesajı ayrıştırarak pencere fonksiyonunu çağırır.

```
while (GetMessage(&msg, NULL, 0, 0)){
        TranslateMessage(&msg);
        DispatchMessage(&msg);
}
```

Fonksiyonun prototipi aşağıdaki gibidir:

```
LONG DispatchMessage( CONST MSG *lpmsg);
```

Mesaj döngüsü içindeki TranslateMessage fonksiyonunun, mesajın işlenmesi ile ilgili önemli bir etkisi yoktur. Hatta bu fonksiyon döngüden çıkarılabilir.

```
BOOL TranslateMessage( CONST MSG *lpMsg);
```

Neden pencere fonksiyonunu biz doğrudan çağırmıyoruz da DispatchMessage fonksiyonu ile çağrılmasını sağlıyoruz? DispatchMessage fonksiyonu bazı mesajlar için bazı önemli işlemler yapmaktadır. Eğer fonksiyonu biz çağıracak olsaydık bu işlemleri biz yapmak zorunda kalırdık.

Her pencerenin bir pencere fonksiyonu vardır. Pencere fonksiyonu sınıfın register ettirilmesi belirlenmesi işlemi sırasında belirlenir. WNDCLASS yapısının lpfnWndProc elemanına, pencere fonksiyonunun ismi yani adresi yerleştirilmelidir. DispatchMessage, sistemdeki kayıtlı bilgilerden bir handle değerine ilişkin pencerenin pencere fonksiyonunu bulabilmektedir.

Pencere fonksiyonunda mesajın numarasına bakılmalı ve uygun işlemler yapılmalıdır. Bu işlem tipik olarak bir switch deyimiyle yapılabilir. Programcı en azından programın doğru sonlandırılmasını sağlamak amacıyla WM_DESTROY mesajını işlemelidir.

```
switch(message){
        case WM_DESTROY: PostQuitMessage(0); break;
        ....
}
```

WM_DESTROY mesajı ve PostQuitMessage fonksiyonu ileride ele alınacaktır.

Windows'ta yüzlerce mesaj vardır. Üstelik bazı mesajlarda programcının bazı kritik işlemleri yapması gerekir. İşte böylesi kritik işlemler, DefWindowProc fonksiyonuna bırakılabilir. Bu durumda programcı, işlemediği mesajları DefWindowProc fonksiyonuna vermeli ve bu fonksiyona işletmelidir. Şüphesiz her mesajın DefWindowProc tarafından işletilmesi zorunlu değildir. DefWindowProc, bazı mesajlara ilişkin

kritik işlemleri yapar bazıları için ise hiç işlem yapmaz. Fakat programcı prensip olarak işlemediği mesajlar için bu fonksiyonu çağırmalıdır. En uygun durum, switch ifadesinin default kısmına yerleştirmektir. DefWindowProc fonksiyonunun parametrik yapısı, pencere fonksiyonu ile aynıdır.

```
LRESULT DefWindowProc(
  HWND hWnd,            // handle to window
  UINT Msg,             // message identifier
  WPARAM wParam,        // first message parameter
  LPARAM lParam         // second message parameter
);
```

Pencere fonksiyonundan hangi değerle geri dönülmelidir? Aksi belirtilmediği sürece işlenen mesajlar için 0 değeri ile, işlenmeyen mesajlar için DefWindowProc fonksiyonunun geri dönüş değeri ile geri dönülmelidir. O halde pencere fonksiyonunun yapısı aşağıdaki gibi olacaktır:

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam)
{
        switch (message) {
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

Windows GUI Programlarının Microsoft Visual C++ Derleyicilerinde Kullanılması

API programı Visual C 6.0 derleyici siteminde sırasıyla şu aşamalarla derlenip çalıştırılır.

1. File/New/Project menüsü açılır ve Win32 Application projesi seçilir. Project name olarak bir isim girilir. Geliştirme sistemi (IDE), girilen isime ilişkin location kısmında belirtilen dizin altında bir dizin açar.
2. Çıkan menüde "An empty project" seçilir.
3. Proje yaratıldıktan sonra ismine project workspace denilen bir pencere oluşur. (Bu pencere kapanırsa Alt+0 ile açılabilir.). Project workspace içerisinde ClassView ve FileView isminde iki kısım vardır. FileView'da proje işçindeki kaynak dosyalar görüntülenmektedir.
4. Artık Kaynak dosyayı projeye ekleme zamanı gelmiştir. Bu, iki biçimde yapılabilir.
   a) Project/Add to project/Files menüsü kullanılarak
   b) Bold yapılmış kısım üzerinde sağ click yapılarak Add files to project yolu ile yapılabilir.
5. Artık editördeki dosyanın derlenmesi için Build/Compile; .exe oluşturmak için Build/build seçilir. En sonunda program, Build/execute ile çalıştırılır. Aslında Build/Execute (Ctrl+F5) yapıldığında bütün bu işlemler otomatik olarak yapılıp programlar çalıştırılmaktadır.

FARE MESAJLARI

Fare ile yapılan temel işlemlerde, çeşitli mesajlar, Windows tarafından kuyruğa bırakılmaktadır. Önemli fare mesajları şunlardır:

WM_LBUTTONDOWN
WM_LBUTTONUP

WM_RBUTTONDOWN
WM_RBUTTONUP
WM_MOUSEMOVE
WM_NCLBUTTONDOWN
WM_NCLBUTTONUP
WM_NCRBUTTONDOWN
WM_NCRBUTTONUP
WM_NCMOUSEMOVE

Bütün fare mesajlarının parametreleri aşağı yukarı aynıdır.

LOWORD(lParam) → x
HIWORD(lParam) → y
wParam → Aynı anda basılan tuşlar

DOWN son ekli mesajlar, frenin tuşuna basıldığı zaman; UP son ekli mesajlar, el fareden çekildiği zaman gönderilirler. Çalışma alanının dışı için NC önekli mesajlar gönderilmektedir. Örneğin biz çalışma alanının içerisinde farenin sol tuşuna bastığımızda WM_LBUTTONDOWN mesajı kuyruğa bırakılır. Örneğin pencere başlığında farenin sol tuşuna bastığımızda WM_NCLBUTTONDOWN mesajı kuyruğa bırakılır. Pencerenin çalışma alanı için oluşturulan fare mesajlarının koordinat orijini, çalışma alanının sol üst köşesidir.

```
/* mouse1.c */
….
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message) {
                case WM_LBUTTONDOWN:
                        MessageBox(hWnd, "LButton down", "Message", MB_OK);
                        break;
                case WM_NCLBUTTONDOWN:
                        MessageBox(hWnd, "NC LButton down", "Message", MB_OK);
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
/* mouse2.c */
….
#include <stdio.h>
...
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        char s[100];

        switch (message) {
                case WM_LBUTTONDOWN:
```

```
                sprintf(s, "X=%d Y=%d", LOWORD(lParam), HIWORD(lParam));
        MessageBox(hWnd, s, "Message", MB_OK);
                break;
            case WM_DESTROY:
                PostQuitMessage(0);
                break;
            default:
              return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

WM_MOUSEMOVE ve WM_NCMOUSEMOVE, fare hareket ettirildiğinde oluşan mesajlardır. Fare bir noktadan başka bir noktaya hareket ettirilmiş olsun. Bu mesajların her bir piksel için gönderileceğinin bir garantisi yoktur. Windows, sistemin o anki durumuna göre kuyruğa mesajları bırakır. Aynı nokta için 2 kere bu mesajlardan bırakılmaz. Mesajlar, yalnızca fare hareket ettirildiğinde bırakılmaktadır.

Fare mesajlarının wParam parametresi, o fare mesajı geldiğinde özel tuşlardan birine basılmış olup olmadığını belirtir. Aşağıdaki sembolik sabitler, tüm bitleri 0;yalnızca bir bitleri 1 olan sayılardır. Hangi olay gerçekleşmişse wParam'ın ilgili biti 1 olur.

MK_CONTROL	Set if the ctrl key is down.
MK_LBUTTON	Set if the left mouse button is down.
MK_MBUTTON	Set if the middle mouse button is down.
MK_RBUTTON	Set if the right mouse button is down.
MK_SHIFT	Set if the shift key is down.

Sınıf Çalışması: Çalışma alanının içinde farenin sağ tuşuna basılmışken aynı zamanda CTRL tuşuna basılmışsa MessageBox çıkartan uygulamayı yapın. Bak: mouse3.c

```
....
case WM_LBUTTONDOWN:
        if (MK_CONTROL & wParam){
        MessageBox(hWnd, "Ctrl + right", "Message", MB_OK);
        break;
        }
...
```

PostMessage ve SendMessage Foksiyonları:

Bazı olaylar geliştiğinde mesajlar, sistemin kendisi tarafından kuyruğa bırakılır. Fakat bunun yanısıra istenirse PostMessage fonksiyonu ile programcı da bir pencereye mesaj gönderebilir. PostMessage fonksiyonu, bir pencerenin handle değerini, mesajın numarasını ve wParam ile lParam parametrelerini parametre olarak alır. Mesajı, pencereyi yaratan thread'in mesaj kuyruğuna bırakır. PostMessage ile başka programların yarattığı pencerelere de mesaj gönderilebilir.

```
BOOL PostMessage(
HWND hWnd,          // handle of destination window
UINT Msg,           // message to post
WPARAM wParam,      // first message parameter
```

```
LPARAM lParam     // second message parameter
);
```

Mesajın kuyruğa bırakılması hemen işleneceği anlamına gelmez. Fonksiyon, mesajı kuyruğa bırakır ve geri döner. Mesajın işlenmesi, o thread akışının davranışına bağlıdır.

SendMessage fonksiyonu da mesaj göndermekte kullanılır. Ancak bu fonksiyon, mesajı kuyruğa bırakmaz. Doğrudan pencerenin pencere fonksiyonunu tespit eder ve o fonksiyonu çağırır. Yani bu yolla, mesaj kuyruğa bırakılmadan hızlı bir biçimde işletilmiş olur. SendMessage fonksiyonundan çıkıldığında mesajın işlenmiş olduğu garanti edilmiştir.

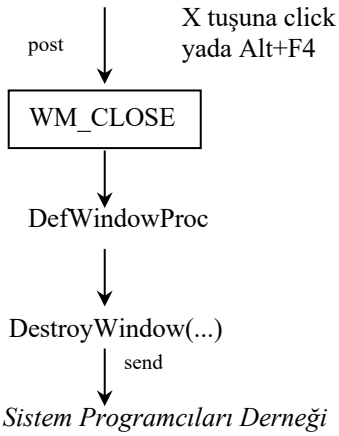
```
LRESULT SendMessage(
HWND hWnd,        // handle of destination window
UINT Msg,         // message to send
WPARAM wParam,    // first message parameter
LPARAM lParam     // second message parameter
);
```

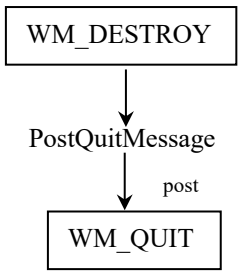
SendMessage, çağırdığı pencere fonksiyonunun geri dönüş değeri ile geri döner.

İngilizce metinlerde, bir mesajın gönderildiği durumlarda, mesajın send edilmesi ile post edilmesi farklı anlamlarda kullanılmaktadır. Mesajın send edilmesi demek, SendMessage ile gönderilmesi demektir. Post edilmesi demek PostMessage yoluyla gönderilmesi demektir.

Windows Programının Sonlandırılması

Bir Windows programı, bir dizi olay sonrası sonlanmaktadır. Kullanıcı, pencerenin X tuşuna tıkladığında yada Alt+F4 tuşlarına bastığında WM_CLOSE mesajı Windows tarafından kuyruğa bırakılır. Yani bu işlemlerle Windows sadece pencerenin kapatılmak istendiğini pencereye iletmektedir. WM_CLOSE mesajını programcı işlememişse mesaj, DefWindowProc tarafından işlenir. DefWindowProc fonksiyonunun WM_CLOSE için DestroyWindow fonksiyonunu çağırdığı belirtilmiştir. DestroyWindow fonksiyonu, pencereyi yok eden fonksiyondur. Bu fonksiyon kendi içerisinde, önce pencerenin görüntüsünü siler sonra WM_DESTROY mesajını send eder (Yani SendMessage'ı çağırır).Şimdi akış, yine pencere fonksiyonunun WM_DESTROY mesajındadır. İşte programcı bu noktada PostQuitMessage API fonksiyonunu çağırır. Bu fonksiyonun tek yaptığı şey kuyruğa WM_QUIT mesajını bırakmasıdır. PostQuitMessage fonksiyonundan dönüldüğünde akış da DetroyWindow fonksiyonuna geri döner. Bu noktada DestroyWindow, pencerenin handle alanını da yok eder. Sonra akış, sırasıyla DestroyWindow fonksiyonundan, DefWindowProc fonksiyonundan, DispatchWindow fonksiyonundan çıkarak GetMessage fonksiyonuna geri döner. GetMessage, WM_QUIT mesajını alarak 0 ile geri döner; mesaj döngüsünden çıkılır; WinMain biter ve program sonlanır.





Çıkış işleminin WM_CLOSE ile başlatılmasının nedeni, henüz pencere yok edilmeden bazı işlemlerin yapılmasına hatta işlemin iptal edilmesine olanak sağlamak içindir. Örneğin editör programlarında bir değişiklik yapılmışsa programı kapatırken çıkartılan yazı, WM_CLOSE mesajında çıkartılmaktadır.

Bir Windows programı, programlama yoluyla nasıl sonlandırılır? Bunun için 3 yol önerilebilir:

1. Kuyruğa WM_QUIT bırakmak: Bu işlem PostQuitMessage fonksiyonuyla yapılabileceği gibi doğrudan PostMessage fonksiyonu ile de yapılabilir. Bu yöntemde mesaj döngüsünden çıkıldığında henüz pencere yok edilmemiştir. Gerçi program sonlanınca bütün pencereler otomatik yok edilir ama yine de diğer yöntemlere göre tavsiye edilemeyecek bir yöntemdir. Örneğin: programcı WM_CLOSE yada WM_DESTROY mesajlarında önemli birtakım işlemler yapmış olabilir. Bu durumda bu işlemler devre dışı kalacaktır.
2. Programın Ana Penceresine DestroyWindow Uygulamak: Nasıl olsa bütün programlar programın ana penceresi yok edildiğinde PostQuitMessage fonksiyonunu çağırarak sonlanacak biçimde yazılmaktadır. Bu yöntemde WM_DESTROY mesajında yapılanlar da işlem görür.
3. Mesaj Kuyruğuna WM_CLOSE Mesajını Bırakmak: Bu yöntem, kapanma olaylarını en baştan yöntemdir. Nasıl olsa tüm programlar WM_CLOSE için kapatma işlemini başlatmaktadır.

2. ve 3. yöntemler kullanılabilecek yöntemlerdir. Eğer programı biz yazmışsak ve WM_CLOSE mesajında bir şey yapmadığımızı biliyorsak doğrudan DestroyWindow uygulayabiliriz. Fakat başka programları (başkasının yazdığı) sonlandıracaksak en güvenli yol WM_CLOSE yolunu uygulamaktır.

PostQuitMessage fonksiyonunun bir parametresi vardır.

```
VOID PostQuitMessage( int nExitCode     // exit code );
```

PostQuitMessage parametre olarak aldığı bu değeri, WM_QUIT (CLOSE???) mesajının wParam parametresine yerleştirir. İskelet Windows programının bitiş kısmı şöyledir:

```
while (GetMessage(&message, ...)){
        .....
}
return message.wParam;
```

Bu durumda program PostQuitMessage fonksiyonunda belirtilen değer ile sona ermektedir. PostQuitMessage fonksiyonunun dökümanlarında parametrenin programın çıkış kodunu belirttiği söylenir. Bunun doğru olabilmesi için şüphesiz programcının message.wParam değeri ile geri dönmesi gerekir.

Mesajların İşlenme Biçimi

Bilindiği gibi okunabilirlik bakımından case ifadelerinin uzatılmaması gerekir. Eğer case ifadelerinde uzun işlemler yapılacaksa orada bir fonksiyon çağırılıp işlemler fonksiyona yaptırılabilir. Genellikle geleneksek olarak bu tür fonksiyonlar ONXXX biçiminde isimlendirilmektedir. Örneğin OnClose, OnButtonDown gibi...Gerekirse bu fonksiyonlara mesajın parametreleri de geçirilebilir.

API Programlamada Statik Ömürlü Nesnelerin Kullanılması:

API programlamada pek çok durumda bir mesajda bir değişkenin set edilmesi ve onun başka bir mesajda kullanılması durumlarıyla karşılaşılır. Bunun için nesnenin her pencere fonksiyonu çağırıldığında yeniden yaratılmaması gerekir. Değişken ya statik yerel alınmalı yada global alınmalı. Ayrıca pencere fonksiyonunda pek çok yerel değişken tanımlamaktansa bu yerel değişkenlerin, pencere fonksiyonunun çağırdığı fonksiyonlarda tanımlanması daha uygun olur.

WM_CREATE Mesajı

Bir pencere CreateWindow yada CreateWindowEx fonksiyonu ile yaratılıp DestroyWindow fonksiyonu ile yok edilir. WM_CREATE mesajı, CreateWindow yada CreateWindowEx fonksiyonları tarafından çağırıldığında pencerenin handle alanı yaratıldıktan sonra fakat henüz pencere görünür değilken SendMessage yolu ile gönderilmektedir. Yani, akış WM_CREATE mesajına geldiğinde henüz CreateWindow yada CreateWindowEx fonksiyonları sonlanmamıştır. Bu anlamıyla WM_CREATE mesajı WM_DESTROY mesajının tersidir.

Programcı, pencere yaratılır yaratılmaz o pencereye yönelik birtakım ilk işlemler yapmak istiyor olabilir. Örneğin dinamik bir alanı tahsis edebilir yada bir dosyayı açabilir. İşte bu işlemler WM_CRATE mesajında yapılmalıdır. Bu işlemlerin geri alınması da WM_DESTROY mesajında yapılabilir.

WM_CREATE mesajının wParam parametresi kullanılmaz. Fakat lParam parametresine CREATESTRUCT isimli bir yapının adresi geçirilmektedir. Bu yapı, CreateWindow tarafından tahsis edilip mesaj send edildikten sonra silinmektedir. (Bütün yapı bildirimleri windows.h içinde). CREATESTRUCT yapısının elemanları kısaca CreateWindow fonksiyonuna geçilen parametrelerden oluşmaktadır.

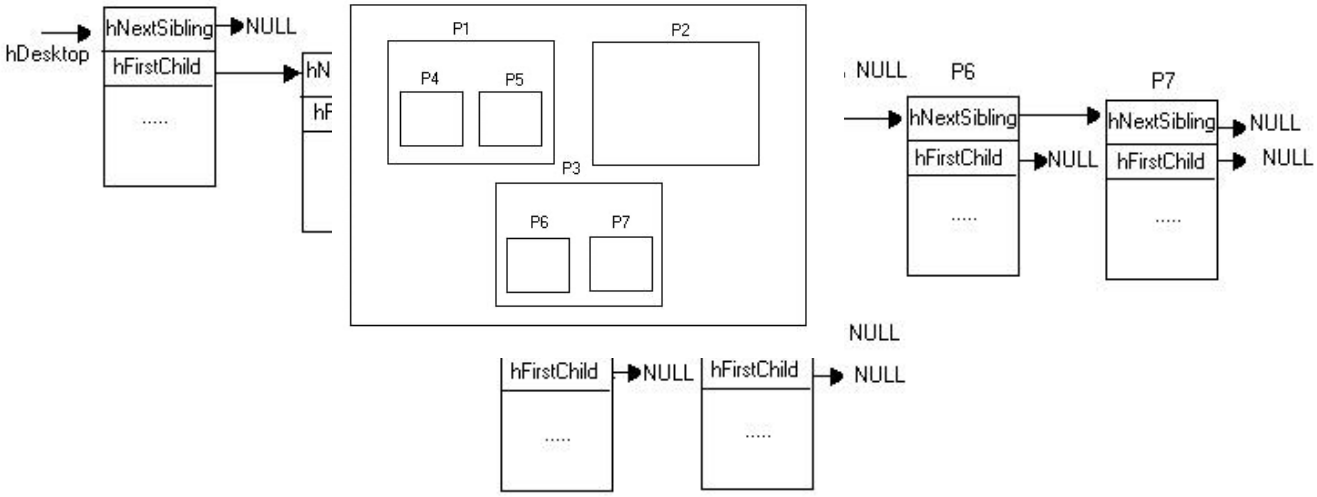
WM_CREATE mesajından normal olarak 0 değeri ile geri dönülür. –1 değeri özel bir anlam ifade eder. CreateWindow yada CreateWindowEx, pencere fonksiyonunun –1 ile geri döndüğünü görürse pencereyi yaratmaktan vazgeçip NULL değeri ile geri dönmektedir. O halde WM_CREATE mesajında yapılan ilk işlemlerde ölümcül bir hata söz konusuysa biz bu mesajdan –1 ile çıkarak pencerenin bile açılmamasını isteyebiliriz.

WINDOWS PENCERE SİSTEMİ

Windows'un kendine özgü bir pencere yapısı vardır. En dıştaki pencereye "**masaüstü (desktop)**" denilmektedir. Masaüstüne açılan pencerelere "**ana pencereler (top level windows)**" denilmektedir. Bazı pencereler, başka bir pencerenin içindedir ve onun sınırları dışına çıkamaz. Bu tür pencerelere "**alt pencereler (child window**)" denilmektedir. Alt pencerelerin de alt pencereleri olabilir. Bir alt pencerenin içinde bulunduğu pencereye o alt pencerenin "**üst penceresi (parent window)**" denilmektedir. Bir alt pencerenin tek bir üst penceresi olabilir ama bir üst pencere pek çok alt pencereye sahip olabilir. Üst pencereleri ortak olan pencerelere başka bir deyişle aynı pencerenin içerisindeki alt pencerelere "**kardeş pencereler (sibling windows)**" denilmektedir. Aslında ana pencereler masaüstünün alt pencereleridir. Başka bir deyişle bütün ana pencereler kardeş pencerelerdir.

Bu pencerelerin yanı sıra bir de owned pencere kavramı vardır. Owned pencere, bir çeşit alt penceredir. Fakat her zaman üst penceresinin üstünde görünür ve üst penceresinin sınırları dışına çıkabilir. Üst penceresi minimize edildiğinde owned pencere de minimize edilir.

Windows pencere sisteminin veri yapısı bağlı liste biçimindedir. Her hWnd handle alanı, sonraki kardeş pencereyi ve kendi ilk alt penceresini tutar. Örneğin aşağıda, bir pencere sisteminin veri yapısı çizilmiştir.



desktop

Z SIRASI

Kardeş pencereler Z sırası (Z Order) denilen bir sıraya göre bağlı listede tutulur. Z sırası, pencereler birbirini kestiği zaman onların görünüş sırasını da belirtir. En dışta görülen pencere z sırasına göre en önde penceredir. Bu pencere aynı zamanda kardeş pencere listesinde de öndedir. Kardeş pencerelerin Z sırası, karşılıklı etkileşim altında sürekli değişebilir. Kullanıcı pencereyi fareye tıklayarak öne çıkartmadıysa Z sırası, pencerelerin yaratılma sırasına göredir. Z sırasına göre en son yaratılan pencere en öndedir.

PENCERE SİSTEMİNİ DOLAŞAN API FONKSİYONLARI

Pencere sistemini dolaşan çeşitli API fonksiyonları vardır.

FindWindow Fonksiyonu

Bu fonksiyon, pencereyi WNDCLASS sınıf ismine göre ve/veya pencere başlık yazısına göre arar.

```
HWND FindWindow(
  LPCTSTR lpClassName,          // pointer to class name
  LPCTSTR lpWindowName          // pointer to window name
);
```

FindWindow, yalnızca ana pencerelerde arama yapar. Fonksiyonun 1. parametresi, aranacak pencerenin sınıf ismi, 2. parametresi başlık yazısıdır. Parametrelerden biri NULL geçilebilir. Bu durum o koşula bakılmayacağını belirtir. Fonksiyon başarılıysa pencerenin handle değerine, başarısızsa NULL değerine geri döner.

Sınıf Çalışması: Belirli bir başlık yazısına sahip ana pencereyi bulunuz. Pencereye PostMessage fonksiyonu ile WM_CLOSE mesajı göndererek pencereyi kapatınız. Bak: FindWindow

```
switch (message) {
        case WM_LBUTTONDOWN:
                OnLButtonDown(hWnd);
                break;
        case WM_DESTROY:
```

```
        PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);

}

void OnLButtonDown(HWND hWnd)
{
        HWND hDestWnd;

        hDestWnd = FindWindow(NULL, "untitled - Paint" );
        if (hDestWnd == NULL) {
                MessageBox(NULL, "Cannot Find Window", "Error", MB_OK);
                return;
        }
        PostMessage(hDestWnd, WM_QUIT, 0, 0);
}
```

EnumWindows Fonksiyonu

Bu fonksiyon, yalnızca ana pencereleri gezinen temel bir fonksiyondur. Fonksiyon, tüm ana pencereler için belirlenen bir fonksiyonu çağırır.

```
BOOL EnumWindows(
 WNDENUMPROC lpEnumFunc,  // pointer to callback function
 LPARAM lParam            // application-defined value
);
```

Fonksiyonun 1. parametresi çağırılacak fonksiyonun ismi yani adrestir. Fonksiyonun şu parametrik yapıya sahip olması gerekir:

```
BOOL CALLBACK EnumWindowsProc(
  HWND hwnd,        // handle to parent window
  LPARAM lParam     // application-defined value
);
```

Fonksiyonun 2. parametresi, 1. parametresinde belirtilen fonksiyona geçirilecek lParam değeridir. 1. parametredeki fonksiyon, dolaşıma devam edilecekse 0 dışı bir değerle; edilmeyecekse 0 değeri ile geri dönmelidir.

Sınıf Çalışması: Farenin sol tuşuna basıldığında bütün ana pencereleri kapatan programı yazınız. Fonksiyon, kendi ana penceremiz kapatmamalıdır.

```
/*  EnumWindows.c */
#include <windows.h>

LRESULT CALLBACK WndProc( HWND hWnd, UINT message,WPARAM wParam,
                                        LPARAM lParam);
void OnLButtonDown(HWND hWnd);

HWND g_hWnd;
```

```
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }
        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc( HWND hWnd, UINT message, WPARAM wParam,
                                LPARAM lParam)
{
        switch (message) {
                case WM_CREATE:
                        g_hWnd = hWnd;
                        break;
```

```
		case WM_LBUTTONDOWN:
			OnLButtonDown(hWnd);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
			return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}


BOOL CALLBACK EnumProc(HWND hWnd, LPARAM lParam)
{
	if (hWnd != g_hWnd)
		PostMessage(hWnd, WM_QUIT, 0, 0);

	return TRUE;
}

void OnLButtonDown(HWND hWnd)
{
	EnumWindows(EnumProc, 0);
}
```

EnumChildWindows Fonksiyonu

Bu fonksiyon, handle değeri verilen bir pencerenin tüm alt pencerelerini recursive bir biçimde dolaşır.

```
BOOL EnumChildWindows(
  HWND hWndParent,              // handle to parent window
  WNDENUMPROC lpEnumFunc,       // pointer to callback function
  LPARAM lParam                 // application-defined value
);
```

Fonksiyonun 1. parametresi, alt penceresi dolaşılacak pencerenin handle değeridir. 2. parametre, çağırılacak fonksiyonu belirtir. Bu fonksiyonun parametrik yapısı EnumWindows fonksiyonundaki gibi olmalıdır. 3. parametre, bu fonksiyona geçirilecek lParam parametresidir.

Masaüstü de bir penceredir. Masaüstü penceresinin handle değeri, GetDesktopWindow fonksiyonu ile alınır.

```
HWND GetDesktopWindow(VOID)
```

Bu durumda EnumChildWindows fonksiyonunun 1. parametresi olarak GetDesktopWindow fonksiyonundan alınan handle değeri verilirse, tüm pencere listesi dolaşılır.

PENCERELERLE İLGİLİ TEMEL İŞLEMLER

Bir pencerenin başlık yazısı her zaman GetWindowText fonksiyonu ile alınabilir.

```
int GetWindowText(
```

```
 HWND hWnd,        // handle to window or control with text
 LPTSTR lpString,  // address of buffer for text
 int nMaxCount     // maximum number of characters to copy
);
```

Fonksiyonun 1. parametresi, başlık yazısı alınacak pencerenin handle değeridir. 2. parametre, başlık yazısının yerleştirileceği dizinin başlangıç adresidir. 3. parametre, dizinin uzunluğu olarak girilir. Fonksiyon NULL karakter dahil olmak üzere en fazla bu parametreyle belirtilen sayıda karakteri diziye yerleştirir. Fonksiyon, başarı durumunda diziye yerleştirdiği karakter sayısıyla geri döner. Bu sayıya NULL karakter dahil değildir. Bak: GetWindowText.c

```
#include <windows.h>
#include <stdio.h>
…

void OnLButtonDown(HWND hWnd);

HWND g_hWnd;
…..

LRESULT CALLBACK WndProc( HWND hWnd, UINT message, WPARAM wParam,
                                    LPARAM lParam)
{
        switch (message) {
                case WM_CREATE:
                        g_hWnd = hWnd;
                        break;
                case WM_LBUTTONDOWN:
                        OnLButtonDown(hWnd);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

FILE *g_f;

BOOL CALLBACK EnumProc(HWND hWnd, LPARAM lParam)
{
        char text[100];

        GetWindowText(hWnd, text, 100);

        fprintf(g_f, "%s\n", text);

        return TRUE;
}
```

```
void OnLButtonDown(HWND hWnd)
{
	if ((g_f = fopen("wnd.txt", "w")) == NULL) {
		MessageBox(hWnd, "Cannot create file!..", "ERror", MB_OK);
		return;
	}

	EnumWindows(EnumProc, 0);

	fclose(g_f);
}
```

SetWindowText Fonksiyonu

SetWindowText fonksiyonu, istenildiği zaman pencerenin başlık yazısını değiştirmekte kullanılır.

```
BOOL SetWindowText(
 HWND hWnd,        // handle to window or control
 LPCTSTR lpString  // address of string
);
```

Fonksiyonun 1. parametresi başlık yazısı değiştirilecek pencerenin handle değeri; 2. parametresi set edilecek yazıdır. Bak: SetWindowText.c

```
#include <windows.h>
#include <stdio.h>
….
void OnLButtonDown(HWND hWnd);

HWND g_hWnd;
……

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch (message) {
		case WM_CREATE:
			g_hWnd = hWnd;
			break;
		case WM_LBUTTONDOWN:
			OnLButtonDown(hWnd);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}

BOOL CALLBACK EnumProc(HWND hWnd, LPARAM lParam)
{
```

```
	static int count = 0;
	char text[30];

	sprintf(text, "Number: %d", count);  //  text diye bir diziye yazılıyor. !!

	SetWindowText(hWnd, text);

	++count;

	return TRUE;
}

void OnLButtonDown(HWND hWnd)
{
	EnumWindows(EnumProc, 0);
}
```

Bir pencere, fare ve klavye mesajlarına EnableWindow API fonksiyonu yoluyla kapatılıp açılabilir.

```
BOOL EnableWindow(
  HWND hWnd,    // handle to window
  BOOL bEnable   // flag for enabling or disabling input
);
```

Fonksiyonun 1. parametresi, ilgili pencerenin handle değeridir. 2. parametre, pencerenin etkin mi yoksa pasif duruma mı getirileceğini belirtir. Pencere pasif duruma getirilirse, görüntüsü kaybolmaz; yalnızca fare ve klavye mesajları artık pencereye gitmez. Bak: Enablewindow

```
#include <windows.h>
#include <stdio.h>
….
void OnLButtonDown(HWND hWnd);

HWND g_hWnd;

…

LRESULT CALLBACK WndProc( HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam)
{
	switch (message) {
		case WM_CREATE:
			g_hWnd = hWnd;
			break;
		case WM_LBUTTONDOWN:
			OnLButtonDown(hWnd);
			break;
		case WM_RBUTTONDOWN:
			MessageBox(hWnd, "Test", "Message", MB_OK);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
```

```
                    break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnLButtonDown(HWND hWnd)
{
        EnableWindow(hWnd, FALSE);
}
```

Pencere, ShowWindow fonksiyonuyla aşağıdaki gibi görünmez yapılabilir:

```
ShowWindow(hWnd, SW_HIDE);
```

Pencereyi yeniden görünür yapmak için yine aynı fonksiyon kullanılır:

```
ShowWindow(hWnd, SW_SHOW);
```

MoveWindow Fonksiyonu

MoveWindow, bir pencerenin konumunu ve boyutunu değiştirmek amacıyla kullanılır.

```
BOOL MoveWindow(
  HWND hWnd,        // handle to window
  int X,            // horizontal position
  int Y,            // vertical position
  int nWidth,       // width
  int nHeight,      // height
  BOOL bRepaint     // repaint flag
);
```

Fonksiyonun 1. parametresi, pencerenin handle değeridir. Fonksiyonun diğer parametreleri ise pencerenin yeni konumunun sol üst köşegeni ve boyutudur. Son parametre, pencere taşındıktan sonra içerisindeki görüntünün tazelenip tazelenmeyeceğini belirtir. TRUE geçilebilir. Orijin noktası ana pencereler için masaüstünün sol üst köşesi; alt pencereler için üst pencerenin çalışma alanının sol üst köşesidir.

GetClientRect Fonksiyonu

Bu fonksiyon, çalışma alanının genişliğini ve yüksekliğini almak için kullanılır.

```
BOOL GetClientRect(
 HWND hWnd,        // handle to window
 LPRECT lpRect     // address of structure for client coordinates
);
```

RECT yapısı şöyledir:

```
typedef struct tagRECT {
  LONG left;
  LONG top;
```

```
  LONG right;
  LONG bottom;
} RECT;
```

GetClientRect fonksiyonunun 1. parametresi, pencerenin handle değeri; 2. parametresi, çalışma alanının koordinatlarının yerleştirileceği RECT yapısının adresidir. Fonksiyon her zaman sol üst köşegeni (0,0) olarak verir.

GetWindowRect Fonksiyonu

Bu fonksiyon, pencerenin tamamının koordinatlarını masaüstünün sol üst köşesine göre vermektedir.

```
BOOL GetWindowRect(
 HWND hWnd,        // handle to window
 LPRECT lpRect     // address of structure for window coordinates
);
```

Sınıf Çalışması: Farenin sol tuşuna basıldığında pencereyi her yönde 50 birim büyüten; sağ tuşa basıldığında her yönden 50 birim küçülten programı yazınız.

```
....
RECT g_rect;
…..
LRESULT CALLBACK WndProc( HWND hWnd, UINT message, WPARAM wParam,
                                    LPARAM lParam)
{
        static int widht, height;

        switch (message) {
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                case WM_LBUTTONDOWN:
                        GetWindowRect(hWnd, &g_rect);

                        widht = g_rect.right - g_rect.left;
                        height = g_rect.bottom - g_rect.top;

                        MoveWindow(hWnd, g_rect.left - 50, g_rect.top - 50, widht + 100,
                                height + 100, TRUE);
                        break;

                case WM_RBUTTONDOWN:
                        GetWindowRect(hWnd, &g_rect);

                        widht = g_rect.right - g_rect.left;
                        height = g_rect.bottom - g_rect.top;

                        MoveWindow(hWnd, g_rect.left + 50, g_rect.top + 50, widht - 100,
                                height - 100, TRUE);
                        break;
                default:
```

```
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
.......
```

WM_SIZE Mesajı

Bu mesaj, pencerenin boyutu değiştirildiğinde kuyruğa bırakılmaktadır. Mesajın parametreleri şöyledir:

LOWORD(lParam) → Çalışma alanının yeni genişliği
HIWORD(lParam) → Çalışma alanınn yeni yüksekliği
wParam, şunlardan biri olabilir: SIZE_MAXIMIZED
SIZE_MINIMIZED
SIZE_RESTORED

```
.....
switch (message) {
        case WM_SIZE:
                switch (wParam) {
                        case SIZE_MAXIMIZED:
                                MessageBox(hWnd, "Maximized", "Message", MB_OK);
                                break;
                        case SIZE_MINIMIZED:
                                MessageBox(hWnd, "Minimized", "Message", MB_OK);
                                break;
                        case SIZE_RESTORED:
                                MessageBox(hWnd, "Restored", "Message", MB_OK);
                                break;
                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
}
.....
```

Anahtar Notlar:

Windows'ta bir pencerenin boyutunun değiştirilmesi ve taşınması işlemi seçenekli olarak 2 biçimde yapılmaktadır.

1. *Pencerenin kendisinin taşınması ve boyutunun değiştirilmesi*
2. *Pencerenin çerçeve çizgilerinin işleme sokulması; el fareden bırakılınca gerçek işlemin yapılması*

Bu seçenekler, Control Panel/Display/Effects/Show window contents while dragging seçeneği ile belirlenir. WM_SIZE mesajı birinci durumda boyut değiştirilme sürecinde sürekli gelir. İkinci durumda ise yalnızca el fareden bırakılınca gelir.

ALT PENCERELERİN YARATILMASI

Daha önce gördüğümüz gibi ana pencere yaratmak için CreateWindow yada CreateWindowEx fonksiyonunun hWndParent parametresi NULL olarak girilir; pencere biçimi parametresine de WS_OVERLAPPED eklenir. Alt pencere yaratmak için ise hWndParnet parametresi üst pencerenin handle değeri olarak girilmeli; pencere biçimi parametresine de WS_CHILD eklenmelidir. Owned pencere yaratmak isteniyorsa da hWndParent parametresine üst pencerenin handle değeri girilmeli; pencere biçim parametresine ise WS_POPUP girilmelidir.

hWndParent	**Pencere biçimi**
NULL Üst pencere handle değeri Üst pencere handle değeri	WS_OVERLAPPED → Ana pencere WS_CHILD → Alt pencere WS_POPUP → Owned pencere

Bilindiği gibi pencerenin davranışı, pencere fonksiyonu tarafından belirlenir. Eğer biz alt pencereyi de üst pencere ile aynı pencere sınıfını kullanarak yaratırsak bunların pencere fonksiyonları aynı olur ve pencereler aynı biçimde davranır. Pencerelerin davranışlarını değiştirmek için alt pencere için farklı bir pencere sınıfı, dolayısıyla farklı bir pencere fonksiyonu oluşturmak gerekir.

Alt pencere nerede yaratılmalıdır? İki yerde yaratabiliriz:
1. Üst pencere yaratıldıktan sonra herhangi bir yerde ve herhangi bir zaman
2. Üst pencerenin WM_CREATE mesajında

Alt Pencere ID Değeri:

Alt pencereler, ID denilen word sınırlarında bir değere sahiptir. ID değerleri, kardeş pencereleri birbirinden ayırmakta kullanılır ve daha sonra ayrıntılı incelenecektir. Ana pencereler ID değerine sahip değildir. Yalnızca alt pencerelerin ID değerleri vardır. Alt pencere ID değeri, CreateWindow fonksiyonunun 9. parametresi olan hMenu parametresi ile girilir. Bu parametre, ana pencereler için menü belirlemesi amacıyla, alt pencereler için ID belirlemesi amacıyla kullanılır. Şüphesiz ID değerini, tür dönüştürme operatörüyle HMENU türüne dönüştürerek fonksiyona vermek gerekir.

Alt Pencerenin Konumu Ve Özellikleri:

Alt pencere yaratılırken verilen yaratım koordinatları, üst pencerenin çalışma alanının (client area) sol üst köşesi orijinlidir. Alt pencere yaratılırken WS_VISIBLE pencere biçimi kullanılmazsa alt pencere yaratılır ama görünür olmaz. Bu durumda görünür yapmak için daha sonra ShowWindow uygulamak gerekir. Alt pencere, sınır çizgisine sahip olmak zorunda değildir. Eğer WS_BORDER yada WS_THICKFRAME kullanılmazsa alt pencere sınır çizgilerine sahip olmaz. Alt pencere farklı pencere sınıfından yaratıldığına göre onun zemin rengini yada fare oku biçimini değiştirebiliriz. Alt pencereye MoveWindow uygulanırsa fonksiyonda belirtilen konum, çalışma alanının sol üst köşesi orijinlidir.

Alt pencere de normal bir penceredir. Alt pencerenin ve üst pencerenin mesajları aynı kuyruğa bırakılır. Her iki mesaj da GetMessage ile alınır. DispatchMessage, hangi pencere fonksiyonunun çağırılacağını belirler. Üst pencereye DestroyWindow uygulandığında recursive olarak onun bütün alt pencereleri de yok edilir. Fakat alt pencereye DestroyWindow uygulanırsa onun alt pencereleri dışında diğer pencereler bu işlemden etkilenmez.

Bir alt pencere üzerinde fare işlemi yapıldığında eğer alt pencere pasif durumda değilse fare mesajları yalnızca alt pencereye gönderilir. Eğer alt pencere pasif durumdaysa (örneğin EnableWindow fonksiyonuyla

pasif duruma geçirilmişse) bu durumda sanki alt pencere yokmuş gibi işlem yapılır. Yani fare mesajları bu durumda onun üst penceresine gönderilecektir.

```
/*   Child.c  */

#include <windows.h>

#define ID_CHILD                100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam);
LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message,WPARAM wParam,
                                            LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd, hWndChild;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "SampleChild";
                wndClass.lpfnWndProc = (WNDPROC) WndChildProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }
```

```
        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        hWndChild = CreateWindow("SampleChild", "Generic App",
                WS_CHILD|WS_VISIBLE|WS_BORDER|WS_CAPTION|WS_THICKFRAME,
                100, 100, 100, 100, hWnd,
                (HMENU) ID_CHILD,
                hInstance,
                NULL);

        if (!hWndChild)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam)
{
        switch (message) {
                case WM_LBUTTONDOWN:
                        MessageBox(hWnd, "Parent window", "Message", MB_OK);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam)
```

```
{
        switch (message) {
                case WM_LBUTTONDOWN:
                        MessageBox(hWnd, "Child window", "Message", MB_OK);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
/* Child2.C */

#include <windows.h>

#define ID_CHILD                100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam);
LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message,WPARAM wParam,
                                                LPARAM lParam);

HINSTANCE g_hInstance;

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
```

```
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) CreateSolidBrush(RGB(255, 0, 0));
		wndClass.hCursor = LoadCursor(NULL,IDC_WAIT);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "SampleChild";
		wndClass.lpfnWndProc = (WNDPROC) WndChildProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	g_hInstance = hInstance;

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;


	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
					LPARAM lParam)
{
	static HWND hWndChild;

	switch (message) {
		case WM_CREATE:
			hWndChild = CreateWindow("SampleChild", "Generic App",
				WS_CHILD|WS_VISIBLE,
					100, 100, 100, 100, hWnd,
					(HMENU) ID_CHILD,
					((LPCREATESTRUCT) lParam)->hInstance,
					NULL);
```

```
                if (!hWndChild)
                        return -1;
                EnableWindow(hWndChild, FALSE);
                break;

        case WM_LBUTTONDOWN:
                MessageBox(hWnd, "Parent window", "Message", MB_OK);
                ShowWindow(hWndChild, SW_SHOW);
                break;

        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message) {
        case WM_MOUSEMOVE:
                MessageBeep(0xFFFFFFFF);
                break;
        case WM_LBUTTONDOWN:
                MessageBox(hWnd, "Child window", "Message", MB_OK);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

ÜST PENCERE VE ALT PENCERE ÜZERİNDE BELİRLEME İŞLEMİ YAPAN API FONKSİYONLARI

Bir alt pencerenin tek bir üst penceresi olabilir. Bir alt pencerenin üst penceresi GetParent fonksiyonu ile elde edilebilir.

```
HWND GetParent(
 HWND hWnd     // handle to child window
);
```

Fonksiyonun parametresi alt pencerenin handle değeri; geri dönüş değeri ise onun üst penceresinin handle değeridir. Eğer parametre olarak bir ana pencere verilirse fonksiyon NULL değeri ile geri döner.

Bir üst pencerenin pek çok alt penceresi olabilir. Yani biz üst pencerenin handle değerini vererek alt pencerenin handle değerini alamayız. İşte, alt pencerelerin ID değerleri aslında kardeş pencereleri

birbirinden ayırmak için düşünülmüştür. Üst pencerenin handle değeri ve alt pencerenin ID değeri verildiğinde alt pencerenin handle değerini veren GetDlgItem isimli bir fonksiyon vardır.

```
HWND GetDlgItem(
 HWND hWnd,        // handle of dialog box
 int id            // identifier of control
);
```

Fonksiyonun 1. parametresi üst pencerenin handle değeri, 2. parametresi alt pencerenin ID değeridir. Fonksiyon, alt pencerenin handle değerine geri döner. Başarısızlık durumunda ise fonksiyon NULL değeri ile geri dönmektedir. Maalesef bu fonksiyon sanki sadece diyalog pencerelerine özgü çalışıyormuş gibi kötü isimlendirilmiştir. Her ne kadar alt pencerenin handle değerini bulma daha çok diyalog pencerelerinde ortaya çıkıyorsa da fonksiyon genel bir fonksiyondur; her türlü pencere için aynı biçimde çalışmaktadır.

Kardeş pencerelerin ID değerleri birbirinden farklı olmak zorunda değildir. Fakat ID değeri alt pencereleri ayırmak için düşünüldüğünden bunların farklı yapılması anlamlıdır. Aynı ID'ye sahip birden fazla kardeş pencere varsa GetDlgItem, Z sırasına göre ilk bulduğu alt pencere ile geri döner.

Pencere handle değeri sistem genelinde tek (unique) olan bir değerdir. Alt pencerenin handle değeri verildiğinde onun ID değeri GetDlgCtrlID fonksiyonu ile elde edilebilir.

```
int GetDlgCtrlID(
 HWND hWnd         // handle of control
);
```

Fonksiyonun parametresi pencerenin handle değeri, geri dönüş değeri ise alt pencerenin ID değeridir. Parametresi ile belirtilen pencere ana pencere ise yada fonksiyon başarısız ise fonksiyon 0 ile geri döner. 0 değeri, ID olarak kullanılamaz.

Üst Pencere İle Alt Pencere Arası Mesajlaşma İşlemleri

Üst pencere ile alt pencere arasında mesaj gönderme işlemlerine çok sık rastlanmaktadır. Genellikle bu tür mesajlaşma PostMessage ile değil SendMessage fonksiyonu ile yapılır. Üst pencere alt pencerenin handle değerini zaten onu kendisi yarattığı için bilmektedir. Alt pencere de üst pencerenin handle değerini istediği zaman GetParent fonksiyonunun kullanarak istediği zaman elde edebilir.

Pencereler arasında mesajlaşma yaparken bir mesaj numarası gerekmektedir. Oysa bazı mesaj numaraları Windows'un kendisi tarafından kullanılır. Örneğin WN_LBUTTONDOWN, bir fare olayı ile ilgili bir mesajdır. Biz pencereye bu mesajı gönderirsek pencere fareye basıldığını zannedebilir. İşte Windows'un bu standart mesajları, 0'dan WM_USER – 1 değerine kadar olan mesajlardır. WM_USER, bir sınır değer olarak düşünülmüş bir sayıdır. WM_USER değerinden 0x7FFF değerine kadar olan değerleri Windows kullanmamaktadır. Yani bu mesaj numaraları mesajlaşmada kullanılabilir. Bak Child3.

```
#include <windows.h>

#define ID_CHILD                100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                                    LPARAM lParam);
LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message,WPARAM wParam,
                                    LPARAM lParam);
```

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "SampleChild";
                wndClass.lpfnWndProc = (WNDPROC) WndChildProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;
```

```
        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam)
{
        static HWND s_hWndChild;

        switch (message) {
                case WM_CREATE:
                        s_hWndChild = CreateWindow("SampleChild", "Generic App",

        WS_CHILD|WS_VISIBLE|WS_BORDER|WS_CAPTION|WS_THICKFRAME,
                                        100, 100, 100, 100, hWnd,
                                        (HMENU) ID_CHILD,
                                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        if (!s_hWndChild)
                                return -1;
                        break;
                case WM_LBUTTONDOWN:
                        SendMessage(s_hWndChild, WM_USER, 0, 0);
                        break;
                case WM_USER:
                        MessageBox(hWnd, "Message From Child!..", "Message", MB_OK);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
        switch (message) {
                case WM_USER:
                        MessageBox(hWnd, "Message From Parent!..", "Message", MB_OK);
                        break;
                case WM_LBUTTONDOWN:
                        SendMessage(GetParent(hWnd), WM_USER, 0, 0);
                        break;

                default:
```

```
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

Sınıf Çalışması: Üst pencerenin WM_CREATE mesajında bir döngü içerisinde 10 tane alt pencere yaratınız. Bu alt pencerelerin handle değerlerini statik bir HWND dizisinde saklayınız. Tüm alt pencereler aynı pencere sınıfından yaratılacak dolayısıyla aynı pencere fonksiyonuna sahip olacaktır. Pencereleri yaratırken bunlara ID değeri olarak 100'den başlayan ardışıl sayılar veriniz. Alt pencereler de farenin sol tuşuna basıldığında alt pencere üst pencereye WM_USER mesajı göndersin. Mesajı alan üst pencere MessageBox içerisinde mesajın hangi alt pencereden geldiğini yazdırsın. Bak: child4.c, child.c -> DÜZELECEK

/* Child4.c */

```
#include <windows.h>

#define CHILD_SIZE      10
#define TEXT_SIZE       100
#define ID_CHILD        100


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam);
LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message,WPARAM wParam,
                                            LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) CreateSolidBrush(RGB(255, 0, 0));
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "SampleChild";
                wndClass.lpfnWndProc = (WNDPROC) WndChildProc;
                if (!RegisterClass(&wndClass))
```

```
                return -1;
        }

        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND s_hWndChild[CHILD_SIZE];
        static char s_text[TEXT_SIZE];
        int i;

        switch (message) {
                case WM_CREATE:
                        for (i = 0; i < CHILD_SIZE; ++i) {
                                sprintf(s_text, "Child: %d", i + 1);
                                s_hWndChild[i] = CreateWindow("SampleChild", s_text,
                                        WS_CHILD|WS_BORDER|WS_CAPTION|WS_THICKFRAME,
                                        100, 100, 100, 100, hWnd,
                                        (HMENU) (ID_CHILD + i),
                                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                                if (!s_hWndChild[i])
                                        return -1;

                                ShowWindow(s_hWndChild[i], SW_MINIMIZE);
                        }
                        break;
                case WM_USER:
                        {
                                char text[TEXT_SIZE];
                                int id = GetDlgCtrlID((HWND) wParam);

                                sprintf(text, "Message from Child: %d", id - ID_CHILD + 1);
                                MessageBox(hWnd, text, "Message", MB_OK);
                        }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
```

```
}

LRESULT CALLBACK WndChildProc( HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam)
{
        switch (message) {
                case WM_LBUTTONDOWN:
                        SendMessage(GetParent(hWnd), WM_USER, (HWND) (hWnd), 0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

/* Child.c */

```
#include <windows.h>

#define CHILD_SIZE      10
#define TEXT_SIZE       100
#define ID_CHILD        100


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message,WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) CreateSolidBrush(RGB(255, 0, 0));
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "SampleChild";
                wndClass.lpfnWndProc = (WNDPROC) WndChildProc;
                if (!RegisterClass(&wndClass))
                        return -1;
```

```
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND s_hWndChild[CHILD_SIZE];
	static char s_text[TEXT_SIZE];
	int i;

	switch (message) {
		case WM_CREATE:
			for (i = 0; i < CHILD_SIZE; ++i) {
				sprintf(s_text, "Child: %d", i + 1);
				s_hWndChild[i] = CreateWindow("SampleChild", s_text,
					WS_CHILD|WS_BORDER|WS_CAPTION|WS_THICKFRAME,
					100, 100, 100, 100, hWnd,
					(HMENU) (ID_CHILD + i),
					((LPCREATESTRUCT) lParam)->hInstance, NULL);
				if (!s_hWndChild[i])
					return -1;

				ShowWindow(s_hWndChild[i], SW_MINIMIZE);
			}

			break;
		case WM_USER:
			{
				char text[TEXT_SIZE];
				int id = GetDlgCtrlID((HWND) wParam);

				sprintf(text, "Message from Child: %d", id - ID_CHILD + 1);
				MessageBox(hWnd, text, "Message", MB_OK);
			}
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
```

```
}

LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message) {
                case WM_LBUTTONDOWN:
                        SendMessage(GetParent(hWnd), WM_USER, (HWND) (hWnd), 0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

STANDART WINDOWS KONTROLLERİ

Windows'ta sistem genelinde sistem açılırken register ettirilmiş bazı pencere sınıfları vardır. Bu pencere sınıflarının pencere fonksiyonları USER32.DLL içerisindedir. Yani bu pencere sınıflarının pencere fonksiyonları yazılmış bir biçimde bellekte bulunmaktadır. Biz bu pencere sınıflarını kullanarak alt pencereler yaratabiliriz. Bu durumda bu pencereler üzerinde işlemler gerçekleştiğinde mesajlar yine kuyruğa bırakılacak, kuyruktan GetMessage fonksiyonu ile alınacak ve DispatchMessage fonksiyonu ile pencere fonksiyonu çağırılacaktır. DispatchMessage, şüphesiz bu durumda USER32.DLL içerisindeki zaten hazır bir biçimde bulunan bu pencere fonksiyonlarını çağıracaktır. İşte, Windows'un içerisinde hazır bir biçimde bulunan bu pencere sınıflarına "Windows Kontrolleri" denilmektedir. Windows programlarında gördüğümüz düğmeler, listeleme kutuları (list box), edit alanları, araç çubukları hep birer standart alt penceredir. Özetle:

- Standart alt pencerelerin pencere fonksiyonları zaten Windows'un içerisinde yazılmış bir biçimde bulunmaktadır.
- Standart alt pencerelere ilişkin pencere sınıfları, Windows yüklenirken sistem genelinde register ettirilmiş durumdadır.
- Eğer biz bu pencerelerin pencere sınıf isimlerini bilirsek CreateWindow yada CreateWindowEx fonksiyonlarıyla da bunları yaratabiliriz.
- Örneğin bir düğme kontrolünü yaratmış olalım. Kontrolün üstüne tıkladığımızda fare mesajı Windows tarafından bu pencereyi hedef alarak kuyruğa yerleştirilecektir. Sonuçta düğme kontrolünün pencere fonksiyonu çağırılacaktır. Tuşa basılma görüntüsünü fare mesajında düğme kontrolünün pencere fonksiyonu gerçekleştirmektedir.

Standart Windows kontrolleri 2 kısma ayrılabilir:

1. Win16 zamanlarından beri varolan kontroller: Bu kontroller daha Windows 3.x zamanlarından beri benzer biçimde varlığını sürdürmektedir. Örneğin düğmeler (push button), listeleme kutuları (list box), seçimli listeleme kutuları (combo box), seçenek kutuları (check box), statik kontrol gibi...
2. Win32 sistemleri ile standart hale getirilmiş olan gelişkin kontroller: araç çubukları, ağaç kontrolleri, liste kontrolleri, durum penceresi gibi yetenekli pek çok kontrol, daha sonra eklenmiştir. Windows API kursunda yalnızca temel kontrollerin kullanımı üzerinde durulacaktır.

Kontrollerin Yaratılması

Kontroller normal CreateWindow fonksiyonu ile yaratılır. Fonksiyonun 1. parametresindeki sınıf ismi, kontrolün sınıf ismi olarak girilir. Fonksiyonun 2. parametresi olan pencere başlık yazısının bazı kontroller için anlamı yoktur. Fakat bazı kontrollerde bu yazı önemlidir. Örneğin düğme kontrolünde pencere başlık

yazısı düğmenin üzerine çıkacak yazıyı belirtir. Edit kontrolünde pencere başlık yazısı edit kontrolündeki tüm yazıyı temsil eder. Örneğin biz edit control içerisindeki tüm yazıyı GetWindowText fonksiyonu ile alabiliriz. Pencere başlık yazısı ile bu yazıların bir alakası yokmuş gibi görülebilir. Pencere başlık yazısının oluşturulması WM_GETTEXT ve WM_SETTEXT gibi mesajlarla yapılmaktadır ve bu konu ileride ele alınacaktır.

Windows'ta her pencere için anlamlı olan pencere biçimleri WS_ öneki ile isimlendirilmiştir. Fakat bazı kontrollerin özel pencere biçimleri de vardır. Bu pencere biçimleri o kontroller yaratılırken kullanılabilir. Örneğin edit kontrolüne özgü pencere biçimleri ES_ öneki ile, düğme kontrolüne ilişkin pencere biçimleri BS_ öneki ile isimlendirilmiştir. Kontroller bir alt pencere biçiminde yaratılacağına göre WS_CHILD pencere biçimini içermelidir. Bu biçime ek olarak WS_VISIBLE ve WS_BORDER gibi biçimler de yaratım sırasında kullanılabilir. Ayrıca yukarıda belirtildiği gibi özel pencere biçimlerini de kombine edebiliriz.

Statik Kontrol

Statik kontrol, en basit kontroldür. Tek amacı, bir alt pencere üzerinde bir yazıyı çıkartmaktır. Pencere default pasif durumda olduğu için fare ve klavye mesajlarına kapalıdır. Statik kontrol "static" sınıf ismi kullanılarak yaratır. Kontrol, CreateWindow fonksiyonunun pencere başlığı parametresiyle belirtilen yazıyı alt pencerenin içerisine yazar. Dolayısıyla bu yazıyı istediğimiz zaman SetWindowText fonksiyonu ile değiştirebiliriz. Bak: Satic.c

```
/* Static.c */

#include <windows.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}
	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);
```

```
	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hStatic;

	switch (message) {
		case WM_CREATE:
			hStatic = CreateWindow("static", "Test", WS_CHILD|WS_VISIBLE, 100, 100, 100,
					100, hWnd, (HMENU) 100,
					((LPCREATESTRUCT) lParam)->hInstance, NULL);
			break;
		case WM_LBUTTONDOWN:
			SetWindowText(hStatic, "New Text\nFalan filan");
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

Statik kontrol yaratılırken SS_ önekli özel pencere biçimleri de kombine edilebilir. Önemli pencere biçimleri şunlardır:

SS_CENTER, SS_LEFT, SS_RIGHT: Bu pencere biçimleri yazıyı hizalamak amacıyla kullanılır. Örneğin:

```
hStatic = CreateWindow("static", "Test",
                       WS_CHILD|WS_VISIBLE|WS_BORDER|SS_CENTER ,
                       100, 100, 300, 100, hWnd, (HMENU) 100,
                       ((LPCREATESTRUCT) lParam)->hInstance, NULL);
```

SS_NOWORDWRAP: Default olarak statik kontrol, pencerenin sağına gelindiğinde sözcüğü bölmeden aşağı satıra geçiş yapar. Bu pencere biçimi, aşağı satıra geçişi engellemektedir.
Bak: Static1.c, Static2.c, Static3.c

```
/* Static1.c */
….
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hStatic;

	switch (message) {
		case WM_CREATE:
			hStatic = CreateWindow("static", "",
```

```
                WS_CHILD|WS_VISIBLE|SS_LEFT|SS_LEFTNOWORDWRAP ,
                100, 100, 300, 100, hWnd, (HMENU) 100,
                ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                break;
        case WM_LBUTTONDOWN:
                {
                        int i;
                        char text[30];

                        for (i = 0; i < 10; ++i) {
                                sprintf(text, "%d", i);
                                SetWindowText(hStatic, text);
                                Sleep(1000);
                        }
                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

```
/* Static2.c */
….
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HWND hStatic;

    switch (message) {
        case WM_CREATE:
            hStatic = CreateWindow("static", "",
              WS_CHILD|WS_VISIBLE|SS_LEFT|SS_LEFTNOWORDWRAP ,
              0, 0, 0, 0, hWnd, (HMENU) 100,
              ((LPCREATESTRUCT) lParam)->hInstance, NULL);
            break;
        case WM_LBUTTONDOWN:
            {
                int i;
                char text[30];

                for (i = 0; i < 10; ++i) {
                    sprintf(text, "%d", i);
                    SetWindowText(hStatic, text);
                    Sleep(1000);
                }
            }
            break;
        case WM_SIZE:
            MoveWindow(hStatic, 0, 0, LOWORD(lParam),
```

```
                                        HIWORD(lParam), TRUE);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
/* Static3.c */
….
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hStatic;

        switch (message) {
                case WM_CREATE:
                        hStatic = CreateWindow("static", "",
                          WS_CHILD|WS_VISIBLE|WS_VSCROLL|SS_LEFT|SS_LEFTNOWORDWRAP,
                          0, 0, 0, 0, hWnd, (HMENU) 100,
                          ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        break;
                case WM_LBUTTONDOWN:
                        {
                                FILE *f;
                                char *pText;
                                long size;

                                if ((f = fopen("static3.c", "r")) == NULL) {
                                        MessageBox(hWnd, "Cannot open file!..", "Error", MB_OK);
                                        break;
                                }
                                fseek(f, 0, SEEK_END);
                                size = ftell(f);
                                fseek(f, 0, SEEK_SET);

                                pText = (char *) malloc(size + 1);
                                fread(pText, 1, size, f);
                                pText[size] = '\0';
                                SetWindowText(hStatic, pText);
                                free(pText);
                                fclose(f);
                        }
                        break;
                case WM_SIZE:
                        MoveWindow(hStatic, 0, 0, LOWORD(lParam),
                                        HIWORD(lParam), TRUE);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

Kontrollere Gönderilen Mesajlar

Kontroller yaratıldıktan sonra onlara birtakım faydalı işlemler yaptırabilmek için standart olarak önceden tespit edilmiş mesajlardan faydalanılır. Bu mesajlar, windows.h dosyası içerisinde XM_YYY biçiminde sembolik sabitlerle isimlendirilmiştir. Burada X, kontrol isminden yapılan bir kısaltma; YYY ise mesajın anlamına yönelik bir kısaltmadır. Mesaj, kontrole SendMessage yada PostMessage yolu ile gönderilebilir. Fakat, mesajın senkron bir biçimde SendMessage ile gönderilmesi daha uygundur. Mesaj gönderilirken, wParam ve lParam değerlerinin ne anlam ifade ettiği programcı tarafından bilinmelidir.

Kontrollerin Üst Pencereye Gönderdiği Mesajlar

Alt pencerelerde çeşitli olaylar olduğunda bu olaydan üst pencereyi haberdar etmek için alt pencereler, üst pencerelere mesaj gönderirler. Örneğin düğme bir alt penceredir. Yani düğmeye tıklandığında mesajı düğmenin pencere fonksiyonu işleyecektir. Peki biz düğmeye tıklandığını nereden anlayacağız? İşte düğme, kendisine tıklandığını bize mesaj göndererek ifade edecektir. Yani özetle alt pencerelerdeki birtakım olayları biz, alt pencerelerin gönderdiği mesajlarla anlayabiliriz.

Standart kontroller genel olarak üst pencereye WM_COMMAND isimli mesajı gönderirler. Bu mesaj, kontrol tarafından SendMessage olarak gönderilir. WM_COMMAND mesajının parametreleri (wParam, lParam parametrelerini anlamalıyız.) mesajın hangi kontrolden ve ne nedenle gönderildiğini belirtir.

***WM_COMMAND mesajının parametreleri**:*

LOWORD(wParam) → Kontrolün id değeri
HIWORD(wParam) → Mesajın ne anlamda gönderildiğine ilişkin işlem kodu (notification code)
lParam → Kontrolün handle değeri

WM_COMMAND mesajını işleyen programcı ilk iş olarak mesajın hangi kontrolden geldiğini tespit etmeye çalışır. Bunun için LOWORD(wParam) değerine bakar. Mesajın hangi kontrolden geldiğini tespit ettikten sonra bu kez mesajın ne amaçla gönderildiğini belirlemeye çalışır. Mesajın neden gönderildiğine ilişkin işlem kodları önceden tespit edilmiştir. Bu işlem kodları XN_YYY biçiminde sembolik sabitlerle define edilmiştir. Burada X, kontrolün türüne ilişkin bir kısaltma; YYY ise, mesajın anlamına ilişkin bir kısaltmadır.

Düğme Kontrolü

Düğme kontrolü (push button), Windows programlarında en sık kullanılan kontroldür. Bir olayı başlatmak yada bitirmek amacıyla kullanılır. Düğmeye tıklandığında düğme önce basılma hareketini yapar, el fareden çekilince bu kez çekilme hareketini yapar. Düğmenin basılmış bir biçimde bekletilmesi söz konusu değildir. Düğme üzerinde çift tıklama biçiminde bir kabul gören davranış yoktur.

Düğme kontrolünün sınıf ismi "button" biçimindedir. Yani CreateWindow fonksiyonunda sınıf ismi olarak "button" girilmelidir. Fonksiyonun pencere başlığı parametresi, düğmenin üzerine çıkartılacak yazıyı belirler. (Bak: button.c) buton sınıfı aslında yalnızca düğme kontrolü için kullanılan bir sınıf değildir. Düğmelerin, seçenek kutularının (check box), radyo düğmelerinin pencere fonksiyonları aynı fonksiyonlardır yani bu üç pencere de button sınıfı ile yaratılırlar.

/* button.c */

```
#include <windows.h>
#include <stdio.h>
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}
	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hButton;

	switch (message) {
		case WM_CREATE:
			hButton = CreateWindow("button", "Test",
					WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
					100, 100, 100, 100, hWnd, (HMENU) 100,
					((LPCREATESTRUCT) lParam)->hInstance, NULL);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
```

```
	return 0;
}
```

button sınıfının özel pencere biçimleri şunlardır:

BS_PUSHBUTTON: Düğme kontrolü için bu pencere biçimi belirtilmek zorundadır. Diğer pencere biçimleri, diğer düğme kontrollerine ilişkindir.

Düğme Kontrolüne Gönderilen Mesajlar

Düğme kontrolüne WM_CLICK mesajı gönderilirse programlama yoluyla düğmeye basılma çekilme hareketi yaptırılır. Mesajın wParam ve lParam parametreleri kullanılmaz; sıfırlanmalıdır. Örneğin:

SendMessage(hButton, BM_CLICK, 0 ,0);
BAk: button1.c, button2.c

/* button1.c */

```
…..
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hButton;

	switch (message) {
		case WM_CREATE:
			hButton = CreateWindow("button", "Test",
					WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
					100, 100, 100, 100, hWnd, (HMENU) 100,
					((LPCREATESTRUCT) lParam)->hInstance, NULL);
			break;
		case WM_LBUTTONDOWN:
			SendMessage(hButton, BM_CLICK, 0, 0);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

/* button2.c*/

```
…..
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hButton;

	switch (message) {
		case WM_CREATE:
			hButton = CreateWindow("button", "Test",
				WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
				100, 100, 100, 100, hWnd, (HMENU) 100,
				((LPCREATESTRUCT) lParam)->hInstance, NULL);
			break;
```

```
                case WM_LBUTTONDOWN:
                        PostMessage((HWND) 0xb0296, BM_CLICK, 0, 0); // Spy++ ile pencerenin handle değeri alınıp
yazıldı.
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

Düğme Kontrolünün Gönderdiği Mesajlar

Düğme kontrolü, üzerine tıklandığında üst pencereye BM_CLICKED işlem koduyla WM_COMMAND mesajını gönderir. Programcı düğmeye tıklandığını tipik olarak şöyle tespit eder:

```
case WM_COMMAND:
        if (LOWORD(wParam) == ID_BUTTON
            && HIWORD(wParam) == BN_CLICKED){
                ....
                ....
                ...
        }
        break;
```

Düğme kontrolü, BM_CLICKED kodlu mesajı, el fareden çekildiğinde göndermektedir.
Düğme kontrolü, BM_CLICKED işlem kodu dışında birkaç mesaj gönderse de bunlar önemli değildir.

Listeleme Kutusu Kontrolü

Listeleme kutusu, bir grup elemanı listelemek için kullanılan genel bir kontroldür. Bu kontrol "listbox" sınıf ismi kullanılarak yaratılır. Listeleme kutusu için pencere başlığı yazısının bir önemi yoktur. Bu yazı, kontrol tarafından kullanılmamaktadır. Listeleme kutusunun özel pencere biçimleri şunlardır:

LBS_SORT: Bu pencere biçimi eklenirse listeleme kutusu elemanları otomatik olarak sıralı eklenir.

LBS_NOTIFY: Listeleme kutusunun seçim sırasında üst pencereye mesaj göndermesi isteniyorsa bu pencere biçiminin eklenmesi gerekir. Default olarak kontrol, üst pencereye bu mesajları göndermemektedir.

LBS_EXTENDEDSEL, LBS_MULTIPLESEL: İlk pencere biçimi, bir elemandan başlayarak ardışıl birden fazla eleman seçebilmek için; ikincisi ise, ardışıl olmayan birden fazla elemandan seçmek için kullanılır.

LBS_MULTICOLUMN: Birden fazla kolonlu listeleme kutusu yaratmak için kullanılır.

LBS_STANDARD: Bu pencere biçimi, LBS_NOTIFY, LBS_SORT ve WS_VSCROLL pencere biçimlerinin or'lanmış halidir.

WS_HSCROLL ve ***WS_VSCROLL*** pencere biçimleri, kaydırma çubuklarının çıkması için kullanılır.
BAK:listbox_.c

```
/* listbox_.c */
```

```
#include <windows.h>
#include <stdio.h>

#define ID_LISTBOX              100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }
        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hListBox;

        switch (message) {
                case WM_CREATE:
                        hListBox = CreateWindow("listbox", "",
                                        WS_CHILD|WS_VISIBLE|LBS_STANDARD,
                                        100, 100, 100, 200, hWnd, (HMENU) ID_LISTBOX,
                                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
```

```
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

Listeleme Kutusuna Gönderilen Mesajlar

Listeleme kutusuna çeşitli faydalı işlemleri yaptırabilmek için ona önceden belirlenmiş mesajları göndeririz.

LB_ADDSTRING Mesajı: Bu mesaj, listeleme kutusuna eleman eklemek için kullanılır. Mesajın wParam parametresi kullanılmaz; sıfırlanmalıdır. Yerleştirilecek yazının başlangıç adresi, lParam parametresine yerleştirilir. Eğer listeleme kutusu, LBS_SORT pencere biçimini bulunduruyorsa elemanlar sırayı bozmayacak biçimde eklenir. Bulundurmuyorsa elemanlar sona eklenir.

Sınıf Çalışması: Bir listeleme kutusu yaratınız, içerisine 20 kadar ilin ismini ekleyiniz.

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void AddCity(HWND hlistBox);

#define  ID_LISTBOX     100

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }
        hWnd = CreateWindow("Generic", "pencere...",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);
        if (!hWnd)
                return -1;
        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
```

```
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HWND hlistBox;

	switch (message) {
		case WM_CREATE:
			hlistBox = CreateWindow("listbox", "",
				WS_CHILD|WS_VISIBLE|LBS_STANDARD|LBS_MULTIPLESEL,
				100, 100, 100, 200, hWnd, (HMENU) ID_LISTBOX,
				((LPCREATESTRUCT) lParam)->hInstance, NULL);
			AddCity(hlistBox);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
			return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}

void AddCity(HWND hlistBox)
{
	char *cities[] = {"Kars", "Balıkesir", NULL};

	int i;
	for (i = 0 ; cities[i] != NULL; ++i)
		SendMessage(hlistBox, LB_ADDSTRING, 0, (LPARAM) cities[i]);
}
```

LB_ADDSTRING mesajı gönderildikten sonra SendMessage, eklenen elemanın index numarası ile geri döner. listeleme kutusunda her elemanın 0'dan başlayan bir index değeri vardır. Ekleme işlemi başarısız olabilir. Bu durumda SendMessage, LB_ERR özel değeri ile geri döner.

LB_GETCURSEL Mesajı: Bu mesajda wParam ve lParam değerleri kullanılmaz; 0'lanmalıdır. SendMessage fonksiyonu, o andaki aktif elemanın index değeri ile geri döner.

```
#include <windows.h>
#include <stdio.h>

#define ID_LISTBOX	100
#define ID_BUTTON		101

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);

void AddCity(HWND hListBox);
void OnButtonClicked(HWND hListBox);

int WINAPI WinMain(…..)
{
	….
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
```

```
{
	static HWND hListBox, hButton;

	switch (message) {
		case WM_CREATE:
		    hListBox = CreateWindow("listbox", "",
			WS_CHILD|WS_VISIBLE|LBS_STANDARD|LBS_EXTENDEDSEL|
			LBS_MULTIPLESEL, 100, 100, 100, 200, hWnd,
			(HMENU) ID_LISTBOX, ((LPCREATESTRUCT) lParam)->hInstance, NULL);
		    hButton = CreateWindow("button", "Ok",
			WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON, 220, 200, 100, 100,
			hWnd, (HMENU) ID_BUTTON,
			((LPCREATESTRUCT) lParam)->hInstance, NULL);
		    AddCity(hListBox);
		    break;
		case WM_COMMAND:
		    if (LOWORD(wParam) == ID_BUTTON && HIWORD(wParam) == BN_CLICKED)
				OnButtonClicked(hListBox);
		    break;
		case WM_DESTROY:
		    PostQuitMessage(0);
		    break;
		default:
		    return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}

void AddCity(HWND hListBox)
{
	char *cities[] = {"Kars", "Adana", "Adıyaman", "Afyon", "Ağrı", "Amasya", "Ankara",
			"Antalya", "Artvin", "Aydın", "Balıkesir", "Bilecik", "Edirne", "Eskişehir",
			"Kırşehir", "Viranşehir", NULL};
	int i;

	for (i = 0; cities[i] != NULL; ++i)
		SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM) cities[i]);
}

void OnButtonClicked(HWND hListBox)
{
	char text[30];
	int index = SendMessage(hListBox, LB_GETCURSEL, 0, 0);

	sprintf(text, "%d", index);

	MessageBox(NULL, text, "Message", MB_OK);

}
```

LB_GETTEXT: Bu mesaj, belirli bir indexteki elemana ilişkin yazıyı elde etmek için kullanılır. Mesajın wParam parametresine index numarası, lParam parametresine ise yazının yerleştirileceği dizinin adresi girilir. SendMessage fonksiyon başarısızsa LB_ERR değerine; başarılıysa yazının karakter uzunluğuna geri döner. O halde aktif elemanın yazısını şöyle alabiliriz:

```
int index;
char text[SIZE];
....
index = SendMessage(hListBox, LB_GETCURSEL, 0, 0);
SendMessage(hListBox, LB_GETTEXT, index, (LPARAM) text);
```

Bak: listbox1.c

/* listbox1.c */

```
#include <windows.h>
#include <stdio.h>

#define ID_LISTBOX              100
#define ID_BUTTON               101

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void AddCity(HWND hListBox);
void OnButtonClicked(HWND hListBox);

int WINAPI WinMain(….)
{
….
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hListBox, hButton;

        switch (message) {
                case WM_CREATE:
                        hListBox = CreateWindow("listbox", "",
                              WS_CHILD|WS_VISIBLE|LBS_STANDARD|LBS_EXTENDEDSEL|
                              LBS_MULTIPLESEL, 100, 100, 100, 200,       hWnd,
                              (HMENU) ID_LISTBOX, ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        hButton = CreateWindow("button", "Ok",
                                WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                                220, 200, 100, 100,          hWnd, (HMENU) ID_BUTTON,
                                ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        AddCity(hListBox);
                        break;
                case WM_COMMAND:
                       if (LOWORD(wParam) == ID_BUTTON && HIWORD(wParam) == BN_CLICKED)
                                OnButtonClicked(hListBox);
                       break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void AddCity(HWND hListBox)
{
        char *cities[] = {"Kars", "Adana", "Adıyaman", "Afyon", "Ağrı", "Amasya", "Ankara", "Antalya",
                "Artvin", "Aydın", "Balıkesir", "Bilecik", "Edirne", "Eskişehir", "Kırşehir", "Viranşehir", NULL
        };
        int i;

        for (i = 0; cities[i] != NULL; ++i)
                SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM) cities[i]);
}

void OnButtonClicked(HWND hListBox)
```

```
{
        char text[30];
        int index = SendMessage(hListBox, LB_GETCURSEL, 0, 0);

        SendMessage(hListBox, LB_GETTEXT, index, (LPARAM) text);

        MessageBox(NULL, text, "Message", MB_OK);

}
```

LB_GETSELITEMS: Eğer listeleme kutusu birden fazla elemanın seçilebilmesine olanak veriyorsa seçilen tüm elemanların indexleri bu mesaj ile elde edilir. Mesajın wParam parametresi indexlerin yerleştirileceği int türden dizinin uzunluğunu alır. Yani en fazla bu kadar eleman diziye yerleştirilir. int dizinin başlangıç adresi de mesajın lParam parametresine girilir. SendMessage fonksiyonunun geri dönüş değeri diziye yerleştirilen eleman sayısıdır. Örneğin:

```
int indexes[SIZE];
int count;                                          (wParam)
...                                                 ↗
count = SendMessage(hListBox, LB_GETSELITES, SIZE, (LPARAM) indexes);
```

cout içinde diziye yerleştirilen eleman sayısı vardır.

LB_GETTEXTLEN: Bu mesaj listeleme kutusunun belirli bir indexteki eleman yazısının karakter uzunluğunu bulmak için kullanılır. Mesajın wParam parametresine elemanın index numarası yerleştirilir. lParam kullanılmaz; 0'lanmalıdır. Fonksiyon, o indexteki yazının karakter uzunluğuna geri döner.

LB_GETCOUNT: Bu mesaj, listeleme kutusundaki toplam eleman sayısını bulmakta kullanılır. Mesajın wParam ve lParam parametreleri kullanılmaz, 0'lanmalıdır. Fonksiyon, listeleme kutusundaki eleman sayısına geri döner.

LB_DELETESTRING: Bu mesaj, belirli bir indexteki elemanı silmek için kullanılır. Mesajın wParam parametresi, elemanın index numarasını alır. lParam kullanılmaz, 0'lanmalıdır. Fonksiyonun (SendMessage) geri dönüş değeri, listede kalan eleman sayısını belirtir. Hata durumunda fonksiyon, LB_ERR değerine geri döner. Bak:listbox3.c

/* listbox3.c */

```
#include <windows.h>
#include <stdio.h>

#define ID_LISTBOX              100
#define ID_BUTTON               101

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void AddCity(HWND hListBox);
void OnButtonClicked(HWND hListBox);

int WINAPI WinMain(…)
{
….
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hListBox, hButton;
```

```
	switch (message) {
		case WM_CREATE:
			hListBox = CreateWindow("listbox", "", WS_CHILD|WS_VISIBLE|
				LBS_STANDARD|LBS_EXTENDEDSEL|LBS_MULTIPLESEL,
				100, 100, 100, 200, hWnd, (HMENU) ID_LISTBOX,
				((LPCREATESTRUCT) lParam)->hInstance, NULL);
			hButton = CreateWindow("button", "Ok",
				WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
				220, 200, 100, 100,		hWnd, (HMENU) ID_BUTTON,
				((LPCREATESTRUCT) lParam)->hInstance, NULL);
			AddCity(hListBox);
			break;
		case WM_COMMAND:
			if (LOWORD(wParam) == ID_BUTTON && HIWORD(wParam) == BN_CLICKED)
				OnButtonClicked(hListBox);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}

void AddCity(HWND hListBox)
{
	char *cities[] = {"Kars", "Adana", "Adıyaman", "Afyon", "Ağrı", "Amasya", "Ankara", "Antalya",
		"Artvin", "Aydın", "Balıkesir", "Bilecik", "Edirne", "Eskişehir", "Kırşehir", "Viranşehir", NULL};
	int i;

	for (i = 0; cities[i] != NULL; ++i)
		SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM) cities[i]);
}

void OnButtonClicked(HWND hListBox)
{
	int indexes[30], count, i;

	count = SendMessage(hListBox, LB_GETSELITEMS, 30, (LPARAM) indexes);

	for (i = 0; i < count; ++i) {
		SendMessage(hListBox, LB_DELETESTRING, indexes[i] - i, 0);
	}

}
```

Sınıf Çalışması: 2 listeleme kutusu, 2 de düğme kontrolü yaratınız. 2 listeleme kutusuna da birkaç eleman giriniz. Düğmelerin birine basıldığında bir kutuda seçilen eleman diğer kutuya, diğerine basıldığında diğer kutuda seçilen eleman öbür kutuya aktarılacaktır. BAK: listbox4.c LB_ERR gelince kontrol yap ve seçim olmadığını belirt.

/* listbox4.c */

```
#include <windows.h>
#include <stdio.h>

#define ID_LISTBOX1		100
#define ID_BUTTON1		101
#define ID_LISTBOX2		102
```

```
#define ID_BUTTON2		103

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);

void AddCity(HWND hListBox);
void OnButtonClicked(HWND hListBox1, HWND hListBox2);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}
	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hListBox1, hListBox2, hButton1, hButton2;

	switch (message) {
		case WM_CREATE:
			hListBox1 = CreateWindow("listbox", "",
WS_CHILD|WS_VISIBLE|LBS_STANDARD|LBS_EXTENDEDSEL|LBS_MULTIPLESEL,
				100, 100, 100, 200, hWnd, (HMENU) ID_LISTBOX1,
((LPCREATESTRUCT) lParam)->hInstance, NULL);
			hButton1 = CreateWindow("button", ">>>", WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
```

```
                                                220, 100, 50, 25, hWnd,          (HMENU)          ID_BUTTON1,
((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        hListBox2                =                CreateWindow("listbox",                  "",
WS_CHILD|WS_VISIBLE|LBS_STANDARD|LBS_EXTENDEDSEL|LBS_MULTIPLESEL,
                                                300, 100, 100, 200,      hWnd,      (HMENU)      ID_LISTBOX2,
((LPCREATESTRUCT) lParam)->hInstance, NULL);
                                hButton2 = CreateWindow("button", "<<<", WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                                                220, 275, 50, 25, hWnd,          (HMENU)          ID_BUTTON2,
((LPCREATESTRUCT) lParam)->hInstance, NULL);
                                        AddCity(hListBox1);
                                break;
                        case WM_COMMAND:
                                if (LOWORD(wParam) == ID_BUTTON1 && HIWORD(wParam) == BN_CLICKED)
                                        OnButtonClicked(hListBox1, hListBox2);
                                else if ((LOWORD(wParam) == ID_BUTTON2 && HIWORD(wParam) == BN_CLICKED))
                                        OnButtonClicked(hListBox2, hListBox1);

                                break;
                        case WM_DESTROY:
                                PostQuitMessage(0);
                                break;
                        default:
                          return DefWindowProc(hWnd, message, wParam, lParam);
                }
                return 0;
}

void AddCity(HWND hListBox)
{
        char *cities[] = {"Kars", "Adana", "Adıyaman", "Afyon", "Ağrı", "Amasya", "Ankara", "Antalya",
              "Artvin", "Aydın", "Balıkesir", "Bilecik", "Edirne", "Eskişehir", "Kırşehir", "Viranşehir", NULL};
        int i;

        for (i = 0; cities[i] != NULL; ++i)
                SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM) cities[i]);
}

void OnButtonClicked(HWND hListBox1, HWND hListBox2)
{
        int index;
        char text[30];

        index = SendMessage(hListBox1, LB_GETCURSEL, 0, 0);
        if (SendMessage(hListBox1,LB_GETCOUNT, 0,0) == 0){
                MessageBox(NULL, "eleman bulunamadı", "Message", MB_OK);
                return;
        }

        SendMessage(hListBox1, LB_GETTEXT, index, (LPARAM)text);
        SendMessage(hListBox1, LB_DELETESTRING, index, 0);
        SendMessage(hListBox2, LB_ADDSTRING, 0, (LPARAM)text);

}

veya…
void OnButtonClicked(HWND hListBox1, HWND hListBox2)
{
        int index;
        char text[30];

        if ((index = SendMessage(hListBox1, LB_GETCURSEL, 0, 0)) == LB_ERR) {
                MessageBox(NULL, "Lütfen bir eleman seçiniz!", "Hata", MB_OK);
                return;
```

```
        }

        SendMessage(hListBox1, LB_GETTEXT, index, (LPARAM)text);
        SendMessage(hListBox1, LB_DELETESTRING, index, 0);
        SendMessage(hListBox2, LB_ADDSTRING, 0, (LPARAM)text);
} // listbox5.c'de
```

LB_DIR: Bu mesaj, bir yol ifadesi ile belirtilen dosyaları listeleme kutusuna yerleştirir. Mesajın lParam parametresi yol ifadesine; wParam parametresi belirtilen yol ifadesindeki hangi özelliğe sahip dosyaların ekleneceğini belirtir. wParam, aşağıdaki sembolik sabitlerin bit veya işlemine sokulması ile elde edilir:

Value	Description
DDL_ARCHIVE	Includes archived files.
DDL_DIRECTORY	Includes subdirectories. Subdirectory names are enclosed in square brackets ([ ]).
DDL_DRIVES	Includes drives. Drives are listed in the form [-*x*-], where *x* is the drive letter.
DDL_EXCLUSIVE	Includes only files with the specified attributes. By default, read-write files are listed even if DDL_READWRITE is not specified.
DDL_HIDDEN	Includes hidden files.
DDL_READONLY	Includes read-only files.
DDL_READWRITE	Includes read-write files with no additional attributes.
DDL_SYSTEM	Includes system files.

Bak: listboxdir.c, listboxdir1.c -> sort olayı devre dışı bırakılmış.

```
/* listboxdir.c */
#include <windows.h>

#define ID_LISTBOX              100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(…)
{
        …..
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hListBox;

        switch (message) {
                case WM_CREATE:
                        hListBox = CreateWindow("listbox", "",
                                WS_CHILD|WS_VISIBLE|LBS_STANDARD, 100, 100, 200, 200,
                                hWnd, (HMENU) ID_LISTBOX,
                                ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        SendMessage(hListBox, LB_DIR, DDL_DIRECTORY,
                                        (LPARAM) "c:\\windows\\*.*");
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
```

```
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
/* listboxdir1.c * /
hListBox = CreateWindow("listbox", "", WS_CHILD|WS_VISIBLE|LBS_STANDARD & ~LBS_SORT,
                        100, 100, 200, 200, hWnd, (HMENU) ID_LISTBOX,
                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);

// sort devre dışı bırakılmış.
```

Dizinler, köşeli parantez içerisinde belirtilmektedir.

LB_RESETCONTENT: Bu mesaj, listeleme kutusunun tüm elemanlarını silmek için kullanılır. Mesajın wParam ve lParam parametreleri kullanılmaz; 0'lanmalıdır.

LİSTELEME KUTUSUNUN ÜST PENCEREYE GÖNDERDİĞİ MESAJLAR:

Listeleme kutusu, bazı işlemler gerçekleştiğinde üst pencereye WM_COMMAND mesajı göndererek bu işlemden üst pencereyi haberdar ederler. HIWORD(wParam)'da kontrolün ne sebeple mesaj gönderdiği bilgisi yer alır. Anımsanacağı gibi kontrolün mesaj göndermesini sağlamak için pencere yaratılırken LBS_NOTIFY pencere biçiminin kullanılması gerekir.

Listeleme kutusundaki en önemli hareket, bir elemana çift tıklama işlemidir. Bu durumda kontrol, üst pencereye **LBN_DBLCLK** işlem kodlu mesajı gönderir. Bu mesajı yakalayan programcı, hangi elemana çift tıklanmış olduğunu bilmez. Önce LB_GETCURSEL mesajı ile aktif elemanı elde etmelidir. BAK: listboxdir2.c

```
/* listboxdir2.c */
#include <windows.h>
#include <io.h>
#include <stdio.h>

#define ID_LISTBOX              100
#define PATH                    "*.*"

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnListBoxDblClk(HWND hListBox);

int WINAPI WinMain(…)
{
        ….
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hListBox;

        switch (message) {
                case WM_CREATE:
                        hListBox = CreateWindow("listbox", "",
```

```
                                WS_CHILD|WS_VISIBLE|LBS_STANDARD & ~LBS_SORT,
                                100, 100, 200, 200, hWnd, (HMENU) ID_LISTBOX,
                                ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        SendMessage(hListBox, LB_DIR, DDL_ARCHIVE, (LPARAM) PATH);
                        break;
                case WM_COMMAND:
                        if (LOWORD(wParam) == ID_LISTBOX && HIWORD(wParam) == LBN_DBLCLK)
                                OnListBoxDblClk(hListBox);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnListBoxDblClk(HWND hListBox)
{
        int index = SendMessage(hListBox, LB_GETCURSEL, 0, 0);
        char fileName[256];
        struct _finddata_t fd;
        char fileInfo[1000];


        SendMessage(hListBox, LB_GETTEXT, index, (LPARAM) fileName);
        if (!_findfirst(fileName, &fd)) {
                MessageBox(NULL, "Cannot get file info!..", "Error", MB_OK);
                return;
        }

        sprintf(fileInfo, "File Name: %s\nFile Size: %ld", fd.name, fd.size);
        MessageBox(NULL, fileInfo, "File Info", MB_OK);
}
```

Listeleme kutusunun her aktif elemanı değiştiğinde listeleme kutusu üst pencereye LBN_SELCHANGE işlem koduyla WM_COMMAND mesajını gönderir. Bu mesaj sayesinde aktif eleman değiştikçe yeni aktif hale getirilen eleman hakkında bilgiler bir biçimde görüntülenebilir. Bu mesajı işlerken de LB_GETCURSEL mesajıyla önce aktif elemanın elde edilmesi gerekir. BAK: listboxdir3.c

/* listboxdir3.c */

```
#include <windows.h>
#include <io.h>
#include <stdio.h>

#define ID_LISTBOX              100
#define ID_STATIC               101
#define PATH                    "*.*"

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnListBoxDblClk(HWND hListBox);
void OnListBoxSelChange(HWND hListBox, HWND hStatic);

int WINAPI WinMain(…)
{
        …
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
	static HWND hListBox, hStatic;

	switch (message) {
		case WM_CREATE:
			hListBox = CreateWindow("listbox", "",
				WS_CHILD|WS_VISIBLE|LBS_STANDARD & ~LBS_SORT,
				100, 100, 200, 200, hWnd, (HMENU) ID_LISTBOX,
				((LPCREATESTRUCT) lParam)->hInstance, NULL);

			SendMessage(hListBox, LB_DIR, DDL_ARCHIVE, (LPARAM) PATH);

			hStatic = CreateWindow("static", "", WS_CHILD|WS_VISIBLE,
					350, 100, 300, 100, hWnd, (HMENU) ID_STATIC,
					((LPCREATESTRUCT) lParam)->hInstance, NULL);
			break;
		case WM_COMMAND:
			if (LOWORD(wParam) == ID_LISTBOX)
				switch (HIWORD(wParam))  {
					case LBN_DBLCLK:
						OnListBoxDblClk(hListBox);
						break;
					case LBN_SELCHANGE:
						OnListBoxSelChange(hListBox, hStatic);
						break;
				}
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}

void OnListBoxDblClk(HWND hListBox)
{
	int index = SendMessage(hListBox, LB_GETCURSEL, 0, 0);
	char fileName[256];
	struct _finddata_t fd;
	char fileInfo[1000];
	long handle;


	SendMessage(hListBox, LB_GETTEXT, index, (LPARAM) fileName);
	if (!(handle = _findfirst(fileName, &fd))) {
		MessageBox(NULL, "Cannot get file info!..", "Error", MB_OK);
		return;
	}

	sprintf(fileInfo, "File Name: %s\nFile Size: %ld", fd.name, fd.size);
	MessageBox(NULL, fileInfo, "File Info", MB_OK);

	_findclose(handle);
}

void OnListBoxSelChange(HWND hListBox, HWND hStatic)
{
	int index = SendMessage(hListBox, LB_GETCURSEL, 0, 0);
	char fileName[256];
	struct _finddata_t fd;
	char fileInfo[1000];
	long handle;
```

```
        SendMessage(hListBox, LB_GETTEXT, index, (LPARAM) fileName);
        if (!(handle = _findfirst(fileName, &fd))) {
                MessageBox(NULL, "Cannot get file info!..", "Error", MB_OK);
                return;
        }

        sprintf(fileInfo, "File Name: %s\nFile Size: %ld", fd.name, fd.size);

        SetWindowText(hStatic, fileInfo);

        _findclose(handle);
}
```

EDIT Kontrolü

EDIT kontrolü, temel kontrollerden biridir. Kullanıcıdan string almak için ve birtakım bilgileri görüntülemek için kullanılır. EDIT kontrolü, "edit" sınıf ismi kullanılarak yaratılır. EDIT kontrolünün özel pencere biçimleri şunlardır:

ES_CENTER, ES_LEFT, ES_RIGHT: Bu pencere biçimleri, girilen yazının hizalanması için kullanılır.

ES_LOWERCASE, ES_UPPERCASE: Bu pencere biçimleri ne olursa olsun girilen karakterlerin hep küçük harf ve hep büyük harf çıkmalarını sağlar.
BAK: editbox.c

```
#include <windows.h>
#include <stdio.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(…)
{
        …
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hStatic;

        switch (message) {
                case WM_CREATE:
                        hStatic = CreateWindow("static", "",
                        WS_CHILD|WS_VISIBLE|WS_VSCROLL|SS_LEFT|SS_LEFTNOWORDWRAP ,
                        0, 0, 0, 0,hWnd, (HMENU) 100,
                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        break;
                case WM_LBUTTONDOWN:
                        {
                                FILE *f;
                                char *pText;
                                long size;

                                if ((f = fopen("editbox.c", "r")) == NULL) {
                                        MessageBox(hWnd, "Cannot open file!..", "Error", MB_OK);
                                        break;
                                }
                                fseek(f, 0, SEEK_END);
                                size = ftell(f);
```

```
                        fseek(f, 0, SEEK_SET);

                        pText = (char *) malloc(size + 1);
                        fread(pText, 1, size, f);
                        pText[size] = '\0';
                        SetWindowText(hStatic, pText);
                        free(pText);
                        fclose(f);
                    }
                    break;
                case WM_SIZE:
                    MoveWindow(hStatic, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE);
                    break;
                case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

ES_MULTILINE: Default olarak EDIT kontrolü tek satırdır fakat bu pencere biçimiyle kontrol, çok satırlı biçime getirilebilir. BAK: editbox1.c

ES_PASSWORD: Parola girmek için kullanılır. Basılan karakterler yerine * gibi bir karakter görüntülenir. BAK: editbox2.c

ES_READONLY: Bu pencere biçimi, kullanıcının doğrudan yazı girmesini engeller. Tabi yazı, programlama yoluyla kontrole yerleştirilebilir.

ES_AUTOHSCROLL, ES_AUTOVSCROLL: Normal olarak çok satırlı edit kontrolünde, o andaki pencere büyüklüğünden daha fazla satır giremeyiz. Fakat ES_AUTOVSCROLL pencere biçimiyle pencere yukarı doğru scroll edilerek istenildiği kadar satır girilebilir. Fakat ES_AUTOVSCROLL tek başına düşey kaydırma çubuklarının çıkmasını sağlamaz. Bunun için WS_VSCROLL pencere biçiminin de eklenmesi gerekir. Aslında kolaylık olsun diye edit kontrolünü yaratırken WS_VSCROLL kullanıldığında ES_AUTOVSCROLL biçiminin de kullanıldığı varsayılır. ES_AUTOVSCROLL pencere biçiminin tek satırlı edit kontrolünde bir kullanımı yoktur. Bak: editbox3-4.c

ES_AUTOHSCROLL, tek satırlı edit kontrolünde sola doğru scroll işlemi yapar. Yani biz yazıyı kaydırarak pencere uzunluğundan daha fazla yazı girebiliriz. Çok satırlı edit kontrolünde ise bu pencere biçimi belirtilmediğinde satır sonuna gelindiğinde sarma yapar. Fakat bu pencere biçimi belirtilmişse her satır bağımsız olarak sola doğru scroll eder. Yine bu pencere biçimi kendi başına yatay kaydırma çubuklarının çıkmasını sağlamaz. Bunun için ayrıca WS_HSCROLL kullanmak gerekir. Bu pencere biçimi kullanıldığında ise ES_AUTOHSCROLL kullanmamıza gerek kalmaz.

EDIT Kontrolünde Pencere Başlık Yazısı

Edit kontrolünde pencere başlık yazısı, edit kontrolü içerisindeki yazıdır. Biz SetWindowText fonksiyonu ile bir yazı set etmek istersek bu durumda kontroldeki eski yazı tamamen silinir; yerine yeni yazı gelir. Benzer biçimde GetWindowText ile biz tek hamlede kontrol içerisindeki tüm yazıyı alabiliriz.

edit kontrolünde aşağı satırın başına geçme işlemi, yalnızca '\n' ile değil; '\r\n' (carriage return, line feed) ile yapılmaktadır. Örneğin statik kontrolde ve MessageBox fonksiyonunda bu işlem sadece '\n' ile yapılmaktadır. Bak: editbox5.c

```
/* editbox5.c */

#include <windows.h>
#include <io.h>
#include <stdio.h>

#define ID_LISTBOX              100
#define ID_EDIT                 101
#define PATH                    "*.*"

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnListBoxDblClk(HWND hListBox);
void OnListBoxSelChange(HWND hListBox, HWND hEdit);

int WINAPI WinMain(…)
{
        …
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hListBox, hEdit;

        switch (message) {
                case WM_CREATE:
                        hListBox = CreateWindow("listbox", "",
                                WS_CHILD|WS_VISIBLE|LBS_STANDARD & ~LBS_SORT,
                                25, 100, 200, 200, hWnd, (HMENU) ID_LISTBOX,
                                ((LPCREATESTRUCT) lParam)->hInstance, NULL);

                        SendMessage(hListBox, LB_DIR, DDL_ARCHIVE, (LPARAM) PATH);

                        hEdit = CreateWindow("edit", "", WS_CHILD|WS_VISIBLE|WS_BORDER|
                                WS_VSCROLL|WS_HSCROLL|ES_MULTILINE, 275, 100, 300, 200, hWnd,
                                (HMENU) ID_EDIT, ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        break;
                case WM_COMMAND:
                        if (LOWORD(wParam) == ID_LISTBOX)
                                switch (HIWORD(wParam))  {
                                        case LBN_DBLCLK:
                                                OnListBoxDblClk(hListBox);
                                                break;
                                        case LBN_SELCHANGE:
                                                OnListBoxSelChange(hListBox, hEdit);
                                                break;
                                }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnListBoxDblClk(HWND hListBox)
{
        int index = SendMessage(hListBox, LB_GETCURSEL, 0, 0);
        char fileName[256];
        struct _finddata_t fd;
```

```
	char fileInfo[1000];
	long handle;


	SendMessage(hListBox, LB_GETTEXT, index, (LPARAM) fileName);
	if (!(handle = _findfirst(fileName, &fd))) {
		MessageBox(NULL, "Cannot get file info!..", "Error", MB_OK);
		return;
	}

	sprintf(fileInfo, "File Name: %s\nFile Size: %ld", fd.name, fd.size);
	MessageBox(NULL, fileInfo, "File Info", MB_OK);

	_findclose(handle);
}

void OnListBoxSelChange(HWND hListBox, HWND hEdit)
{
	int index = SendMessage(hListBox, LB_GETCURSEL, 0, 0);
	char fileName[256], *pStr;
	FILE *f;
	long flen;

	SendMessage(hListBox, LB_GETTEXT, index, (LPARAM) fileName);

	if ((f = fopen(fileName, "rb")) == NULL) {
		MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
		return;
	}
	fseek(f, 0, SEEK_END);
	flen = ftell(f);

	pStr = (char *) malloc(flen + 1);
	if (pStr == NULL) {
		MessageBox(NULL, "Cannot allocate memory!..", "Error", MB_OK);
		fclose(f);
		return;
	}

	rewind(f);

	fread(pStr, 1, flen, f);
	pStr[flen] = '\0';

	SetWindowText(hEdit, pStr);

	free(pStr);
	fclose(f);
}
```

EDIT Kontrolüne Gönderilen Mesajlar

edit kontrolüne mesajlar gönderilerek çeşitli faydalı işlemler yapılabilir.

EM_LIMITTEXT: Bu mesaj, edit kontroldeki toplam karakter sayısını sınırlandırmaktadır. Mesajın wParam parametresi, maksimum karakter sayısını belirtir. lParam parametresi kullanılmaz; 0'lanmalıdır. Bak: editbox6.c

```
#include <windows.h>
#include <stdio.h>
```

```
#define ID_EDIT		100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(…)
{
	….
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hEdit;

	switch (message) {
		case WM_CREATE:
			hEdit = CreateWindow("edit", "",
				WS_CHILD|WS_VISIBLE|WS_BORDER|ES_AUTOHSCROLL,
				100, 100, 200, 100, hWnd, (HMENU) ID_EDIT,
				((LPCREATESTRUCT) lParam)->hInstance, NULL);
			SendMessage(hEdit, EM_LIMITTEXT, 10, 0);
			break;

		/*case WM_SIZE:
			MoveWindow(hEdit, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE);
			break;
		*/
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

EM_GETLINECOUNT: Bu mesaj, çok satırlı edit kontrolünde kontroldeki toplam satır sayısını elde etmekte kullanılır. Mesajın wParam ve lParam parametreleri kullanılmaz; 0'lanmalıdır. SendMessage fonksiyonunun geri dönüş değeri, toplam satır sayısıdır.

EM_GETLINE: Bu mesaj, edit kontrolündeki belli bir satırda bulunan yazıları elde etmek için kullanılır. Mesajın wParam parametresine satır numarası girilir; lParam parametresine ise yazının yerleştirileceği dizinin başlangıç adresi girilir. Bu mesaj, yazının sonuna NULL karakterini kendisi yerleştirmemektedir. SendMessage fonksiyonunun geri dönüş değeri, yerleştirilen karakter sayısıdır. NULL karakter bu değer kullanılarak yerleştirilebilir. Yazının yerleştirileceği dizinin ilk WORD elemanı maksimum yerleştirilecek karakter sayısını içermelidir. Tek satırlı edit kontrolünde wParam parametresi zaten kullanılmamaktadır. Bu durumda n. satırdaki yazı şöyle alınabilir:

```
int len;
char buf[SIZE];

* (WORD *) buf = SIZE – 1;
...
len = SendMessage(hEdit, EM_GETLINE, n, (LPARAM) buf);
buf[len] = '\0';
```

Bak: sample edit.c

```
/* Sample.c */
```

```
#include <windows.h>
#include <stdio.h>

#define ID_EDIT                 100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(…)
{
        …
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hEdit;

        switch (message) {
                case WM_CREATE:
                        hEdit= CreateWindow("edit", "",
                             WS_CHILD|WS_VISIBLE|WS_VSCROLL|WS_BORDER|ES_MULTILINE ,
                             100, 100, 300, 300, hWnd, (HMENU) ID_EDIT,
                             ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        break;
                case WM_LBUTTONDOWN:
                        {
                                char buf[100];
                                int n;

                                *(WORD *) buf = 99;
                                n = SendMessage(hEdit, EM_GETLINE, 3, (LPARAM) buf);
                                buf[n] = '\0';
                                MessageBox(NULL, buf, "Message", MB_OK);
                        }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

EM_LINEINDEX: Edit kontrolündeki her karakterin ilk karakter 0 olmak üzere bir index numarası vardır. Bazı mesajlarda bu index numaraları kullanılmaktadır (enter tuşu için cr/lf biçiminde iki karakterin kullanıldığı unutulmamalıdır). İşte EM_LINEINDEX mesajı, bir satırın ilk karakterinin index numarasını bulmaktadır. Mesajın wParam parametresi satır numarasını alır; lParam kullanılmaz, sıfırlanmalıdır. SendMessage fonksiyonunun geri dönüş değeri index numarasıdır. wParam, -1 olarak geçilirse o anda cursor'ın bulunduğu satırın ilk karakterinin index numarası elde edilir.

EM_SETSEL: Bu mesaj, belirli bir metni programlama yoluyla seçmek için kullanılmaktadır. Mesajın wParam parametresi, seçilecek metnin başlangıç index numarası, lParam parametresi bitiş index numarasıdır. Eğer başlangıç index numarası 0, bitiş index numarası –1 olarak girilirse tüm metin seçilir. Eğer başlangıç index numarası –1 olarak girilirse bitiş index numarası ne olursa olsun aktif seçim kaldırılır.

EM_REPLACESEL: Bu mesaj, belirli bir yazıyı o anda seçilmiş olan yazıyı silerek onun yerine insert eder. Eğer herhangi bir yazı seçili değilse silmeden insert işlemi yapılır. Mesajın wParam parametresi

yapılan bu işlemin undo kuyruğuna eklenip eklenmeyeceğini belirtir. TRUE yada FALSE biçiminde geçilebilir. lParam parametresi yerleştirilecek yazının adresini almaktadır.

En sık rastlanan uygulamalardan biri, bir yazının sonuna scroll işlemi yaparak yeni bir yazının programlama yoluyla eklenmesidir. Örneğin klasik sohbet programlarında böyle bir durum söz konusudur. Bu işlem, önce GetWindowText ile tüm yazı alınıp ekleme yapıldıktan sonra SetWindowText ile gerçekleştirilemez. Çünkü bu durumda SetWindowText işleminden sonra kaydırma çubukları yazının başında görüntülenir. Bu işlem şöyle yapılmalıdır.

1. Tüm yazı GetWindowText ile alınır ve yazı strcat ile sona eklenir.
2. EM_SETSEL mesajı ile wParam = 0, lParam = -1 yapılarak tüm metin seçilir.
3. EM_REPLACESEL ile eklenmiş olan yazı, seçilmiş yazı ile değiştirilir. Bak: chat.c

/* chat.c */

```
#include <windows.h>
#include <stdio.h>

#define ID_EDIT_CHAT            100
#define ID_EDIT_MESSAGE         101
#define ID_BUTTON               102

#define MSG_SIZE                100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);

void OnButtonClick(HWND hEditChat, HWND hEditMessage);

int WINAPI WinMain(…)
{
        …
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hEditChat, hEditMessage, hButton;

        switch (message) {
                case WM_CREATE:
                    hEditChat = CreateWindow("edit", "",
                        WS_CHILD|WS_VISIBLE|WS_VSCROLL|WS_BORDER|ES_MULTILINE ,
                        100, 50, 300, 300, hWnd, (HMENU) ID_EDIT_CHAT,
                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                    hEditMessage = CreateWindow("edit", "", WS_CHILD|WS_VISIBLE|WS_BORDER ,
                        100, 400, 300, 20, hWnd, (HMENU) ID_EDIT_MESSAGE,
                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                    hButton = CreateWindow("button", "Ok",
                        WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON, 200, 450, 100, 30,
                        hWnd, (HMENU) ID_BUTTON,
                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);

                    break;
                case WM_COMMAND:
                    if (LOWORD(wParam) == ID_BUTTON && HIWORD(wParam) == BN_CLICKED)
                            OnButtonClick(hEditChat, hEditMessage);
                    break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
```

```
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnButtonClick(HWND hEditChat, HWND hEditMessage)
{
        char message[MSG_SIZE], *pText;
        int textSize;

        GetWindowText(hEditMessage, message, MSG_SIZE);

        textSize = GetWindowTextLength(hEditChat) + strlen(message) + 2;  /* 2 for CR/LF */
        pText = (char *) malloc(textSize + 1);
        if (pText == NULL) {
                MessageBox(NULL, "Cannot allocate memory!..", "Error", MB_OK);
                return;
        }

        GetWindowText(hEditChat, pText, textSize);
        strcat(pText, "\r\n");
        strcat(pText, message);

        SendMessage(hEditChat, EM_SETSEL, 0, -1);
        SendMessage(hEditChat, EM_REPLACESEL, TRUE, (LPARAM) pText);

        SetWindowText(hEditMessage, "");
        SetFocus(hEditMessage);

        free(pText);
}
```

EM_UNDO: Bu mesaj programlama yoluyla undo işlemi yapmak için kullanılmaktadır. wParam ve lParam parametreleri kullanılmaz; sıfırlanmalıdır.

WM_CUT, WM_COPY, WM_PASTE: Bu mesajlar edit penceresine gönderilirse seçili alan üzerinde clipboard işlemleri yapılmış olur. Yani örneğin biz EM_SETSEL ile metnin bir kısmını seçip pencereye WM_COPY mesajını gönderirsek sanki ctrl + c tuşuna basmış gibi seçilenler clipboard'a taşınır.

EM_GETSEL: Bu mesaj bir seçilmiş alanın başlangıç ve bitiş index numarasını almakta kullanılır. Mesajın wParam ve lParam parametrelerine 2 DWORD türünden bir nesnenin adresi girilir. Mesaj, seçili alanın index numaralarını bu nesnelere yerleştirir.

EDIT KONTROLÜNÜN ÜST PENCEREYE GÖNDERDİĞİ MESAJLAR

EDIT kontrolü, diğer kontrollerde olduğu gibi üst pencereye WM_COMMAND mesajını göndermektedir. edit kontrolünün gönderdiği mesajlardan en önemlileri **EN_UPDATE** ve **EN_CHANGE** adlı mesajlardır. EN_UPDATE, tuşa basıldıktan sonra fakat tuş görüntüsü henüz edit kontrolünde çıkmadan önce gönderilir. EN_CHANGE ise EN_UPDATE mesajından sonra basılan tuş edit kontrolünde görüldükten sonra gönderilir. Bu mesajların sanıldığından daha dar bir kullanımı vardır. Bu mesajlar ilgili tuşu bize vermezler. Örneğin biz bu mesajlarla yasaklı bir karakteri bastıramamazlık yapamayız. edit kontrolünün bu tür amaçlarla değiştirilmesi işlemi, pencere fonksiyonunun kancalanmasıyla (Windows subclassing) etkin bir biçimde yapılabilmektedir. Bul: projektör: D:\Winsys\systemhook

SEÇENEK KUTULARI (Check Box)

Seçenek kutuları küçük bir dikdörtgen kutu ve yazıdan oluşan basit bir kontroldür. Kullanıcı fare ile kutunun üzerine tıklayarak kutuyu çarpılar yada kutunun üzerindeki çarpıyı kaldırır. Bu kontrol genellikle kullanıcıdan bilgi istendiği durumlarda bir özelliğin bütüne dahil edilip edilmeyeceğini belirlemek amacıyla kullanılır. Form doldurulduktan sonra programcı kutunun çarpılanıp çarpılanmadığını tespit eder. Böylece formu dolduran kullanıcının o özelliği isteyip istemediğini anlar.

Seçenek kutuları manuel ve otomatik olmak üzere 2'ye ayrılmaktadır. Manuel seçenek kutularında kutuya tıklandığında çarpılama işlemi seçenek kutusunun kendisi tarafından otomatik olarak yapılmaz. Seçenek kutusu yalnızca üst pencereye mesaj gönderir; çarpılama işlemini programcı yapar. Halbuki otomatik seçenek kutularında kutuya tıklandığında çarpılama işlemi kontrolün kendisi tarafından otomatik yapılmaktadır.

Seçenek kutuları da "button" sınıf ismiyle yaratılır. Fakat yaratılırken pencere biçimi olarak **BS_CHECKBOX** yada **BS_AUTOCHECKBOX** belirlemesinin yapılması gerekmektedir. Kontrol yaratılırken CreateWindow fonksiyonunun pencere başlığı yazısı, kutucuğun yanında görüntülenecek yazıyı belirtir.

SEÇENEK KUTULARINA GÖNDERILEN MESAJLAR

Seçenek kutularına BM_GETCHECK mesajı gönderilerek seçenek kutusunun çarpılanmış olup olmadığı anlaşılır. Meajın wParam ve lParam parametreleri kullanılmaz sıfırlanmalıdır. SendMessage fonksiyonunun geri dönüş değeri sıfır yada sıfır dışı bir değerdir. Sıfır ise çarpılanmamış, değilse çarpılanmış demektir.

```
case WM_CREATE:
                hCheckBox = CreateWindow("button", "Test",
                        WS_CHILD|WS_VISIBLE|WS_BORDER|BS_AUTOCHECKBOX,
                        100, 100, 50, 20, hWnd, (HMENU) ID_CHECKBOX,
                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                break;
        case WM_LBUTTONDOWN:
                MessageBox(hWnd, SendMessage(hCheckBox, BM_GETCHECK, 0, 0) ?
                                "Checked" : "Unchecked", "Test", MB_OK);
                break;
```

Seçenek kutusu BM_SETCHECK mesajı ile programlama yoluyla çarpılanır yada çarpısı kaldırılabilir. Mesajın wParam parametresi TRUE yada FALSE geçilir. LParam kulanılmaz sıfırlanmalıdır.

SendMessage(hCheckBox,BM_SETCHECK, TRUE, 0); // Seçenek kutusu seçili olarak açılır.

Seçenek kutularıda tıpkı düğmeler gibi üst pencereye BN_CLICKED işlem kodlu mesaj göndermektedir. Bu mesaj özellikle manual seçenek kutularında işlenmektedir.

```
case WM_CREATE:
                hCheckBox = CreateWindow("button", "Test",
                        WS_CHILD|WS_VISIBLE|WS_BORDER|BS_CHECKBOX,
                        100, 100, 50, 20, hWnd, (HMENU) ID_CHECKBOX,
                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                SendMessage(hCheckBox, BM_SETCHECK, TRUE, 0);
                break;
        case WM_COMMAND:
                if (LOWORD(wParam) == ID_CHECKBOX &&
                    HIWORD(wParam) == BN_CLICKED) {
                        int stat = SendMessage(hCheckBox, BM_GETCHECK, 0, 0);
                        SendMessage(hCheckBox, BM_SETCHECK, !stat, 0);
                }
```

```
                break;
```

RADYO DÜĞMELERİ

Radyo düğmeleri birden çok seçeneğin sözkonusu olduğu durumda bu seçeneklerden yalnızca birini seçmek için kullanılan bir kontroldür. Bir yuvarlak ve bir yazıdan oluşur. Tıklandığında yuvarlağın içi çarpılır. Her radyo düğmesi ayrı bir kontroldür(penceredir), o halde programcı seçenekleri oluşturabilmek için birden fazla radyo düğmesi yaratması gerekir.
Radyo düğmeleride otomatik ve manual olamk üzere ikiye ayrılır. Manual radyo düğmelerinde grup kavramı yoktur. Bir radyo düğmesi çarpılandığında daha önce çarpılanmış olanın çarpısını kaldırmak programcının sorumluluğunadadır. Halbuki otomatik radyo düğmelerinde bir grup oluşturulur, bu grupta bir radyo düğmesi çarpılandığında onu çarpılamak ve daha öncesinde çarpılanmış olanın çarpısını kaldırmak kontroller tarafından otomatik olarak yapılmaktadır. Bir düğme grubu WS_GROUP pencere biçimine sahip olan pencereden başlar, Z sıarasına göre ilk WS_GROUP pencere biçimine sahip düğmeye kadar devam eder. Ilk WS_GROUP pencere biçimine sahip düğme gruba dahildir fakat ikincisi değildir.

Radyo düğmeleride “button” sınıf ismi kullanılarak yaratılır. Ancak pencere biçimi olarak BS_RADIOBUTTON yada BS_AUTORADIOBUTTON girilmelidir. Eğer hiçbir radyo düğmesi WS_GROUP özelliğine sahip değilse tüm radyo düğmeleri aynı grupta kabul edilir.

```
static HWND hRadioButtons[NRADIO_BUTTONS];

        switch (message) {
                case WM_CREATE:
                        {
                        int i;
                        char text[2] = {'A', '\0'};

                        for (i = 0; i < 5; ++i) {

                                hRadioButtons[i] = CreateWindow("button", text ,
                                        WS_CHILD|WS_VISIBLE|WS_BORDER|
                                        BS_AUTORADIOBUTTON | ((i == 0) ? WS_GROUP : 0),
                                        100, 100 + 20 * i, 50, 20,  hWnd,
                                        (HMENU) (ID_RADIOBUTTON_BASE + i),
                                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                                ++text[0];
                                }

                        text[0] = '1';
                        for (;i < NRADIO_BUTTONS; ++i) {

                                hRadioButtons[i] = CreateWindow("button", text ,
                                        WS_CHILD|WS_VISIBLE|WS_BORDER|
                                        BS_AUTORADIOBUTTON | ((i == 5) ? WS_GROUP : 0),
                                        100, 100 + 20 * i, 50, 20,  hWnd,
                                        (HMENU) (ID_RADIOBUTTON_BASE + i),
                                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                                ++text[0];
                                }
                        }
                        break;
```

BM_GETCHECK ve BM_SETCHECK mesajları aynı şekilde radyo düğmeleri için de kullanılmaktadır.

Hangi radyo düğmesinin çarpılanmış olduğunun anlamak için grup içersindeki düğmelerin durumşlarının tek tek sorgulanması gerekir. eğer düğmelerin Id değerlerinin peşisıra verirsek aşağıdaki gibi bir döngü ile çarpılanmış olan düğmeyi tespit edebiliriz.

```
for(i = 0; i < NBUTTONS; ++i) {
        hRadio = GetDlgItem(hWnd, IDBUTTONBASE + i);
        if(SendMessage(hRadio, BM_GETCHECK, 0, 0)
                break;
}
```

Radyo düğmeleride diğer düğmelerde olduğu gibi BN_CLICKED işlem kodlu mesaj göndermektedir.

KAYNAK KULANIMI

Bir windows programındaki menüler, bitmapler, simgeler, diyalog pencereleri gibi görsel öğelere kaynak(resource) denilmektedir. Programın kaynakları kaynak dili(resource language) denilen basit bir script dilinde yazılır ve ismine kaynak derleyivi(resource compiler) denilen derleyiviyle derlenir. Kaynak dilinde yazılmış kodun uzantısı ".rc" biçiminde, derlenmiş kaynak dosyasının uzantısı ".res" biçimindedir. Kaynak dili normal text biçimde oluşturulan basit script dilidir. Derlenerek ".res" haline dönüştürülmüş olan dosya linker tarafından PE formatının kaynak bölümü denilen yerine yerleştirilir.

```
Kaynak Dili   ->  Kaynak derleyici  -> Linker ->  PE
                  .rc                        .res
```

Visual studio IDE ortamında .rc dosyasını projeye yerleştirmek yeterlidir. Artık build işlemi yapıldığında bu .rc dosyasının derlenmesi ve PE formatına yerleştirilmesi otomatik olarak yapılmaktadır.

PE formatının kaynak bölümü PE formatının başlık kısmında yazmaktadır. Başlık kısmı ise dosyanın hemen başında bulunur. Bu nedenle kaynak işlemi yapan fonksiyonlar bizden hInstance değerini parametre olarak isterler. Fonksiyonlar bu sayede başlık kısmına ulaşarak kaynak bölümüne erişebilmektedir.

Kaynakların PE formatın kaynak bölümüne yerleştirilmesi bunların otomatik olarak görüntüleneceği anlamına gelmez. Kaynağın görüntülenmesi için programcının o kaynağı oradan çekip belleğe yüklemesi ve sonra işleme sokması gerekmektedir. Kaynakları yüklemek için "LoadXXX " biçiminde ismlendirilmiş bir grup API fonksiyonu vardır. Örneğin LoadMenu, menu kaynağını yüklemek için, LoadIcon Icon kaynağını yüklemek için, LoadBitmap Bitmap kaynağını görüntülemek için kullanılır. LoadResource isimli fonksiyon ise genel olarak hertürlü kaynağı yükleyebilmektedir.

Bir PE formatının içerisindeki kaynakları görüntüleyen ve onların değiştirilmesini sağlayan araçlar vardır. Böylelikle örneğin bir programın menüsü hiç o programı derleyip link etmeden bu araçlar sayesinde değiştirilebilir. Bir programdaki bir kaynak kopyalanarak başka bir programa aktarılabilir.

Örnek1: autoradiobutton

```
#define ID_CHECKBOX             100
….
….
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hCheckBox;

        switch (message) {
                case WM_CREATE:
                        hCheckBox = CreateWindow("button", "Test",
                                WS_CHILD|WS_VISIBLE|WS_BORDER|BS_CHECKBOX,
```

```
                        100, 100, 50, 20, hWnd, (HMENU) ID_CHECKBOX,
                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                SendMessage(hCheckBox, BM_SETCHECK, TRUE, 0);
                break;
        case WM_COMMAND:
                if (LOWORD(wParam) == ID_CHECKBOX &&
                    HIWORD(wParam) == BN_CLICKED) {
                        int stat = SendMessage(hCheckBox, BM_GETCHECK, 0, 0);
                        SendMessage(hCheckBox, BM_SETCHECK, !stat, 0);
                }
                break;
        case WM_LBUTTONDOWN:
                MessageBox(hWnd, SendMessage(hCheckBox, BM_GETCHECK, 0, 0) ?
                                "Checked" : "Unchecked", "Test", MB_OK);
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

Örnek2: bm_checkradiobutton

```
#define ID_CHECKBOX             100
…
…
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HWND hCheckBox;

    switch (message) {
        case WM_CREATE:
            hCheckBox = CreateWindow("button", "Test",
                WS_CHILD|WS_VISIBLE|WS_BORDER|BS_CHECKBOX,
                100, 100, 50, 20, hWnd, (HMENU) ID_CHECKBOX,
                ((LPCREATESTRUCT) lParam)->hInstance, NULL);
            SendMessage(hCheckBox, BM_SETCHECK, TRUE, 0);
            break;
        case WM_COMMAND:
            if (LOWORD(wParam) == ID_CHECKBOX &&
                HIWORD(wParam) == BN_CLICKED) {
                int stat = SendMessage(hCheckBox, BM_GETCHECK, 0, 0);
                SendMessage(hCheckBox, BM_SETCHECK, !stat, 0);
            }
            break;
        case WM_LBUTTONDOWN:
            MessageBox(hWnd, SendMessage(hCheckBox, BM_GETCHECK, 0, 0) ?
                        "Checked" : "Unchecked", "Test", MB_OK);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

Örnek3: ws_groupradiobutton:

```
#define ID_CHECKBOX             100
…
…
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hCheckBox;

        switch (message) {
                case WM_CREATE:
                        hCheckBox = CreateWindow("button", "Test",
                                WS_CHILD|WS_VISIBLE|WS_BORDER|BS_CHECKBOX,
                                100, 100, 50, 20, hWnd, (HMENU) ID_CHECKBOX,
                                ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        SendMessage(hCheckBox, BM_SETCHECK, TRUE, 0);
                        break;
                case WM_COMMAND:
                        if (LOWORD(wParam) == ID_CHECKBOX &&
                            HIWORD(wParam) == BN_CLICKED) {
                                int stat = SendMessage(hCheckBox, BM_GETCHECK, 0, 0);
                                SendMessage(hCheckBox, BM_SETCHECK, !stat, 0);
                        }
                        break;
                case WM_LBUTTONDOWN:
                        MessageBox(hWnd, SendMessage(hCheckBox, BM_GETCHECK, 0, 0) ?
                                        "Checked" : "Unchecked", "Test", MB_OK);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

Uygulamada kaynaklar kaynak diliyle oluşturulmazlar. İsmine kaynak editörü denilen editörlerle görsel bir biçimde oluşturulurlar. Kaynak editörü fare hareketleri ile yapılan görsel editörlerden .RC dosyasını üretir. Yani programcının aslında kaynak dilini öğrenmesine gerek yoktur.

Kaynak dilinde bir kaynak tanımlamanın genel biçimi aşağıdaki gibidir:

```
<kaynak ismi> <kaynak türü> [parametreler]
{
        Eleman tanımlamaları
}
```

Küme parantezleri yerine begin, end anahtar sözcükleri de kullanılabilir. Kaynak ismi sayısal yada alfabetik olarak belirlenebilir. Yani kaynak ismi bir sayı yada gerçek bir isim biçiminde olabilir. Eğer kaynak isminin tüm karakterleri sayısal ise kaynak ismi sayısal olarak belirtilmiştir; tüm karakterler sayısal değilse alfabetik olarak belirtilmiştir. Örneğin: 123 sayısal bir belirleme; abc alfabetik bir belirlemedir (“123” alfabetik bir belirlemedir). Kaynak isimlerinin sayısal düzeyde oluşturulması genel olarak kaynağa daha hızlı erişilmesine yol açmaktadır. Örneğin Visual C kaynak editörü de kaynakları sayısal olarak isimlendirmektedir. Kaynak isimlendirmesinde büyük harf küçük harf duyarlılığı yoktur. Kaynak ismini kaynağın ne kaynağı olduğunu belirten bir anahtar sözcük izler. Örneğin menu kaynağı için anahtar sözcük

menu; icon kaynağı için icon biçimindedir. Daha sonra o kaynağa ilişkin başka parametrik yapılar varsa onların tanımlamaları yapılır.

ICON Kaynağının Kullanılması

Özel bitmap dosyalarına icon denilmektedir. (32x32, 16x16) boyutlarındadır ve uzantıları da .ico şeklindedir.

Windows'ta iconlar çeşitli biçimde karşımıza çıkabilmektedir. Örneğin pencerelerin sol üst köşelerinde 16x6'lık iconlar bulunur. Görev çubuğundaki iconlar da böyledir.

Bir icon görüntüsünü program içinde kullanmak için şunları yapmak gerekir.

1. önce icon ".rc" dosyasında kaynak olarak belirtilir. İcon kaynağı kaynak dilinde şöyle tanımlanır.:
   <kaynak ismi> ICON <.ico dosyası>

   örneğin: MYICON ICON X.ICO

   X.ICO dosyası kaynak derleyici tarafından alınır ve linker tarafından bu dosyanın içerisindeki görüntü dataları PE formatının içindeki kaynak bölümüne yerleştirilir.
2. Programın çalışma zamanında programın icon kaynağının LoadIcon API fonksiyonuyla PE formatının kaynağından alınması gerekir. bu işlemden HICON türünden bir handle elde edilir.
3. Bu elde edilen handle program içinde kullanılır.

Anahtar Notlar:

Kaynakları yükleyen özel API fonksiyonları vardır. Örneğin LoadIcon icon kaynağını, LoadBitmap bitmap kaynağını yüklemek için kullanılır. Kaynak yükleyen bir API fonksiyonunun tipik olarak parametrik biçimi şöyledir:

```
HXXX LoadXXX(HINSTANCE hInstance, LPCSTR lpszResName);
```

Görüldüğü gibi yükleyici fonksiyonların 1. parametresi PE formatının belleğe yüklenme adresi olan hInstance değeridir. Fonksiyon, PE dosyasının başlık kısmından faydalanarak kaynak bölümünün bellekteki yerini tespit etmektedir.

*Fonksiyonun 2. parametresi const char * türünden kaynak ismidir. Fonksiyon yüklediği kaynağın handle değeri ile geri döner. Kaynak isimleri, bir kaynak türü için tek olmaktadır. Yani aynı isimli farklı türlerden kaynaklar olabilir ama aynı isimli aynı türden kaynak olamaz.*

*Kaynak yükleyen fonksiyonlar kaynağın ismini her zaman const char * türüyle isterler. Eğer kaynak alfabetik olarak isimlendirilmişse bu parametreye kaynak isminin başlangıç adresi girilmelidir. Yok eğer kaynak sayısal isimlendirilmişse adres bilgisi yüksek anlamlı WORD 0; düşük anlamlı WORD kaynağın sayısal değeri olacak biçimde girilmelidir. Yani yükleyici fonksiyon bu adresin önce yüksek anlamlı WORD değerine bakar. Yüksek anlamlı WORD değer 0 ise kaynağın sayısal olduğunu anlar; kaynağın sayısal değerini düşük anlamlı WORD'den çeker. Eğer yüksek anlamlı WORD 0 değilse kaynağın alfabetik isimlendirildiğini anlar; kaynak ismini belirtilen adreste arar. Yani bu durumda sayısal bir kaynak yüksek anlamlı WORD değeri 0, düşük anlamlı WORD değeri kaynağa ilişkin sayı olan bir adres bilgisi olarak fonksiyona verilmelidir. Bu işlemi kolaylaştırmak için windows.h içerisinde MAKEINTRESOURCE denilen bir makro vardır.*

```
#define MAKEINTRESOURCE(val) ((LPCTSTR) (WORD) (val))
```

Özetle kaynak ismi alfabetikse bu isim doğrudan "" içinde string olarak girilebilir. Fakat kaynak sayısalsa MAKEINTRESOURCE makrosuyla girilmelidir.

Programın Icon Görüntüsünün Belirlenmesi

Bir pencerenin icon görüntüsü pencerenin yaratıldığı WNDCLASS yapısının yaratıldığı hIcon elemanı ile belirtilir. WNDCLASS yapısının hIcon elemanına hangi icon kaynağının handle değeri girilirse pencerenin icon görüntüsü o olur. İskelet API programında sınıfın elemanına şöyle değer atanmıştır:

wndClass.hIcon = LoadIcon(NULL, IDI_QUESTION);

LoadIcon fonksiyonunun prototipi şöyledir:

```
HICON LoadIcon(
  HINSTANCE hInstance,        // handle to application instance
  LPCTSTR lpIconName          // icon-name string or icon resource identifier
);
```

Fonksiyonun 1. parametresi NULL geçilirse bu durum windows'un içerisinde hazır bulunan iconların kullanılacağı anlamına gelir. Bu durumda 2. parametre şunlardan biri olabilir:

Value	***Description***
IDI_APPLICATION	Default application icon.
IDI_ASTERISK	Same as IDI_INFORMATION.
IDI_ERROR	Hand-shaped icon.
IDI_EXCLAMATION	Same as IDI_WARNING.
IDI_HAND	Same as IDI_ERROR.
IDI_INFORMATION	Asterisk icon.
IDI_QUESTION	Question mark icon.
IDI_WARNING	Exclamation point icon.
IDI_WINLOGO	Windows logo icon.

Win16 zamanlarında görev çubuğu yoktu ve pencerenin sol üst köşesinde icon resmi de çıkmıyordu. Dolayısıyla o zamanlardaki bütün iconlar 32x32'lik büyük iconlardı. Win32 sistemlerine geçildiğinde 16x16'lık küçük iconlar da tanımlandı. WNDCLASS yapısında belirttiğimiz hIcon, 32x32'lik icon şeklini almaktadır. Bu büyük icondan Windows küçük icon şeklini kendisi elde etmektedir. Eğer programcı bu iki icon görüntüsünü ayırmak isterse WNDCLASS yapısının yeni bir biçimi olan WNDCLASSEX yapısını kullanmaktadır. Bu yeni yapıyı register ettirmek için RegisterClassEx API fonksiyonu kullanılmaktadır.

Kaynak derleyicisi C'nin önişlemcisini içermektedir. Yani kaynak derleyicisi #define, #include gibi önişlemci kodlarını işleyebilir. Kaynaklarda daha düzenli çalışmak için şu yollar önerilir:

1- Sayısal kaynak isimleri ve kaynaktaki parametrik değerler örneğin resource.h isimli bir dosyada tutulabilir.
2- Oluşturulan bu başlık dosyası hem .RC dosyasından hem de .C dosyasından include edilir.

Örnek1: myresource

Source files içerisinde .c ve .rc'ler bulunacak. Resource files içerisinde ise .ico'lar yani resource'lar bulunacak.

/* icon.c */

```
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(MYICON));
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}


	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;


	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HICON hIcon;
	HDC hDC;
	PAINTSTRUCT ps;
```

```
        switch (message) {
                case WM_CREATE:
                        hIcon = LoadIcon(((LPCREATESTRUCT) lParam)->hInstance, MAKEINTRESOURCE(MYICON));
                        if (hIcon == NULL)
                                return -1;
                        break;
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);
                        DrawIcon(hDC, 100, 100, hIcon);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
/* myresource.rc */
#include "resource.h"
MYICON          ICON    "X.ICO"
```

resource.h ise aşağıdaki gibidir:

```
#define MYICON                  100
```

String Kaynağı

Program içerisinde kullanılacak çeşitli yazıları kaynak koda gömmek yerine bir kaynak biçiminde oluşturup kullanabiliriz. Böylece bu yazıların programı yeniden derleyip link etmeden kaynak editörlerini kullanarak değiştirebiliriz.

STRING kaynağının genel biçimi şöyledir:

```
STRINGTABLE
{
        kaynak_ismi   "yazı"
        kaynak_ismi   "yazı"
        kaynak_ismi   "yazı"
}
```

Program içerisindeki tüm string kaynakları bir string tablosu içinde toplanır ve birlikte yazılır. Her string yazısının kaynak ismi birbirinden farklı olabilir. Bir string kaynağını yüklemek için LoadString() API fomksiyonu kullanılır.

```
int LoadString(
  HINSTANCE hInstance,  // handle to module containing string resource
  UINT uID,             // resource identifier
  LPTSTR lpBuffer,      // pointer to buffer for resource
  int nBufferMax        // size of buffer
);
```

String kaynaklarının ismi sayısal olmak zorundadır. Fonksiyonun birinci parametresi PE formatının yüklenme adresidir. İkinci parametresi yüklenecek string kaynağının sayısal ismidir. 3. ve 4. parametreler, string kaynağındaki yazının yerleştirileceği dizinin adresi ve uzunluğudur.

MENU İŞLEMLERİ

Programın ana menüsüne **"Pulldown Menü Sistemi"** denir. Bu menü sistemi, bir menü çubuğu (menu bar) ve popup menülerden oluşmaktadır. Popup menünün satırlarına menü elemanları denir. Bir menü elemanı başka bir popup mneü içerebilir. Bir Windows uygulamasında menü oluşturmak otomatik olarak menü mesalarının işlenmesini gerektirmez. Programcı önce menüyü oluşturmalı, sonra menü elemanı seçildiğinde neler yapılacağını belirlemelidir. Bir programda menü 3 biçimde oluşturlabilir.

1- Menü kaynağı oluşturulur ve programda **WNDCLASS** yapısının **lpszMenuName** elemanına bu menü kaynağının ismi verilir. Böylece bu pencere sınıfı kullanılarak yaratılan bütün pencerelerde bu menü görüntülenir.
2- WNDCLASS yapısının **lpszMenuName** elemanı NULL geçilebilir. Oluşturulan menü kaynağı LoadMenu API fonksiyonuyla yüklenir. Buradan bir handle değeri elde edilir. Bu handle değeri, CreateWindow() fonksiyonun 9. parametresi olarak girilir. Anımsanacağı gibi CreateWindow() fonksiyonun 9. parametresi olan HMENU parametresi, alt pencerelerde, alt peencere ID değeri olarak kullanılır. Alt pencreler menüye sahip olamazlar.
3- Hiç kaynak oluşturmadan menüler, programın çalışma zamanı sırasında, dinamik olarak oluşturulabilirler. Bu konu ileride ele alınacaktır.

Menü kaynağı, .rc dosyasında, aşağıdaki gibi oluşturulur:

```
<kaynak ismi>   MENU
{
        popup "yazı"
        {
                menuitem   "yazı"    id
                menuitem   "yazı"    id
                menuitem   "yazı"    id
                .......
        }
        popup "yazı"
        {
                menuitem   "yazı" ,    id
                menuitem   "yazı" ,   id
                menuitem   "yazı" ,  id
                .......
        }
}
```

Görüldüğü gibi her menü elemanın bir ID si vardır ancak popupların yoktur. Menü ID lerinin sembolik sabit biçimide başlık dosyasında define edilmesi iyi bir tekniktir. Geleneksel olarak bu sembolik sabitler **ID_XXX_YYY** biçimindedir. Burada **XXX,** popup elemanın ismi **YYY** ise menü elemanın ismidir(örnek: ID_FILE_OPEN)

Örneğin şöyle bir menü olsun

File Edit Exit

Open Copy
Close Paste

Bu menüye ilişkin kaynak şöyle yazılabilir.

```
MYMENU MENU
{
        popup "File"
        {
                menuitem "Open"   , ID_FILE_OPEN
                menuitem "Close"  , ID_FILE_CLOSE
        }

        popup "Edit"
        {
                menuitem "Copy"  , ID_FILE_COPY
                menuitem "Paste"  , ID_FILE_PASTE
        }

        menuitem  "Exit"
}
```

Menü yazılarında bir karakterin önüne & getirilirse o karakterin altı çizilir.

```
HMENU LoadMenu(
HINSTANCE hInstance, // handle to application instance
LPCTSTR lpMenuName   // menu name string or menu-resource
                     // identifier
);
```

/* menu.c dosyası */

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(....)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;


        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
```

```
	}

	hWnd = CreateWindow("generic", "Menu Resource",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		LoadMenu(hInstance, "MYMENU"),
		hInstance,
		NULL);

	if (!hWnd)
		return -1;


	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch (message) {
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

---

menu.rc dosyası

```
#include "resource.h"

MYMENU		MENU
{
	popup "&File"
	{
		menuitem "&Open"  ,   ID_FILE_OPEN
		menuitem "&Close" ,   ID_FILE_CLOSE
		menuitem "&Exit"  ,   ID_FILE_EXIT
	}
	popup "&Edit"
	{
		menuitem "&Cut"    ,   ID_EDIT_CUT
		menuitem "&Paste"  ,   ID_EDIT_PASTE

popup "Other"
		{
			menuitem "test", ID_EDIT_OTHER_TEST
		}
	}
}
```

/* **resource.h** dosyası*/

```
#define ID_FILE_OPEN            100
#define ID_FILE_CLOSE           101
#define ID_FILE_EXIT            102
#define ID_EDIT_CUT             103
#define ID_EDIT_PASTE           104
#define ID_EDIT_OTHER_TEST      105
```

MENÜ MESAJLARININ İŞLENMESİ

Kullanıcı bir menü elemanını seçtiğinde Windows, menü elemanın seçildiğini belirtmek için kuyrua **WM_COMMAND** mesajını bırakır. Programcı **WM_COMMAND** mesajını işleyerek menü seçimini ele alır. **WM_COMMAND** mesajı, Windows kontrolleri tarafından da gönderilmektedir. O halde bu mesajın parametrelerini daha geniş olarak yeniden ele almalıyız.

LOWORD(wParam) -> Eğer mesaj kontrolden geliyorsa kontrolün ID değeri; menüden geliyorsa menü elemanın ID değeri vardır.

HIWORD(wParam) -> Mesaj Kontrolden gönderilmişse burada işlem kodu(notification code) yani mesajın ne sebeple gönderildiği bilgisi vardır. Eğer mesaj menü nedeniyle gönderilmişser burada ya "0" ya da "1" bulunur. "0", mesajın menü elemanın seçilmesinden dolayı oluştuğunu, "1", kısayol tuşu ile oluştuğunu bildirir.

lParam -> Mesaj kontrolden geliyorsa burada kontrolün handle değeri, menüden geliyorsa "0" değeri bulunur.

Menü mesajları tipik olarak **LOWORD(wParam)** değerinin **switch** içine alınmasıyla elde edilir.

```
case WM_COMMAND:
{
        switch (LOWORD(wParam)) {
            case ID_FILE_OPEN:
                /*...*/
            case ID_FILE_CLOSE:
                /*...*/
            case ID_BUTTON:
                /*...*/
        }
        break;
}
```

Peki hem kontrollerimiz hem de menümüz olsa, **WM_COMMAND** mesajında karışıklık olurmu? Bu durumda iki şey önerilebilir:

1- Programcı, menü elemanlarının ID değerleriyle kontrollerin ID değerlerini bilinçli olarak farklı verir. Böylece **LOWORD(wParam)** değerini **switch** içerisine alarak her iki kaynaktan gelen mesajıda işler. Bu yöntem, tercih edilmesi gereken yöntemdir.
2- **WM_COMMAND** içerisinde **lParam** değerine bakılarak iki ayrı **switch** oluşturulabilir.

```
case WM_COMMAND:
{
        if (lParam) {
                switch (LOWORD(wParam)) {
        /*...*/
}
        }
        else {
                switch (LOWORD(wParam)) {
        /*...*/
}
        }
}
```

Visual C 6 Kaynak Editörü

Kaynak editörü, kaynakların görsel bir biçimde belirlenmesine olanak sağlar. Programcı kaynakları fare hareketleriyle belirler; kaynak editörü de bu belirlemelere ilişkin iki dosya üretir. .rc dosyasında kaynakların kaynak dilindeki karşılıkları vardır. resource.h dosyasında kaynak isimlerine ilişkin sayısal değerlerin, kaynak parametrelerine ilişkin değerlerin makro ifadeleri vardır. .rc dosyasından resource.h include edilmiştir. Programcı da .rc dosyasını projeye dahil eder ve resource.h dosyasını kaynak koddan include eder.

Visual C kaynak editörü her zaman kaynakları sayısal olarak isimlendirmektedir. Örneğin menu kaynağının ismi IDR_MENU1 biçiminde gözükebilir. Fakat aslında bu isim bir makrodur. Programcı isterse farenin sol tuşuna basarak properties kısmından menu kaynağının veya menu elemanlarının ismini değiştirebilir. Bu durumda kaynak editör resource.h dosyası içerisindeki bu sembolik sabitleri de değiştirecektir. bak: resource_editor

```
 /* generic.c */

#include <windows.h>
#include "resource.h"


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
```

```
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch (message) {
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

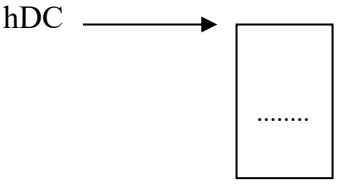
Anahtar Notlar:

Kaynak editörün yarattığı .rc dosyasında resource.h dosyasının yanı sıra afxres.h dosyası da include edilmiştir. Bu dosyanın içerisinde ID_FILE_OPEN, ID_FILE_SAVE gibi klasik menu elemanlarına ilişkin sembolik sabitler de vardır. Maalesef kaynak editör file menusu altında open save gibi elemanlar oluşturulduğunda aynı sembolik sabitleri başka bir değerle resource.h dosyasının içine yazmaktadır. Bu işlem ise derleme sırasında uyarı oluşmasına yol açar. u uyarıyı engellemek için programcı ID'leri bilinçli olarak değiştirebilir.

WINDOWS'TA ÇİZİM İŞLEMLERİ

Windows'ta çizim konusu karmaşık bir konudur. Programcı bir çizim yapmadan önce ismine DC (Device Context) denilen bir handle alanı oluşturur. Tüm GDI fonksiyonları 1. parametre olarak bu handle değerini istemektedir. Çizim, çok bileşenli bir konudur. Çizim yapmak için kalem, fırça gibi pek çok çizim bileşeninin kullanılması gerekir. Bütün bu çizim bileşenleri DC handle alanında saklanmaktadır. Yani biz örneğin bir dikdörtgen çizen rectangle fonksiyonuna bir DC parametre olarak verdiğimizde fonksiyon, dikdörtgenin kenarların DC'deki kalemle çizer, içini de fırça ile boyar.

DC Handle Alanı



Çizim işlemine başlamadan önce bir DC yaratıp Handle elde etmek gerekir. DC yaratan çeşitli API fonksiyonları vardır. DC yaratıldığında bu fonksiyonlar DC alanı içerisindeki çizim bileşenlerini bazı default değerlerle doldururlar. Bu çizim bileşenlerini değiştirmek için SelectObject isimli fonksiyon ile eski bileşeni hadle alanından çıkartıp yeni bileşeni eklemek gerekir.

Windows'un GDI API fonksiyonları pek çok bakımdan eleştirilmektedir. DC alanında çok fazla bilginin tutulması, fonksiyonların parametre sayısını azaltsa da maalesef kullanım zorluğu oluşturmaktadır. .NET sisteminde Microsoft GDI+ denilen farklı bir sınıfsal çizim fonksiyonları kullanmaktadır. Bunlar birer API fonksiyonu değildir. Fakat Windows'un yeni çıkacak Longhorn versiyonunda API fonksiyonları biçimine dönüştürülecektir.

WM_PAINT Mesajı

Windows, bir pencere içerisine yapılan çizimleri otomatik olarak tutup yeniden çizmez. Pencere içerisindeki bozulan görüntünün yeniden çizilmesi programcının sorumluluğuna bırakılmıştır. Windows yalnızca pencerenin görünmeyen bir kısmı görünür hale geldiğinde pencere içerisindeki görüntünün yeniden çizilmesi için kuyruğa WM_PAINT isimli mesajı bırakır. Bu mesajı alan programcının pencere içerisindeki bozulan görüntüyü yeniden çizmesi gerekir. Windows 3 durumda pencere içerisindeki, görüntüyü kendisi saklayıp basmaktadır.

1. Bir menu açıldığında popup penceresi görüntünün bir kısmını kapatır; sonra menu kapatıldığında bu pencerenin altında kalan görüntü Windows tarafından otomatik olarak geri basılır. Yani bu durumda WM_PAINT mesajı oluşmamaktadır.
2. Fare hareket ettirildiğinde fare okunun altında kalan küçük bir alan Windows tarafından otomatik olarak saklanıp yeniden basılmaktadır.
3. Pencere hareket ettirildiğinde Windows pencere içerisindeki görüntüyü yeni yere otomatik olarak taşır. Yani pencerenin taşınması ile WM_PAINT mesajı oluşmamaktadır.

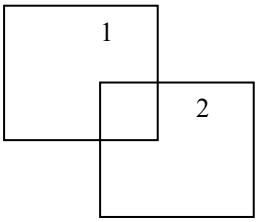
WM_PAINT Mesajı Ne Zaman Oluşur?

WM_PAINT mesajı pencerenin görünmeyen bir kısmı görünür hale geldiğinde sistem tarafından oluşturulmaktadır. Pencerenin görünmeyen bir kısmının görünür hale gelmesi durumu tipik olarak aşağıdaki gibi işlemlerle oluşmaktadır:

— Bir pencere diğer pencerenin üzerinde ise üstteki pencerenin hareket ettirilmesi ile
— Pencerenin bir kısmı masaüstünün dışındadır. Pencere taşınarak görünmeyen bölümleri görünür hale getirilmiştir.
— Pencere minimize durumdan restore, restore durumdan maximize, maximize durumdan restore durumuna getirilmiştir.
— Pencerenin boyutları büyütülmek yada küçültülmek yoluyla değiştiğinde WM_PAINT mesajının oluşturulup oluşturulmayacağı, pencerenin yaratıldığı pencere sınıfında belirtilmektedir. WNDCLASS yapısının style elemanı CD__HREDRAW ve CS_VREDRAW seçeneklerini içeriyorsa pencere yatayda yada düşeyde boyut değiştirdiğinde WM_PAINT mesajı oluşturulur. İskelet programda style elemanına CS_HREDRAW|CS_VREDRAW biçiminde değer verilmiştir. Örneğin biz yalnızca CS_HREDRAW belirlemesi yaparsak pencere düşeyde konum değiştirdiğinde WM_PAINT oluşmayacaktır.

WM_PAINT Mesajının Oluşturulma Biçimi

Windows, her pencere için güncelleme alanı (Update Region) denilen dikdörtgensel bir bilgi tutmaktadır. Güncelleme alanı teknik olarak handle alanı içerisinde saklanır. Güncelleme alanı dikdörtgensel bir bölgedir; eliptik yada dairesel bir alan değildir. Güncelleme alanı, pencere içerisindeki görüntünün düzgün bir biçimde yeniden görüntülenmesi için çizilmesi gereken minimum dikdörtgensel alandır. Örneğin iki pencere aşağıdaki gibi bir durumda olsunlar:



2 numaralı pencere çekildiğinde çizilmesi gereken minimum dikdörtgen, sağ alt köşedeki dikdörtgen alandır.

Windows bir pencerenin görünmeyen kısmı görünür hale geldiğinde hemen WM_PAINT mesajını yollamaz. Yalnızca pencerenin güncelleme alanını günceller. Sonra kendisi için uygun bir zamanda tüm pencerelerin güncelleme alanlarına bakar; güncelleme alanı boş küme olmayan pencerelere WM_PAINT mesajı gönderilir. Güncelleme alanının boş küme olması, pencerenin hiçbir alanının çizilmesine gerek olmadığı anlamına gelir. Özetle pencerenin görünmeyen bir kısmı görünür hale geldiği zaman derhal WM_PAINT mesajı oluşmamakta; uygun bir zamanda bu mesaj oluşturulmaktadır. Aslında Windows hiçbir zaman üst üste kuyruğa WM_PAINT mesajını bırakmaz. Örneğin pencerenin görünmeyen bir kısmı görünür hale gelsin fakat Windows henüz WM_PAINT mesajını bırakmamış olsun. Bir süre sonra daha büyük bir alan görünür hale geldiğinde Windows yine güncelleme alanının günceller ve kuyruğa sadece 1 tane WM_PAINT mesajı bırakır.

DC Alan Fonksiyonlar

Windows'ta çizim yapmak için gereken DC alanın handle değeri hDC ile temsil edilmektedir. DC alan 3 temel API fonksiyonu vardır: BeginPaint, GetDC ve GetWindowDC. Bu fonksiyonların her biri ile alınan DC'lerin çizim özellikleri farklıdır.

BeginPaint fonksiyonu ile alınan DC Orijin noktası, çalışma alanının sol üst kösesidir. Bu DC ile yalnızca güncelleme alanı içerisinde çizim yapılabilir. Örneğin bu DC'yi kullanarak tüm çalışma alanının boyamak isteyelim. Fonksiyon yalnızca güncelleme alanı içerisini boyayabilir. Güncelleme alanı bos küme ise API fonksiyonları hiç çizim yapmazlar. GetDc fonksiyonunun da orijin noktası, çalışma alanının sol üst kösesidir. Fakat bu DC çizim fonksiyonlarına verildiğinde fonksiyonlar yalnızca güncelleme alanını değil çalışma alanının her yerini çizebiliriler. GetWindowDC isle alınan DC'nin orijin noktası ise pencerenin sol üst kösesidir. Bu fonksiyon da güncelleme alanına bakmaksızın her yere çizim yapabilir. Örneğin biz bu DC ile pencere baslığına bile çizim yapabiliriz.

Fonksiyon	Orijin	API Fonksiyonunun Çizildiği Yer	Zemini Silme Durumu
BeginPaint	Çalışma alanının sol üst kösesi	Güncelleme Alanı	Güncelleme Alanının zeminini siler
GetDC	Çalışma alanının sol üst kösesi	Tüm çalışma Alanı	Zemini siler
GetWindowDC	Pencerenin sol üst kösesi	Tüm pencere	Zemini silmez

BeginPaint fonksiyonu diğer iki fonksiyondan farklı olarak zemini de ileride açıklanacağı biçimde silmektedir.

Biz WM_PAINT mesajının dışında örneğin WM_MOUSEMOVE fare mesajının içerisinde çizim yapmak isteyelim. Bu noktada güncelleme alanı genellikle bos küme biçimindedir. O halde biz GeginPaint fonksiyonu ile çizim yapamayız. Çünkü BeginPaint fonksiyonu ile alınan DC ile, sadece güncelleme alanına gerçek bir çizim yapabilmektedir. Peki WM_PAINT mesajı içerisinde çizim yapacaksak DC'yi hangi fonksiyon ile almalıyız? WM_PAINT mesajı, zaten güncelleme alanının bos küme olmaması nedeniyle gönderilmiştir. Biz de en ekonomik olarak yalnızca güncelleme alanının içerisini çizmek isteriz. O halde WM_PAINT mesajı içerisinde çizim yapılacaksa DC'nin BeginPaint fonksiyonuyla alınması gerekir. BeginPaint, çalışma alanının tüm zemininin değil, yalnızca güncelleme alanının zeminini yeniden boyamaktadır. Programcı WM_PAINT mesajının dışında birtakım çizimler yapmışsa bu çizimlerin bilgilerini saklayıp WM_PAINT mesajı içerisinde yeniden çizecek bir mekanizma kurmalıdır. Yoksa pencerenin görünmeyen bir kısmı görünür hale geldiğinde oradaki görüntüler çizilmez.

BeginPaint fonksiyonunun prototipi şöyledir:

```
HDC BeginPaint(
  HWND   hWnd,                      // handle to window
  LPPAINTSTRUCT  lpPaint            //pointer to structure for paint information
);
```

Fonksiyonun 1. parametresi, DC elde edilecek pencerenin handle değeri; 2. parametresi, PAINTSTRUCT türünden bir yapının adresidir. Fonksiyon, bu yapının içini faydalı bilgilerle doldurur ve DC handle değeri ile geri döner. Programcı DC'yi saklamamalıdır. Çizim yaptıktan sonra DC'yi sisteme iade etmelidir. BeginPaint fonksiyonu alınan DC, EndPaint fonksiyonu ile bırakılabilir.

```
BOOL EndPaint(
  HWND hWnd,                        // handle to window
  CONST PAINTSTRUCT *lpPaint        // paint data
);
```

Fonksiyonun 1. parametresi pencerenin handle değeri, 2. parametresi daha önce elde edilen PAINTSTRUCT yapısının adresidir. Bu durumda WM_PAINT mesajı tipik olarak söyle işlenmelidir:

```
HDC hDC;
PAINTSTRUCT ps;
...
case WM_PAINT:
        hDC = BeginPaint(hWnd, &ps);
        ....
        ...
        EndPaint(hWnd, &ps);
        break;
------------
```

GetDC fonksiyonun prototipi ise şöyledir:

```
HDC GetDC(
  HWND hWnd         // handle to a window
);
```

Fonksiyonun parametresi pencerenin handle değeri; geri dönüş değeri DC'nin handle değeridir. GetDC ile alınan handle, ReleaseDC fonksiyonu ile geri bırakılmalıdır:

```
int ReleaseDC(
  HWND hWnd,      // handle to a window
  HDC hDC         // handle to DC
);
```

Fonksiyonun 1. parametresi pencerenin handle değeri; 2. parametresi bırakılacak DC'nin handle değeridir. GetWindowDC'nin parametrik yapısı ise şöyledir:

```
HDC GetWindowDC(
  HWND  hWnd      // handle of window
);
```

Fonksiyonun parametresi pencerenin handle değeri, geri dönüş değeri DC'nin handle değeridir. GetWindowDC ile alınan DC yine ReleaseDC fonksiyonu ile bırakılır.

MoveToEx Ve LineTo Fonksiyonları

LineTo fonksiyonu aktif noktadan belirlenen bir noktaya kadar doğru çizer. LineTo doğruyu DC alanında belirtilen kale ile çizer. DC yaratıldığında default kalemi siyah bir kalemdir. Yine DC yaratıldığında aktif nokta (0,0) noktasıdır. LineTo fonksiyonu aktif noktayı doğrunun ucuna öteler.

```
BOOL LineTo(
  HDC hDC,  // device context handle
  int  nXEnd,  // x-coordinate of line's ending point
  int  nYEnd,  // y-coordinate of line's ending point
);
```

Fonksiyonun 1. parametresi DC'nin handle değeri; 2. ve 3. parametreler doğrunun uç nokta koordinatlarıdır. bak paint.c

```
/* paint.c */

#include <windows.h>


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(...)
{
        ...
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;

        switch (message) {
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);

                        LineTo(hDC, 100, 100);
                        LineTo(hDC, 200, 300);

                        EndPaint(hWnd, &ps);
                        break;
```

```
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

MoveToEx fonksiyonu aktif noktayı değiştirmek için kullanılır. Prototipi şöyledir:

```
BOLL MoveToEx(
  HDC  hDC,          //handle to devie context
  int X,             // x-coordinate of new current position
  int Y,             // y-coordinate of new current position
  LPPOINT lpPoint  //pointer to old current position
);
```

Fonksiyonun 1. parametresi DC'nin handle değeri, 2. ve 3. parametreleri de aktif noktanın koordinatıdır. Fonksiyonun son parametresi aktif noktanın önceki değerinin yerleştirileceği point yapısının adresidir. Bu parametre NULL olarak geçilebilir. Bu durumda önceki nokta yerleştirilmez. bak dikdort.c

```
/* dikdort.c */
.....
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;

	switch (message) {
		case WM_PAINT:
			hDC = BeginPaint(hWnd, &ps);

			MoveToEx(hDC, 100, 100, NULL);

			LineTo(hDC, 200, 100);
			LineTo(hDC, 200, 200);
			LineTo(hDC, 100, 200);
			LineTo(hDC, 100, 100);


			EndPaint(hWnd, &ps);
			break;

		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

YAZ-BOZ TAHTASI PROGRAMI

Yaz-boz tahtası programı çok basittir. Her iki WM_MOUSEMOVE mesajında önceki nokta ile sonraki nokta arasında doğru çizilir. Tabi çizim için farenin tuşunun basili olup olmadığına bakılmalıdır. Çizilen çizgiler bir veri yapısında saklanıp WM_PAINT mesajında yeniden çizilmezse yazılar kaybolabilir.

WM_MOUSEMOVE mesajında çizimi yapmak için DC, GetDC fonksiyonuyla alınmalı, ReleaseDC fonksiyonu ile bırakılmalıdır.

```
...
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static int prevX, prevY;

	switch (message) {
		case WM_LBUTTONDOWN:
			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);
			break;
		case WM_MOUSEMOVE:
			if (!(wParam & MK_LBUTTON))
				break;
			hDC = GetDC(hWnd);
			MoveToEx(hDC, prevX, prevY, NULL);
			LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);
			ReleaseDC(hWnd, hDC);
			break;
		case WM_PAINT:
			hDC = BeginPaint(hWnd, &ps);

			EndPaint(hWnd, &ps);
			break;

		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

Programda WM_LBUTTONDOWN ve WM_MOUSEMOVE mesajlarının işlenmesi söyle olabilir:

```
case WM_LBUTTONDOWN:
	prevX = LOWORD(lParam);
	prevY = HIWORD(lParam);
	break;
case WM_MOUSEMOVE:
	if (!(wParam & MK_LBUTTON))    // iki tusa birden basilip basilmadigi kontrol ediliyor.
		break;
	hDC = GetDC(hWnd);
```

```
		MoveToEx(hDC, prevX, prevY, NULL);
		LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
		prevX = LOWORD(lParam);
		prevY = HIWORD(lParam);
		ReleaseDC(hWnd, hDC);
		break;
```

yapboz.cpp'yi de şu şekilde yazabiliriz:

```
#include <windows.h>
#include <list>
#include "resource.h"

using namespace std;

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnFileSave(const list<POINT> &pointList);
void OnFileOpen(list<POINT> &pointList);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpszCmdParam, int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
```

```
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static int prevX, prevY;
	static list<POINT> pointList;
	POINT point;

	switch (message) {
		case WM_COMMAND:
			if (LOWORD(wParam) == ID_FILE_SAVE_MENU)
				OnFileSave(pointList);
			else if (LOWORD(wParam) == ID_FILE_OPEN_MENU) {
				OnFileOpen(pointList);
				InvalidateRect(hWnd, NULL, TRUE);
			}
			else if (LOWORD(wParam) == ID_FILE_CLEAR_MENU) {
				pointList.clear();
				InvalidateRect(hWnd, NULL, TRUE);
			}
			break;
		case WM_LBUTTONDOWN:
			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);
			break;

		case WM_MOUSEMOVE:
			if (!(wParam & MK_LBUTTON))
				break;
			point.x = prevX;
			point.y = prevY;
			pointList.push_back(point);

			hDC = GetDC(hWnd);
			MoveToEx(hDC, prevX, prevY, NULL);
			LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);

			point.x = prevX;
			point.y = prevY;

			pointList.push_back(point);

			ReleaseDC(hWnd, hDC);
			break;
		case WM_PAINT:
			{
				hDC = BeginPaint(hWnd, &ps);

				list<POINT>::iterator iter = pointList.begin();

				while (iter != pointList.end()) {
					MoveToEx(hDC, iter->x, iter->y, NULL);
					++iter;
					LineTo(hDC, iter->x, iter->y);
					++iter;
				}
```

```
                                EndPaint(hWnd, &ps);

                        }
                        break;

                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnFileSave(const list<POINT> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "wb")) == NULL) {
                MessageBox(NULL, "Cannot create file!..", "Error", MB_OK);
                return;
        }

        list<POINT>::const_iterator iter = pointList.begin();

        for (; iter != pointList.end(); ++iter) {
                fwrite(&*iter, 1, sizeof(POINT), f);
        }

        fclose(f);
}

void OnFileOpen(list<POINT> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "rb")) == NULL) {
                MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
                return;
        }

        pointList.clear();

        POINT point;
        for (;;) {
                fread(&point, 1, sizeof(POINT), f);
                if (feof(f))
                        break;
                pointList.push_back(point);
        }
}

/* resource.h */

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by paint.rc
//
#define IDR_MENU1                       101
#define ID_FILE_OPEN_MENU               40001
#define ID_FILE_SAVE_MENU               40002
```

```
#define ID_FILE_EXIT_MENU              40003
#define ID_FILE_CLEAR_MENU             40004

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE       102
#define _APS_NEXT_COMMAND_VALUE        40005
#define _APS_NEXT_CONTROL_VALUE        1000
#define _APS_NEXT_SYMED_VALUE          101
#endif
#endif
```

Oluşturulan .rc dosyası da aşağıdaki gibidir:

```
/* resource.rc */

//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// Turkish resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_TRK)
#ifdef _WIN32
LANGUAGE LANG_TURKISH, SUBLANG_DEFAULT
#pragma code_page(1254)
#endif //_WIN32

////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MENU1 MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open",                       ID_FILE_OPEN_MENU
```

```
        MENUITEM "&Save",                ID_FILE_SAVE_MENU
        MENUITEM "&Clear",               ID_FILE_CLEAR_MENU
        MENUITEM "&Exit",                ID_FILE_EXIT_MENU
    END
END

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED

#endif    // Turkish resources
////////////////////////////////////////////////////////////////////////////

#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//

////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

Güncelleme Alanı Üzerinde İşlemler

Güncelleme alanının boş küme olduğunu düşünelim. WM_PAINT mesajını SendMessage yada PostMessage ile gönderdiğimiz zaman herhangi bir çizim yapılamayacaktır. Çünkü BeginPaint ile handle alınacağına göre ve BeginPaint ile alınan handle yalnızca güncelleme alanına çizim yapılabileceğine göre gerçek bir çizim yapılamayacaktır. Bu nedenle herhangi bir zaman WM_PAINT mesajında çizim işlemlerini gerçekleştirebilmek için güncelleme alanını boş küme olmaktan çıkartmak gerekir. InvalidateRect fonksiyonu, bu işlemi yapmaktadır.

BOOL InvalidateRect(

```
  HWND hWnd,              // handle of window with changed update region
  CONST RECT *lpRect,     // address of rectangle coordinates
  BOOL bErase             // erase-background flag
);
```

Fonksiyonun 1. parametresi pencerenin handle değerini, 2. parametresi yeni güncelleme alanının koordinatlarını belirtir. Yani yeni güncelleme alanı bu parametre ile belirlenebilmektedir. Fonksiyonun son parametresi zeminin silinip silinmeyeceğini belirtir. TRUE geçilirse zemin silinir, FALSE geçilirse zemin silinmez. Bu konu ileride ele alınacaktır. Fonksiyonun 2. parametresi NULL geçilirse tüm çalışma alanı güncelleme alanı olur.

Bazen tam ters bir biçimde güncelleme alanını küçültmek yada tamamen yok etmek isteriz. Bu işlem ValidateRect fonksiyonu ile yapılmaktadır. Prototipi şöyledir:

```
BOOL ValidateRect(
 HWND hWnd,              // handle of window
 CONST RECT *lpRect      // address of validation rectangle coordinates
);
```

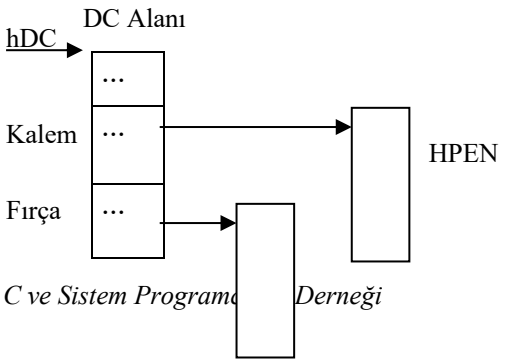
Fonksiyonun 1. parametresi pencerenin handle değerini, 2. parametresi güncelleme alanından çıkartılacak dikdörtgensel alanı belirtir. 2. parametre NULL geçilirse güncelleme alanı boş küme yapılır. BeginPaınt fonksiyonu güncelleme alanının boş küme yapar. Eğer böyle yapmasaydı sürekli WM_PAINT mesajı gelirdi. GetUpdateRect fonksiyonu ile güncelleme alanı herhangi bir zaman elde edilebilir.

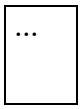
```
BOOL GetUpdateRect(
  HWND hWnd,        // handle of window
  LPRECT lpRect,    // address of update rectangle coordinates
  BOOL bErase       // erase flag
);
```

Fonksiyonun 1. parametresi pencerenin handle değeri, 2. parametresi güncelleme alanının yerleştirileceği RECT türünden yapının adresidir. 3. parametre zeminin silinip silinmeyeceğini belirtir. FALSE geçilebilir. Güncelleme alanı boş küme ise fonksiyon 0 değerine geri dönmekte; değilse 0 dışı bir değere geri dönmektedir.

ÇİZİM NESNELERİ

Windows'ta çizimlerde kullanılan kalem fırça gibi çizim bileşenleri birer handle değerine sahip nesnelerdir. Örneğin kalemin handle değeri HPEN ile, fırçanın handle değeri HBRUSH ile temsil edilir. DC alanında bu çizim bileşenlerinin handle değerleri tutulmaktadır. Yani bir çizim fonksiyonuna biz DC handle değerini geçtiğimizde çizim fonksiyonu, hangi kalemle yada fırçayla çizileceğini DC alanı içerisindeki bu handle değerleri yardımıyla temsil eder.





HBRUSH

DC yaratıldığında DC alanında default çizim nesneleri vardır. Örneğin default kalem nesnesi siyah düz kale; default fırça nesnesi beyaz fırçadır. Bir çizim nesnesinin değiştirilmesi için yeni bir çizim nesnesinin yaratılması ve bunun DC alanına bağlanması gerekir. Yeni bir çizim nesnesinin DC alanına bağlanması için SelectObject SPI fonksiyonu kullanılmaktadır.

```
HGDIOBJ SelectObject(
 HDC hdc,              // handle to device context
 HGDIOBJ hgdiobj   // handle to object
);
```

Fonksiyonun 1. parametresi DC'nin handle değeridir. 2. parametresi çizim nesnesinin handle değeridir. HGDIOBJ, tüm çizim nesnelerini temsil eden bir türdür. Yani biz bu 2. parametreye kalem nesnesinin yada fırça nesnesinin handle değerini geçirebiliriz. Fonksiyonun geri dönüş değeri DC'de saklı bulunan önceki çizim nesnesidir. Yani biz bu fonksiyonla yeni bir kalemi handle (DC) alanına yerleştirecek olsak fonksiyon bize eski kalem nesnesinin handle değerini vermektedir.

Çizim Nesnelerinin Yaratılması Ve Yok Edilmesi

Bir çizim nesnesi CreateXXX gibi bir API fonksiyonuyla yaratılmaktadır. Örneğin kalem yaratmak için CreatePen, fırça yaratmak için CreateSolidBrush gibi bir fonksiyon vardır. Bu fonksiyonlar çizim nesnelerinin handle değerleri ile geri dönerler. Tüm çizim nesneleri gerektiğinde aynı DeleteObject API fonksiyonuyla yok edilebilirler.

Çizim nesneleri ile DC nesnesinin aynı zamanda yaratılması zorunlu değildir. Örneğin kalemler ve fırçalar gibi çizim nesneleri programın başında WM_CREATE mesajında yaratılabilir. WM_PAINT mesajında SelectObject ile seçilebilir.

Kalem Nesnesinin Yaratılması

Kalem nesnesinin yaratılması için CreatePen API fonksiyonu kullanılabilir.

```
HPEN CreatePen(
  int fnPenStyle,              // pen style
  int nWidth,                  // pen width
  COLORREF crColor         // pen color
);
```

Fonksiyonun 1. parametresi yaratılacak kalemin çizgi biçimidir. Şu değerlerden biri olabilir:

Style	Description
PS_SOLID	Pen is solid.
PS_DASH	Pen is dashed. This style is valid only when the pen width is one or less in device units.
PS_DOT	Pen is dotted. This style is valid only when the pen width is one or less in device units.
PS_DASHDOT	Pen has alternating dashes and dots. This style is valid only when the pen width is one or less in device units.
PS_DASHDOTDOT	Pen has alternating dashes and double dots. This style is valid only when the pen width is one or less in device units.

PS_NULL	Pen is invisible.
PS_INSIDEFRAME	Pen is solid. When this pen is used in any graphics device interface (GDI) drawing function that takes a bounding rectangle, the dimensions of the figure are shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen. This applies only to geometric pens.

Fonksiyonun 2. parametresi kalemin kalınlık değeridir. '0', 1 pixel anlamına gelir. Fonksiyonun 3. parametresi kalemin rengini belirtir. Renk konusu, sonraki bölümde yer almaktadır. bak paintden_1.cpp(boyut değişince siyah çiziyor. WM_PAINT'e de gönder....), paintden_2.cpp, paintden_3.cpp

/* paintden_1.cpp */

```
#include <windows.h>
#include <list>
#include "resource.h"

using namespace std;

LRESULT CALLBACK WndProc(...);

void OnFileSave(const list<POINT> &pointList);
void OnFileOpen(list<POINT> &pointList);

int WINAPI WinMain(...)
{
	...
	if (!hPrevInstance) {
		...
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		...
		wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
		...
	}

	...
}
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static int prevX, prevY;
	static list<POINT> pointList;
	static HPEN hPen;
	POINT point;

	switch (message) {
		case WM_CREATE:
			hPen = CreatePen(PS_SOLID, 5, RGB(255, 0, 0));
			break;
		case WM_COMMAND:
			if (LOWORD(wParam) == ID_FILE_SAVE_MENU)
				OnFileSave(pointList);
			else if (LOWORD(wParam) == ID_FILE_OPEN_MENU) {
				OnFileOpen(pointList);
				InvalidateRect(hWnd, NULL, TRUE);
			}
			else if (LOWORD(wParam) == ID_FILE_CLEAR_MENU) {
				pointList.clear();
				InvalidateRect(hWnd, NULL, TRUE);
			}
			break;
```

```
		case WM_LBUTTONDOWN:
			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);
			break;

		case WM_MOUSEMOVE:
			if (!(wParam & MK_LBUTTON))
				break;
			point.x = prevX;
			point.y = prevY;
			pointList.push_back(point);

			hDC = GetDC(hWnd);
			SelectObject(hDC, hPen);
			MoveToEx(hDC, prevX, prevY, NULL);
			LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);

			point.x = prevX;
			point.y = prevY;

			pointList.push_back(point);

			ReleaseDC(hWnd, hDC);
			break;
		case WM_PAINT:
			{
				hDC = BeginPaint(hWnd, &ps);

				list<POINT>::iterator iter = pointList.begin();

				while (iter != pointList.end()) {
					MoveToEx(hDC, iter->x, iter->y, NULL);
					++iter;
					LineTo(hDC, iter->x, iter->y);
					++iter;
				}

				EndPaint(hWnd, &ps);

			}
			break;

		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}

void OnFileSave(const list<POINT> &pointList)
{
	FILE *f;

	if ((f = fopen("paint.dat", "wb")) == NULL) {
		MessageBox(NULL, "Cannot create file!..", "Error", MB_OK);
		return;
	}

	list<POINT>::const_iterator iter = pointList.begin();
```

```
	for (; iter != pointList.end(); ++iter) {
		fwrite(&*iter, 1, sizeof(POINT), f);
	}

	fclose(f);
}

void OnFileOpen(list<POINT> &pointList)
{
	FILE *f;

	if ((f = fopen("paint.dat", "rb")) == NULL) {
		MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
		return;
	}

	pointList.clear();

	POINT point;
	for (;;) {
		fread(&point, 1, sizeof(POINT), f);
		if (feof(f))
			break;
		pointList.push_back(point);
	}
}
```

/* paintden_2.cpp*/

```
#include <windows.h>
#include <list>
#include "resource.h"

using namespace std;

LRESULT CALLBACK WndProc(...);

void OnFileSave(const list<POINT> &pointList);
void OnFileOpen(list<POINT> &pointList);

int WINAPI WinMain(...)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		...
		wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
		...
	}

	...
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                                                  LPARAM lParam)
{
```

```
HDC hDC;
PAINTSTRUCT ps;
static int prevX, prevY;
static list<POINT> pointList;
static HPEN hPen;
POINT point;

switch (message) {
	case WM_CREATE:
		hPen = CreatePen(PS_SOLID, 5, RGB(255, 0, 0));
		break;
	case WM_COMMAND:
		if (LOWORD(wParam) == ID_FILE_SAVE_MENU)
			OnFileSave(pointList);
		else if (LOWORD(wParam) == ID_FILE_OPEN_MENU) {
			OnFileOpen(pointList);
			InvalidateRect(hWnd, NULL, TRUE);
		}
		else if (LOWORD(wParam) == ID_FILE_CLEAR_MENU) {
			pointList.clear();
			InvalidateRect(hWnd, NULL, TRUE);
		}
		break;
	case WM_LBUTTONDOWN:
		prevX = LOWORD(lParam);
		prevY = HIWORD(lParam);
		break;

	case WM_MOUSEMOVE:
		if (!(wParam & MK_LBUTTON))
			break;
		point.x = prevX;
		point.y = prevY;
		pointList.push_back(point);

		hDC = GetDC(hWnd);
		SelectObject(hDC, hPen);
		MoveToEx(hDC, prevX, prevY, NULL);
		LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
		prevX = LOWORD(lParam);
		prevY = HIWORD(lParam);

		point.x = prevX;
		point.y = prevY;

		pointList.push_back(point);

		ReleaseDC(hWnd, hDC);
		break;
	case WM_PAINT:
		{
			hDC = BeginPaint(hWnd, &ps);
			SelectObject(hDC, hPen);

			list<POINT>::iterator iter = pointList.begin();

			while (iter != pointList.end()) {
				MoveToEx(hDC, iter->x, iter->y, NULL);
				++iter;
				LineTo(hDC, iter->x, iter->y);
				++iter;
			}
```

```
                                EndPaint(hWnd, &ps);

                        }
                        break;

                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnFileSave(const list<POINT> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "wb")) == NULL) {
                MessageBox(NULL, "Cannot create file!..", "Error", MB_OK);
                return;
        }

        list<POINT>::const_iterator iter = pointList.begin();

        for (; iter != pointList.end(); ++iter) {
                fwrite(&*iter, 1, sizeof(POINT), f);
        }

        fclose(f);
}

void OnFileOpen(list<POINT> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "rb")) == NULL) {
                MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
                return;
        }

        pointList.clear();

        POINT point;
        for (;;) {
                fread(&point, 1, sizeof(POINT), f);
                if (feof(f))
                        break;
                pointList.push_back(point);
        }
}
```

/* paintden_3.cpp */

```
#include <windows.h>
#include <list>
#include "resource.h"

using namespace std;

LRESULT CALLBACK WndProc(...);
```

```
void OnFileSave(const list<POINT> &pointList);
void OnFileOpen(list<POINT> &pointList);

int WINAPI WinMain(...)
{
	...
	if (!hPrevInstance) {
		...
		wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
		...
	}

	...
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
                         LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static int prevX, prevY;
	static list<POINT> pointList;
	static HPEN hPen;
	POINT point;

	switch (message) {
		case WM_CREATE:
			hPen = CreatePen(PS_DOT, 1, RGB(255, 0, 0));
			break;
		case WM_COMMAND:
			if (LOWORD(wParam) == ID_FILE_SAVE_MENU)
				OnFileSave(pointList);
			else if (LOWORD(wParam) == ID_FILE_OPEN_MENU) {
				OnFileOpen(pointList);
				InvalidateRect(hWnd, NULL, TRUE);
			}
			else if (LOWORD(wParam) == ID_FILE_CLEAR_MENU) {
				pointList.clear();
				InvalidateRect(hWnd, NULL, TRUE);
			}
			break;
		case WM_LBUTTONDOWN:
			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);
			break;

		case WM_MOUSEMOVE:
			if (!(wParam & MK_LBUTTON))
				break;
			point.x = prevX;
			point.y = prevY;
			pointList.push_back(point);

			hDC = GetDC(hWnd);
			SelectObject(hDC, hPen);
			MoveToEx(hDC, prevX, prevY, NULL);
			LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);

			point.x = prevX;
			point.y = prevY;
```

```
                        pointList.push_back(point);

                        ReleaseDC(hWnd, hDC);
                        break;
                case WM_PAINT:
                        {
                                hDC = BeginPaint(hWnd, &ps);
                                SelectObject(hDC, hPen);

                                list<POINT>::iterator iter = pointList.begin();

                                while (iter != pointList.end()) {
                                        MoveToEx(hDC, iter->x, iter->y, NULL);
                                        ++iter;
                                        LineTo(hDC, iter->x, iter->y);
                                        ++iter;
                                }

                                EndPaint(hWnd, &ps);

                        }
                        break;

                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnFileSave(const list<POINT> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "wb")) == NULL) {
                MessageBox(NULL, "Cannot create file!..", "Error", MB_OK);
                return;
        }

        list<POINT>::const_iterator iter = pointList.begin();

        for (; iter != pointList.end(); ++iter) {
                fwrite(&*iter, 1, sizeof(POINT), f);
        }

        fclose(f);
}

void OnFileOpen(list<POINT> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "rb")) == NULL) {
                MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
                return;
        }

        pointList.clear();

        POINT point;
```

```
	for (;;) {
		fread(&point, 1, sizeof(POINT), f);
		if (feof(f))
			break;
		pointList.push_back(point);
	}
}
```

Renklerin Oluşturulması

Bugün kullandığımız RGB monitörlerde rengi oluşturan 3 elektron tabancası vardır. Bunlar kırmızı, yeşil ve mavi renk tabancalarıdır. Renkler, bu tabancaların aynı noktaya belli bir tonda ışımasıyla elde edilir. Renk tabancaları için 1 byte'lık açıklık koyuluk değerleri vardır. En koyu durum 0 ile, en açık durum 255 ile temsil edilir. Windows API fonksiyonları renk bilgilerini COLORREF isminde bir türle isterler.

typedef long COLORREF;

Renk bilgisi 4 byte içerisinde aşağıdaki gibi kodlanır:

3	2	1	0
-	B	G	R

R, G, B değerlerini alarak yukarıdaki gibi long bir tür içerisine kodlayan RGB isimli bir makro vardır. RGB makrosu aşağıdaki gibi yazılabilir:

```
#define		RGB	((COLORREF) ((r)) | (g<<8) | ((b)<<16)))
```

Özetle API fonksiyonları bizden rengi COLORREF türü ile ister. Biz de ona rengi RGB makrosuyla vermeliyiz.

Fırçaların Yaratılması

Fırça yaratan birkaç API fonksiyonu vardır. Desensiz düz bir fırça CreatSolidBrush fonksiyonu ile yaratılır.

```
HBRUSH CreateSolidBrush(
  COLORREF crColor   // brush color value
);
```

Fonksiyon, parametre olarak fırçanın rengini alır; fırçanın handle değeri ile geri döner. bak generic4.cpp, generic5.cpp

/* paintden_4.cpp */

```
#include <windows.h>
#include <list>
#include "resource.h"

using namespace std;

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnFileSave(const list<POINT> &pointList);
void OnFileOpen(list<POINT> &pointList);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
```

```
                    int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;
        static int prevX, prevY;
        static list<POINT> pointList;
        static HPEN hPen;
        POINT point;

        switch (message) {
                case WM_CREATE:
                        hPen = CreatePen(PS_SOLID, 10, RGB(0, 0, 0));
                        break;
                case WM_COMMAND:
                        switch (LOWORD(wParam)) {
                                case ID_FILE_SAVE_MENU:
                                        OnFileSave(pointList);
                                        break;
                                case ID_FILE_OPEN_MENU:
```

```
                        OnFileOpen(pointList);
                        InvalidateRect(hWnd, NULL, TRUE);
                        break;
                case ID_FILE_CLEAR_MENU:
                        pointList.clear();
                        InvalidateRect(hWnd, NULL, TRUE);
                        break;
                case ID_COLOR_BLACK:
                        DeleteObject(hPen);
                        hPen = CreatePen(PS_SOLID, 10, RGB(0, 0, 0));
                        break;
                case ID_COLOR_RED:
                        DeleteObject(hPen);
                        hPen = CreatePen(PS_SOLID, 10, RGB(255, 0, 0));
                        break;
                case ID_COLOR_BLUE:
                        DeleteObject(hPen);
                        hPen = CreatePen(PS_SOLID, 10, RGB(0, 0, 255));
                        break;
                case ID_COLOR_GREEN:
                        DeleteObject(hPen);
                        hPen = CreatePen(PS_SOLID, 10, RGB(0, 255, 0));
                        break;
                case ID_COLOR_CUSTOM:
                        {
                                CHOOSECOLOR cc = {sizeof(cc)};
                                COLORREF colors[16];
                                cc.hwndOwner = hWnd;
                                cc.hInstance = (HWND) GetWindowLong(hWnd,
                                                GWL_HINSTANCE);
                                cc.lpCustColors = colors;
                                ChooseColor(&cc);

                                DeleteObject(hPen);
                                hPen = CreatePen(PS_SOLID, 10, cc.rgbResult);
                        }
                        break;

        }
        break;
case WM_LBUTTONDOWN:
        prevX = LOWORD(lParam);
        prevY = HIWORD(lParam);
        break;

case WM_MOUSEMOVE:
        if (!(wParam & MK_LBUTTON))
                break;
        point.x = prevX;
        point.y = prevY;
        pointList.push_back(point);

        hDC = GetDC(hWnd);
        SelectObject(hDC, hPen);
        MoveToEx(hDC, prevX, prevY, NULL);
        LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
        prevX = LOWORD(lParam);
        prevY = HIWORD(lParam);

        point.x = prevX;
        point.y = prevY;

        pointList.push_back(point);
```

```
                        ReleaseDC(hWnd, hDC);
                        break;
                case WM_PAINT:
                        {
                                hDC = BeginPaint(hWnd, &ps);
                                SelectObject(hDC, hPen);
                                list<POINT>::iterator iter = pointList.begin();

                                while (iter != pointList.end()) {
                                        MoveToEx(hDC, iter->x, iter->y, NULL);
                                        ++iter;
                                        LineTo(hDC, iter->x, iter->y);
                                        ++iter;
                                }

                                EndPaint(hWnd, &ps);

                        }
                        break;

                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnFileSave(const list<POINT> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "wb")) == NULL) {
                MessageBox(NULL, "Cannot create file!..", "Error", MB_OK);
                return;
        }

        list<POINT>::const_iterator iter = pointList.begin();

        for (; iter != pointList.end(); ++iter) {
                fwrite(&*iter, 1, sizeof(POINT), f);
        }

        fclose(f);
}

void OnFileOpen(list<POINT> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "rb")) == NULL) {
                MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
                return;
        }

        pointList.clear();

        POINT point;
        for (;;) {
                fread(&point, 1, sizeof(POINT), f);
                if (feof(f))
```

```
                        break;
                pointList.push_back(point);
        }
}
```

/* paintden_5.cpp

```
#include <windows.h>
#include <list>
#include "resource.h"

using namespace std;

typedef struct tagPOINT_ELEM {
        POINT point;
        COLORREF color;
} POINT_ELEM;

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnFileSave(const list<POINT_ELEM> &pointList);
void OnFileOpen(list<POINT_ELEM> &pointList);


int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                        int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
```

```
			TranslateMessage(&message);
			DispatchMessage(&message);
		}
		return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
		HDC hDC;
		PAINTSTRUCT ps;
		static int prevX, prevY;
		static list<POINT_ELEM> pointList;
		static HPEN hPen;
		static COLORREF curColor;
		POINT point;

		switch (message) {
			case WM_CREATE:
				curColor = RGB(0, 0, 0);
				hPen = CreatePen(PS_SOLID, 10, curColor);
				break;
			case WM_COMMAND:
				switch (LOWORD(wParam)) {
					case ID_FILE_SAVE_MENU:
						OnFileSave(pointList);
						break;
					case ID_FILE_OPEN_MENU:
						OnFileOpen(pointList);
						InvalidateRect(hWnd, NULL, TRUE);
						break;
					case ID_FILE_CLEAR_MENU:
						pointList.clear();
						InvalidateRect(hWnd, NULL, TRUE);
						break;
					case ID_COLOR_BLACK:
						DeleteObject(hPen);
						curColor = RGB(0, 0, 0);
						hPen = CreatePen(PS_SOLID, 10, curColor);
						break;
					case ID_COLOR_RED:
						DeleteObject(hPen);
						curColor = RGB(255, 0, 0);
						hPen = CreatePen(PS_SOLID, 10, curColor);
						break;
					case ID_COLOR_BLUE:
						DeleteObject(hPen);
						curColor = RGB(0, 0, 255);
						hPen = CreatePen(PS_SOLID, 10, curColor);
						break;
					case ID_COLOR_GREEN:
						DeleteObject(hPen);
						curColor = RGB(0, 255, 0);
						hPen = CreatePen(PS_SOLID, 10, curColor);
						break;
					case ID_COLOR_CUSTOM:
						{
							CHOOSECOLOR cc = {sizeof(cc)};
							COLORREF colors[16];
							cc.hwndOwner = hWnd;
							cc.hInstance = (HWND) GetWindowLong(hWnd,
									GWL_HINSTANCE);
							cc.lpCustColors = colors;
							ChooseColor(&cc);
```

```
                                DeleteObject(hPen);
                                hPen = CreatePen(PS_SOLID, 10, cc.rgbResult);
                        }
                        break;
                }
                break;
        case WM_LBUTTONDOWN:
                prevX = LOWORD(lParam);
                prevY = HIWORD(lParam);
                break;

        case WM_MOUSEMOVE:
                {
                        if (!(wParam & MK_LBUTTON))
                                break;
                        point.x = prevX;
                        point.y = prevY;

                        POINT_ELEM pointElem;
                        pointElem.point = point;
                        pointElem.color = curColor;
                        pointList.push_back(pointElem);

                        hDC = GetDC(hWnd);
                        SelectObject(hDC, hPen);
                        MoveToEx(hDC, prevX, prevY, NULL);
                        LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);

                        point.x = prevX;
                        point.y = prevY;

                        pointElem.point = point;
                        pointElem.color = curColor;
                        pointList.push_back(pointElem);

                        ReleaseDC(hWnd, hDC);
                }
                        break;
        case WM_PAINT:
                {
                        HPEN hPen;
                        hDC = BeginPaint(hWnd, &ps);

                        list<POINT_ELEM>::iterator iter = pointList.begin();

                        while (iter != pointList.end()) {
                                hPen = CreatePen(PS_SOLID, 10, iter->color);
                                SelectObject(hDC, hPen);
                                MoveToEx(hDC, iter->point.x, iter->point.y, NULL);
                                ++iter;
                                LineTo(hDC, iter->point.x, iter->point.y);
                                ++iter;
                                DeleteObject(hPen);
                        }

                        EndPaint(hWnd, &ps);

                }
                break;
```

```
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnFileSave(const list<POINT_ELEM> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "wb")) == NULL) {
                MessageBox(NULL, "Cannot create file!..", "Error", MB_OK);
                return;
        }

        list<POINT_ELEM>::const_iterator iter = pointList.begin();

        for (; iter != pointList.end(); ++iter) {
                fwrite(&*iter, 1, sizeof(POINT_ELEM), f);
        }

        fclose(f);
}

void OnFileOpen(list<POINT_ELEM> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "rb")) == NULL) {
                MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
                return;
        }

        pointList.clear();

        POINT_ELEM pointElem;
        for (;;) {
                fread(&pointElem.point, 1, sizeof(POINT_ELEM), f);
                if (feof(f))
                        break;
                pointList.push_back(pointElem);
        }
}
```

Dikdörtgen Çizen Fonksiyonlar

Dikdörtgenin çizgileri kalem ile çizilir, içi fırça ile boyanır. Dikdörtgen çizmek için rectangle fonksiyonu kullanılır. Prototipi şöyledir:

```
BOOL Rectangle(
 HDC hdc,              // handle to DC
 int nLeftRect,        // x-coord of upper-left corner of rectangle
 int nTopRect,         // y-coord of upper-left corner of rectangle
 int nRightRect,       // x-coord of lower-right corner of rectangle
 int  nBottomRect      // y-coord of lower-right corner of rectangle
);
```

Fonksiyonun 1. parametresi handle değeri; diğer parametreleri sol üst ve alt köşenin koordinatlarıdır.

```
#include <windows.h>
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(...)
{
        ...
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;
        static HPEN hPen;
        static HBRUSH hBrush;

        switch (message) {
                case WM_CREATE:
                        hPen = CreatePen(PS_SOLID, 5, RGB(255, 0, 0));
                        hBrush = CreateSolidBrush(RGB(0, 255, 0));
                        break;
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);

                        SelectObject(hDC, hPen);
                        SelectObject(hDC, hBrush);
                        Rectangle(hDC, 100, 100, 200, 200);

                        EndPaint(hWnd, &ps);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

FillRect fonksiyonu yalnızca dikdörtgenin içini boyar; sınır çizgilerini çizmez.

```
int FillRect(
 HDC hDc,                // handle to device context
 CONST RECT *lprc,       // pointer to structure with rectangle
 HBRUSH hbr,             // handle to brush
);
```

Fonksiyon ayrıca dikdörtgenin içinin boyanacağı fırçayı parametre olarak almaktadır. Yani fonksiyon DC'deki fırçayı değil parametre olarak verilen fırçayı kullanır.

Kalemler Ve Fırçalar

azı kalem ve fırça nesneleri zaten yaratılmış olarak Windows'ta bulunmaktadır. GetStockObject API fonksiyonu ile bunların doğrudan handle değerleri alınabilir.

```
HGDIOBJ GetStockObject(
 int fnObject   // type of stock object
);
```

Fonksiyonun parametresi stoktaki hangi nesnenin alınacağını belirtir ve şunlardan biri olabilir:

Value	Meaning
BLACK_BRUSH	Black brush.
DKGRAY_BRUSH	Dark gray brush.
DC_BRUSH	**Windows 98, Windows NT 5.0 and later:** Solid color brush. The default color is white. The color can be changed by using the **SetDCBrushColor** function. For more information, see the following Remarks section.
GRAY_BRUSH	Gray brush.
HOLLOW_BRUSH	Hollow brush (equivalent to NULL_BRUSH).
LTGRAY_BRUSH	Light gray brush.
NULL_BRUSH	Null brush (equivalent to HOLLOW_BRUSH).
WHITE_BRUSH	White brush.
BLACK_PEN	Black pen.
DC_PEN	**Windows 98, Windows NT 5.0 and later:** Solid pen color. The default color is white. The color can be changed by using the **SetDCPenColor** function. For more information, see the following Remarks section.
WHITE_PEN	White pen.

Şekillerin Sürüklenmesi

Bir şekli silmek demek onu zemin rengiyle yeniden çizmek demektedir. Şeklin üstüne click yaparak şekli sürükleme algoritması oldukça basittir. WM_MOUSEMOVE mesajında farenin x ekseninde ve y ekseninde ne kadar ötelendiği hesaplanır; şekil de o kadar ötelenir. Windows'ta bu işlemleri biraz daha kolaylaştırmak için RECT yapısı üzerinde işlem yapan bazı API fonksiyonları bulundurulmuştur. Bu fonksiyonlar şunlardır:

PtInRect: Bu fonksiyonun prototipi aşağıdaki gibidir:

```
BOOL PtInRect(
  CONST RECT *lprc,      // pointer to structure with rectangle
  POINT pt               // structure with point
);
```

Bu fonksiyon, bir noktanın dikdörtgen içerisinde olup olmadığını tespit etmek için kullanılır.

IntersectRect: Bu fonksiyonun prototipi aşağıdaki gibidir:

```
BOOL IntersectRect(
  LPRECT lprcDst,          // pointer to structure for intersection
  CONST RECT *lprcSrc1,    // pointer to structure with first rectangle
  CONST RECT *lprcSrc2     // pointer to structure with second rectangle
);
```

Bu fonksiyon, iki dikdörtgenin kesişiminden elde edilen dikdörtgeni bulmakta kullanılır.

OffsetRect: Bu fonksiyonun prototipi aşağıdaki gibidir:

```
BOOL OffsetRect(
  LPRECT lprc,  // pointer to structure with rectangle
  int dx,       // horizontal offset
```

```
  int dy        // vertical offset
);
```

Bu fonksiyon, bir dikdörtgenin koordinatlarını x, y kadar öteler.

InflateRect: Bu fonksiyonun prototipi aşağıdaki gibidir:

```
BOOL InflateRect(
  LPRECT lprc,        // pointer to rectangle
  int dx,             // amount to increase or decrease width
  int dy              // amount to increase or decrease height
);
```

Bu fonksiyon, bir dikdörtgeni iki köşesinden açmak yada büzmek için kullanılır. bak: painttest1.c

/* painttest1.c */

```
#include <windows.h>
...
int WINAPI WinMain(...)
{
        ...
        if (!hPrevInstance) {
                wndClass.style = 0;/*CS_HREDRAW | CS_VREDRAW;*/
                ...
        }

        ...
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;
        static HPEN hPen;
        static HBRUSH hBrush;
        RECT rect;

        switch (message) {
                case WM_CREATE:
                        hPen = CreatePen(PS_SOLID, 5, RGB(255, 0, 0));
                        hBrush = CreateSolidBrush(RGB(0, 255, 0));
                        break;
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);
                        GetClientRect(hWnd, &rect);
                        InflateRect(&rect, -30, -30);

                        SelectObject(hDC, hPen);
                        SelectObject(hDC, hBrush);
                        Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

                        EndPaint(hWnd, &ps);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
```

```
        }
        return 0;
}
```

UnionRect: Bu fonksiyonun prototipi aşağıdaki gibidir:

```
BOOL UnionRect(
  LPRECT lprcDst,             // pointer to structure for union
  CONST RECT *lprcSrc1,  // pointer to structure with first rectangle
  CONST RECT *lprcSrc2   // pointer to structure with second rectangle
);
```

Bu fonksiyon, iki dikdörtgeni içeren en küçük dikdörtgeni bulmaktadır. bak painttest2.c, painttest3.c (hareket edilince zemin rengi korunmuyor), painttest4.c,

/* painttest2.c*/

```
....
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;
        static RECT rect = {100, 100, 200, 200};
        static int prevX, prevY;
        static int deltaX, deltaY;
        static BOOL bFlag = FALSE;
        static HBRUSH hWhiteBrush;
        static HPEN hWhitePen;

        switch (message) {
                case WM_CREATE:
                        hWhiteBrush = GetStockObject(WHITE_BRUSH);
                        hWhitePen = GetStockObject(WHITE_PEN);

                        break;
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);

                        Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

                        EndPaint(hWnd, &ps);
                        break;
                case WM_LBUTTONDOWN:
                        {
                                POINT point;

                                prevX = point.x = LOWORD(lParam);
                                prevY = point.y = HIWORD(lParam);
                                if (PtInRect(&rect, point))
                                        bFlag = TRUE;
                        }
                        break;

                case WM_LBUTTONUP:
                        bFlag = FALSE;
                        break;
                case WM_MOUSEMOVE:
                        {
                                HPEN hPrevPen;
                                HBRUSH hPrevBrush;
```

```
                                HDC hDC;

                                if (!bFlag)
                                        break;

                                hDC = GetDC(hWnd);
                                hPrevPen = SelectObject(hDC, hWhitePen);
                                hPrevBrush = SelectObject(hDC, hWhiteBrush);

                                Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

                                deltaX = LOWORD(lParam) - prevX;
                                deltaY = HIWORD(lParam) - prevY;
                                OffsetRect(&rect, deltaX, deltaY);

                                SelectObject(hDC, hPrevPen);
                                SelectObject(hDC, hPrevBrush);
                                Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

                                prevX = LOWORD(lParam);
                                prevY = HIWORD(lParam);

                                ReleaseDC(hWnd, hDC);
                        }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

/* painttest3.c */

```
#include <windows.h>


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                        int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) CreateSolidBrush(RGB(255, 0, 0));
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
```

```
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static RECT rect = {100, 100, 200, 200};
	static int prevX, prevY;
	static int deltaX, deltaY;
	static BOOL bFlag = FALSE;
	static HBRUSH hWhiteBrush;
	static HPEN hWhitePen;

	switch (message) {
		case WM_CREATE:
			hWhiteBrush = GetStockObject(WHITE_BRUSH);
			hWhitePen = GetStockObject(WHITE_PEN);

			break;
		case WM_PAINT:
			hDC = BeginPaint(hWnd, &ps);

			Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

			EndPaint(hWnd, &ps);
			break;
		case WM_LBUTTONDOWN:
			{
				POINT point;

				prevX = point.x = LOWORD(lParam);
				prevY = point.y = HIWORD(lParam);
				if (PtInRect(&rect, point))
					bFlag = TRUE;
			}
			break;

		case WM_LBUTTONUP:
			bFlag = FALSE;
			break;
```

```
		case WM_MOUSEMOVE:
			{
				HPEN hPrevPen;
				HBRUSH hPrevBrush;
				HDC hDC;

				if (!bFlag)
					break;

				hDC = GetDC(hWnd);
				hPrevPen = SelectObject(hDC, hWhitePen);
				hPrevBrush = SelectObject(hDC, hWhiteBrush);

				Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

				deltaX = LOWORD(lParam) - prevX;
				deltaY = HIWORD(lParam) - prevY;
				OffsetRect(&rect, deltaX, deltaY);

				SelectObject(hDC, hPrevPen);
				SelectObject(hDC, hPrevBrush);
				Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

				prevX = LOWORD(lParam);
				prevY = HIWORD(lParam);

				ReleaseDC(hWnd, hDC);
			}
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

/* painttest4.c */

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) CreateSolidBrush(RGB(255, 0, 0));
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
```

```
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static RECT rect = {100, 100, 200, 200};
	static int prevX, prevY;
	static int deltaX, deltaY;
	static BOOL bFlag = FALSE;
	static HBRUSH hWhiteBrush, hRedBrush;
	static HPEN hWhitePen, hRedPen;

	switch (message) {
		case WM_CREATE:
			hWhiteBrush = GetStockObject(WHITE_BRUSH);
			hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
			hWhitePen = GetStockObject(WHITE_PEN);
			hRedPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
			break;
		case WM_PAINT:
			hDC = BeginPaint(hWnd, &ps);

			Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

			EndPaint(hWnd, &ps);
			break;
		case WM_LBUTTONDOWN:
			{
				POINT point;

				prevX = point.x = LOWORD(lParam);
				prevY = point.y = HIWORD(lParam);
				if (PtInRect(&rect, point))
					bFlag = TRUE;
			}
			break;
```

```
		case WM_LBUTTONUP:
			bFlag = FALSE;
			break;
		case WM_MOUSEMOVE:
			{
				HPEN hPrevPen;
				HBRUSH hPrevBrush;
				HDC hDC;

				if (!bFlag)
					break;

				hDC = GetDC(hWnd);
				hPrevPen = SelectObject(hDC, hRedPen);
				hPrevBrush = SelectObject(hDC, hRedBrush);

				Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

				deltaX = LOWORD(lParam) - prevX;
				deltaY = HIWORD(lParam) - prevY;
				OffsetRect(&rect, deltaX, deltaY);

				SelectObject(hDC, hPrevPen);
				SelectObject(hDC, hPrevBrush);
				Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

				prevX = LOWORD(lParam);
				prevY = HIWORD(lParam);

				ReleaseDC(hWnd, hDC);
			}
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

Pencerenin Zemin Renginin Oluşturulması ve Değiştirilmesi

Zeminin boyanması işlemi aslında BeginPaint fonksiyonu tarafından yapılmaktadır. BeginPaint kendi içerisinde SendMessage ile **WM_ERASEBKGND** mesajını pencereye yollar. WM_ERASEBKGND mesajı için DefWindowProc fonksiyonu WNDCLASS yapısının hbrBackground elamanıyla belirtilen fırçayı alarak FillRect gibi bir fonksiyonla güncelleme alanını boyar.

BeginPaint $\overset{\text{send}}{\rightarrow}$ WM_ERASEBKGND $\rightarrow$ DefWindowProc $\rightarrow$ WNDCLASS hbrBackground ile güncelleme alanını boyar.

Özetle:

1. Zeminin silinmesi sürecini başlatan fonksiyon BeginPaint fonksiyonudur.
2. Zeminin silinmesi işlemini gerçekleştiren DefWindowProc fonksiyonu yani WM_ERASEBKGND mesajıdır. Biz WM_ERASEBKGND mesajını DefWindowProc fonksiyonuna bırakmazsak zemin silinmez.

3. Sonuçta zeminin rengi WNDCLASS yapısının hbrBackground elemanıyla belirlenmektedir. İskelet Windows programında stoktan beyaz fırça alınarak bu elemana yerleştirilmiştir. bak painttest5.c - > zemin boyanmadı. bak painttest6.c,

/* painttest5.c */

```
#include <windows.h>

...
int WINAPI WinMain(...)
{
	...
	if (!hPrevInstance) {
		...
		wndClass.hbrBackground = (HBRUSH) CreateSolidBrush(RGB(255, 0, 0));
		...
	}

	...
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static RECT rect = {100, 100, 200, 200};
	static int prevX, prevY;
	static int deltaX, deltaY;
	static BOOL bFlag = FALSE;
	static HBRUSH hWhiteBrush, hRedBrush;
	static HPEN hWhitePen, hRedPen;

	switch (message) {
		case WM_CREATE:
			hWhiteBrush = GetStockObject(WHITE_BRUSH);
			hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
			hWhitePen = GetStockObject(WHITE_PEN);
			hRedPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
			break;

		case WM_PAINT:
			hDC = BeginPaint(hWnd, &ps);

			Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

			EndPaint(hWnd, &ps);
			break;
		case WM_LBUTTONDOWN:
			{
				POINT point;

				prevX = point.x = LOWORD(lParam);
				prevY = point.y = HIWORD(lParam);
				if (PtInRect(&rect, point))
					bFlag = TRUE;
			}
			break;

		case WM_LBUTTONUP:
			bFlag = FALSE;
			break;
		case WM_MOUSEMOVE:
```

```
                {
                        HPEN hPrevPen;
                        HBRUSH hPrevBrush;
                        HDC hDC;

                        if (!bFlag)
                                break;

                        hDC = GetDC(hWnd);
                        hPrevPen = SelectObject(hDC, hRedPen);
                        hPrevBrush = SelectObject(hDC, hRedBrush);

                        Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

                        deltaX = LOWORD(lParam) - prevX;
                        deltaY = HIWORD(lParam) - prevY;
                        OffsetRect(&rect, deltaX, deltaY);

                        SelectObject(hDC, hPrevPen);
                        SelectObject(hDC, hPrevBrush);
                        Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);

                        ReleaseDC(hWnd, hDC);
                }
                break;
        case WM_ERASEBKGND:
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

/* painttest6.c */

```
#include <windows.h>

...
int WINAPI WinMain(...)
{
    ...
    if (!hPrevInstance) {
        ...
        wndClass.hbrBackground = (HBRUSH) CreateSolidBrush(RGB(255, 0, 0));
        ...
    }

    ...
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    HDC hDC;
    PAINTSTRUCT ps;
    static RECT rect = {100, 100, 200, 200};
```

```
static int prevX, prevY;
static int deltaX, deltaY;
static BOOL bFlag = FALSE;
static HBRUSH hWhiteBrush, hRedBrush;
static HPEN hWhitePen, hRedPen;

switch (message) {
	case WM_CREATE:
		hWhiteBrush = GetStockObject(WHITE_BRUSH);
		hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
		hWhitePen = GetStockObject(WHITE_PEN);
		hRedPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
		break;

	case WM_PAINT:
		hDC = BeginPaint(hWnd, &ps);

		Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

		EndPaint(hWnd, &ps);
		break;
	case WM_LBUTTONDOWN:
		{
			POINT point;

			prevX = point.x = LOWORD(lParam);
			prevY = point.y = HIWORD(lParam);
			if (PtInRect(&rect, point))
				bFlag = TRUE;
		}
		break;

	case WM_LBUTTONUP:
		bFlag = FALSE;
		break;
	case WM_MOUSEMOVE:
		{
			HPEN hPrevPen;
			HBRUSH hPrevBrush;
			HDC hDC;

			if (!bFlag)
				break;

			hDC = GetDC(hWnd);
			hPrevPen = SelectObject(hDC, hRedPen);
			hPrevBrush = SelectObject(hDC, hRedBrush);

			Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

			deltaX = LOWORD(lParam) - prevX;
			deltaY = HIWORD(lParam) - prevY;
			OffsetRect(&rect, deltaX, deltaY);

			SelectObject(hDC, hPrevPen);
			SelectObject(hDC, hPrevBrush);
			Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);

			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);

			ReleaseDC(hWnd, hDC);
		}
```

```
                break;
        case WM_ERASEBKGND:
                {
                        RECT rect;
                        HBRUSH hBrush;
                        HDC hDC = (HDC) wParam;

                        GetUpdateRect(hWnd, &rect, FALSE);
                        hBrush = (HBRUSH) GetClassLong(hWnd, GCL_HBRBACKGROUND);

                        FillRect(hDC, &rect, hBrush);

                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

DefWindowProc fonksiyonunun WM_ERASEBKGND mesajında yaptığı şey şöyledir: (painttest7.c)

```
case WM_ERASEBKGND:
    {
        RECT rect;
        HBRUSH hBrush;
        HDC hDC = (HDC) wParam;

        GetUpdateRect(hWnd, &rect, FALSE);
        hBrush = (HBRUSH) GetClassLong(hWnd, GCL_HBRBACKGROUND);

        FillRect(hDC, &rect, hBrush);
        InflateRect(&rect, -50, -50);
        FillRect(hDC, &rect, hWhiteBrush);

    }
    break;
```

GetClassLong ve SetClassLong Fonksiyonları

GetClassLong ve SetClassLong fonksiyonları, pencerenin handle değerinden hareketle o pencerenin yaratılmasında kullanılan WNDLASS yapısının elemanlarını almak ve elemanları set etmek için kullanılır.

```
GetClassLong(
 HWND  hwnd,        //handle of window
 int nIndex         // offset of value to retrieve
);
```

Fonksiyonun 1. parametresi pencerenin handle değeri, 2. parametresi ise WNDCLASS yapısının hangi elemanının alınacağını belirten sembolik değerdir. Şu değerlerden birisi olabilir:

Value	Action

GCW_ATOM	Retrieves an **ATOM** value that uniquely identifies the window class. This is the same atom that the **RegisterClassEx** function returns.
GCL_CBCLSEXTRA	Retrieves the size, in bytes, of the extra memory associated with the class.
GCL_CBWNDEXTRA	Retrieves the size, in bytes, of the extra window memory associated with each window in the class. For information on how to access this memory, see **GetWindowLong**.
GCL_HBRBACKGROUND	Retrieves the handle of the background brush associated with the class.
GCL_HCURSOR	Retrieves the handle of the cursor associated with the class.
GCL_HICON	Retrieves the handle of the icon associated with the class.
GCL_HICONSM	Retrieves the handle of the small icon associated with the class.
GCL_HMODULE	Retrieves the handle of the module that registered the class.
GCL_MENUNAME	Retrieves the address of the menu name string. The string identifies the menu resource associated with the class.
GCL_STYLE	Retrieves the window-class style bits.
GCL_WNDPROC	Retrieves the address of the window procedure associated with the class.

Fonksiyon, elemanın değerini DWORD bir bilgiymiş gibi vermektedir. SetClassLong fonksiyonu ise şöyledir:

```
SetClassLong(
 HWND  hWnd,              // handle of window
 int nIndex,              // index of value to change
 LONG  dwNewLong          // new value
);
```

Fonksiyonun 1, parametresi pencerenin handle değeri, 2. parametresi hangi elemanın değiştirileceğini belirten sembolik değerdir. 3. parametre, değiştirilecek elemanın yeni değeridir. Fonksiyon, elemanın eski değeri ile geri döner. Şüphesiz SetClassLong fonksiyonu ile sınıftaki fırçanın değiştirilmesi rengin hemen değişmesi için yeterli değildir. Zeminin boyanması WM_PAINT mesajında yapıldığına göre InvalidateRect uygulanması gerekir.

bak: painttest_cpp

```
#include <windows.h>
#include <list>
#include "resource.h"

using namespace std;

typedef struct tagPOINT_ELEM {
        POINT point;
        COLORREF color;
} POINT_ELEM;

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnFileSave(const list<POINT_ELEM> &pointList);
void OnFileOpen(list<POINT_ELEM> &pointList);


int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
```

```
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static int prevX, prevY;
	static list<POINT_ELEM> pointList;
	static HPEN hPen;
	static COLORREF curColor;
	POINT point;

	switch (message) {
		case WM_CREATE:
			curColor = RGB(0, 0, 0);
			hPen = CreatePen(PS_SOLID, 10, curColor);
			break;
		case WM_COMMAND:
			switch (LOWORD(wParam)) {
				case ID_FILE_SAVE_MENU:
					OnFileSave(pointList);
					break;
				case ID_FILE_OPEN_MENU:
```

```
                    OnFileOpen(pointList);
                    InvalidateRect(hWnd, NULL, TRUE);
                    break;
                case ID_FILE_CLEAR_MENU:
                    pointList.clear();
                    InvalidateRect(hWnd, NULL, TRUE);
                    break;
                case ID_COLOR_BLACK:
                    DeleteObject(hPen);
                    curColor = RGB(0, 0, 0);
                    hPen = CreatePen(PS_SOLID, 10, curColor);
                    break;
                case ID_COLOR_RED:
                    DeleteObject(hPen);
                    curColor = RGB(255, 0, 0);
                    hPen = CreatePen(PS_SOLID, 10, curColor);
                    break;
                case ID_COLOR_BLUE:
                    DeleteObject(hPen);
                    curColor = RGB(0, 0, 255);
                    hPen = CreatePen(PS_SOLID, 10, curColor);
                    break;
                case ID_COLOR_GREEN:
                    DeleteObject(hPen);
                    curColor = RGB(0, 255, 0);
                    hPen = CreatePen(PS_SOLID, 10, curColor);
                    break;
                case ID_COLOR_CUSTOM:
                  {
                     CHOOSECOLOR cc = {sizeof(cc)};
                     COLORREF colors[16];
                     cc.hwndOwner = hWnd;
                     cc.hInstance = (HWND) GetWindowLong(hWnd,GWL_HINSTANCE);
                     cc.lpCustColors = colors;
                     ChooseColor(&cc);

                     DeleteObject(hPen);
                     hPen = CreatePen(PS_SOLID, 10, cc.rgbResult);
                  }
                  break;
                case ID_BACKCOLOR_CUSTOM:
                   {
                     CHOOSECOLOR cc = {sizeof(cc)};
                     HBRUSH hBrush;
                     COLORREF colors[16];

                     cc.hwndOwner = hWnd;
                     cc.hInstance = (HWND) GetWindowLong(hWnd,GWL_HINSTANCE);
                     cc.lpCustColors = colors;
                     ChooseColor(&cc);

                     hBrush = CreateSolidBrush(cc.rgbResult);
                     DeleteObject((HBRUSH) SetClassLong(hWnd,
                                        GCL_HBRBACKGROUND, (LONG) hBrush));
                     InvalidateRect(hWnd, NULL, TRUE);
                  }
                  break;
            }
            break;
        case WM_LBUTTONDOWN:
            prevX = LOWORD(lParam);
            prevY = HIWORD(lParam);
            break;
```

```
		case WM_MOUSEMOVE:
			{
				if (!(wParam & MK_LBUTTON))
					break;
				point.x = prevX;
				point.y = prevY;

				POINT_ELEM pointElem;
				pointElem.point = point;
				pointElem.color = curColor;
				pointList.push_back(pointElem);

				hDC = GetDC(hWnd);
				SelectObject(hDC, hPen);
				MoveToEx(hDC, prevX, prevY, NULL);
				LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
				prevX = LOWORD(lParam);
				prevY = HIWORD(lParam);

				point.x = prevX;
				point.y = prevY;

				pointElem.point = point;
				pointElem.color = curColor;
				pointList.push_back(pointElem);

				ReleaseDC(hWnd, hDC);
			}
				break;
		case WM_PAINT:
			{
				HPEN hPen;
				hDC = BeginPaint(hWnd, &ps);

				list<POINT_ELEM>::iterator iter = pointList.begin();

				while (iter != pointList.end()) {
					hPen = CreatePen(PS_SOLID, 10, iter->color);
					SelectObject(hDC, hPen);
					MoveToEx(hDC, iter->point.x, iter->point.y, NULL);
					++iter;
					LineTo(hDC, iter->point.x, iter->point.y);
					++iter;
					DeleteObject(hPen);
				}

				EndPaint(hWnd, &ps);

			}
			break;

		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}

void OnFileSave(const list<POINT_ELEM> &pointList)
{
```

```
	FILE *f;

	if ((f = fopen("paint.dat", "wb")) == NULL) {
		MessageBox(NULL, "Cannot create file!..", "Error", MB_OK);
		return;
	}

	list<POINT_ELEM>::const_iterator iter = pointList.begin();

	for (; iter != pointList.end(); ++iter) {
		fwrite(&*iter, 1, sizeof(POINT_ELEM), f);
	}

	fclose(f);
}

void OnFileOpen(list<POINT_ELEM> &pointList)
{
	FILE *f;

	if ((f = fopen("paint.dat", "rb")) == NULL) {
		MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
		return;
	}

	pointList.clear();

	POINT_ELEM pointElem;
	for (;;) {
		fread(&pointElem.point, 1, sizeof(POINT_ELEM), f);
		if (feof(f))
			break;
		pointList.push_back(pointElem);
	}
}
```

GetWindowLong ve SetWindowLong Fonksiyonları

CreateWindow fonksiyonu ile yaratılan pencerelerin bu fonksiyonda kullanılan yaratım parametreleri daha sonra GetWindowLong fonksiyonu ile alınabilir ve SetWindowLong fonksiyonu ile set edilebilir. Prototipleri:

```
LONG GetWindowLong (
 HWND hWnd,        // handle of window
 int nIndex        // offset of value to retrieve
);
```

fonksiyonu 1. parametresi pencerenin handle değeri, 2. parametresi pencerenin hangi parametrik özelliğinin alınacağını beliren bir sayıdır. Şunlardan biri olabilir:

GWL_EXSTYLE
GWL_STYLE
GWL_WNDPROC
GWL_HINSTANCE
GWL_HWNDPARENT
GWL_ID
GWL_USERDATA

Örneğin herhangi bir zaman biz hInstance değerini şöyle elde edebiliriz:

hInstance = (HINSTANCE) GetWindowLong(hWnd, GWL_HINSTANCE);

SetWindowLong fonksiyonu da benzer biçimdedir:

LONG SetWindowLong (
HWND hWnd, // handle of window
int nIndex, // offset of value to retrieve
LONG dwNewLong // new value
);

Fonksiyonun 1. parametresi pencerenin handle değeri, 2. parametresi pencerenin hangi özelliğinin değiştirileceği, 3. parametresi ise bu özelliğin yeni değeridir. Fonksiyon, bu özelliğ,in eski değerine geri dönmektedir.

Sınıf Çalışması: Menüden ilgili eleman seçildiğinde pencerenin genişletilebilir özelliğini kaldırınız. Yanıt: Bu İşlem şöyle yapılabilir:

```
SetWindowLong(hWnd, GWL_STYLE, GetWindowLong(hWnd, GWL_STYLE) & ~WS_THICKFRAME);
```

Diğer Önemli Çizim Fonksiyonları

Ellipse fonksiyonu, bir elips şeklini, çizgilerini DC'deki kalemle çizer; içini DC'deki fırçayla boyar. Fonksiyon parametre olarak elipsi sarmalayan dikdörtgenin koordinatlarını alır. Daire, elipsin özel bir durumudur. Bı nedenle Circle gibi bir API fonksiyonu yoktur. Sarmalayan dikdörtgen kare olursa elips de daire olur.

BOOL Ellipse(
HDC hdc, // handle to DC
int nLeftRect, // x-coord of upper-left corner of rectangle
int nTopRect, // y-coord ...
int nRightRect, // .....
int nBottomRect
);

bak: main.cpp

```
#include <windows.h>


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
```

```
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HBRUSH hBrush;
	static HPEN hPen;
	PAINTSTRUCT ps;
	HDC hDC;

	switch (message) {
		case WM_CREATE:
			hBrush = CreateSolidBrush(RGB(255, 0, 0));
			hPen = CreatePen(PS_SOLID, 3, RGB(0, 0, 255));
			break;
		case WM_PAINT:
			hDC = BeginPaint(hWnd, &ps);

			SelectObject(hDC, hBrush);
			SelectObject(hDC, hPen);

			Ellipse(hDC, 100, 100, 500, 300);

			EndPaint(hWnd, &ps);
			break;

		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

Bir noktadan başlayarak n tane noktadan oluşan düz çizgi kümesi, PolyLine fonksiyonu ile oluşturulabilir. Prototipi:

```
BOOL PolyLine(
 HDC hdc,
 CONST        POINT
*lppt,
 int cPoints
);
```

Fonksiyonun 1. parametresi DC, 2. parametresi noktaları tutan POINT türünden yapı dizisinin adresidir. Son parametre, bu dizinin uzunluğudur.

```
case WM_PAINT:     // elips etrafında dikdörtgen çizen kod parçası
        {
                PAINTSTRUCT ps;
                HDC hDC;
                POINT points[] = {{100, 100}, {500, 100}, {500, 300}, {100, 300}, {100, 100}};
                hDC = BeginPaint(hWnd, &ps );

                SelectObject(hDC, hBrush);
                SelectObject(hDC, hPen);
                Ellipse(hDC, 100, 100, 500, 300);
                PolyLine(hDC, points, 5);

                EndPaint(hWnd, &ps);
        break;
        }
```

SetPixel fonksiyonu bir noktaya tek bir pixel basmak için kullanılır.

```
COLORREF
SetPixel(
 HDC hdc,
 int X,
 int Y,
 COLORREF crColor
);
```

Fonksiyonun 1. parametresi DC; 2. ve 3. parametreler, pixelin koordinatları; son parametre ise pixelin rengini belirtir.

```
case WM_PAINT:
                        {
                                PAINTSTRUCT ps;
                                HDC hDC;
                                POINT points[] = {{100, 100}, {500, 100}, {500, 300},
                                {100, 300}, {100, 100}};

                                hDC = BeginPaint(hWnd, &ps);

                                SelectObject(hDC, hBrush);
                                SelectObject(hDC, hPen);
```

```
                Ellipse(hDC, 100, 100, 500, 300);
                Polyline(hDC, points, 5);

                SetPixel(hDC, 10, 10, RGB(0, 0, 255));

                EndPaint(hWnd, &ps);
                break;
        }
```

GetPixel fonksiyonu ile bir koordinattaki pixel rengi elde edilebilir.

```
COLORREF GetPixel(
 HDC hdc,
 int XPos,
 int YPos
);
```

Fonksiyonun 1. parametresi DC'nin handle değeri; 2. ve 3. parametreleri pixelin koordinatlarıdır. Fonksiyonun geri dönüş değeri, pixelin rengini belirtir.

Timer İŞLEMLERİ

Windows sistemlerinde belirli bir periyotta mesaj oluşumun sağlayan yöntemler vardır. Örneğin biz her 1 saniyede bir, bir mesajın kuyruğa bırakılmasını sağlayabiliriz. Böylece bu mesajdan hareketle periyodik işlem gerektiren uygulamalar oluşturabiliriz. Her mesajda saatin saniyesi 1 arttırılabilir ve böylelikle bir saat programı yazılabilir. Bir oyun programındaki şekillerin belirli bir hızdaki hareketi yine bu yöntemle sağlanabilir. Böyle bir timer mekanizması oluşturabilmek için SetTimer isimli API fonksiyonu kullanılır.

```
UINT SetTimer(
 HWND hWnd,
 UINT nIDEvent,
 UINT uElapse,
 TIMERPROC lpTimerFunc
);
```

Fonksiyonun 1. parametresi WM_TIMER mesajının gönderileceği pencerenin handle değeridir. 2. parametre, yaratılacak Timer'ın ID değeridir. Bu ID değerine WM_TIMER mesajında gereksinim duyulabilmektedir. Fonksiyonun 1. parametresi NULL geçilebilir. Bu durumda 2. parametre fonksiyon tarafından dikkate alınmaz. Fonksiyonun 3. parametresi, Timer'ın milisaniye(ms) cinsinden periyot değeridir. Fonksiyonun son parametresi, bir fonksiyon göstericisidir. SetTimer fonksiyonu, belirli periyotta WM_TIMER mesajını bırakmak yerine burada belirtilen fonksiyonu da çağırabilmektedir. Normal mesaj uygulamasında bu parametre NULL geçilir. Fonksiyonun geri dönüş değeri başarı durumunda yaratılan Timer'ın ID değeri; başarısızlık durumunda 0 değeridir.

Görüldüğü gibi programcı SetTimer fonksiyonunda bir ID ve bir periyot değeri vermektedir. İşletim sistemi de belirlenen bu periyotlarda WM_TIMER isimli mesajı belirlenen pencere için kuyruğa bırakmaktadır.

WM_TIMER mesajının wParam parametresi, SetTimer fonksiyonunun 2. parametresinde belirtilen ID değeridir. Böylelikle biz, birden fazla Timer yaratarak WM_TIMER mesajının hangi timer nedeniyle oluştuğunu belirleyebiliriz. Mesajın lParam parametresi, SetTimer fonksiyonuna son parametresi olarak girilen fonksiyon adresidir. Bu konu üzerinde ileride durulacaktır.

Timer mekanizması, sistem genelinde bir kaynaktır. Sistemin bir timer kapasitesi vardır. Her ne kadar bu kapasite gittikçe yükseltilmekte ise de sınırsız değildir. Örneğin bir program çok fazla timer yaratmış ise başka programların yaratabileceği timer sayısı azalacaktır. Ayrıca, Timer mekanizmasının bir hata payı vardır. Windows, çeşitli nedenlerden dolayı, timer mesajını gecikmeli olarak yollayabilir. Yani gerçek zamanlı işlemler gerektiren hassas olaylar, bu timer mekanizması ile kontrol edilemez. Timer mekanizmasının duyarlılığı üzerine MSDN'de makaleler bulunmaktadır.

SetTimer fonksiyonu ile yaratılmış olan bir timer, programcının işi bittiğinde KillTimer fonksiyonu ile yok edilmelidir.

```
BOOL KillTimer(
 HWND hWnd,       // handle of window that istalled timer
 UINT uIDEvent    // timer identifier
);
```

Fonksiyonun 1. parametresi pencerenin handle değeri, 2. parametresi timer'ın ID değeridir. Windows, timer ID değerlerini, pencere genelinde tutmaktadır. Yani farklı pencereler için yaratılmış pencerelerin Timer'larının ID değerleri aynı olabilir. Normal olarak SetTimer fonksiyonunun 2. parametresi olarak girilen ID değeri ile SetTimer geri döner. Fakat SetTimer fonksiyonunda o pencerenin handle değeri NULL geçilirse fonksiyon, programcının verdiği ID değerini dikkate almamakta; pencereden bağımsız bir ID değerini kendisi vermektedir. Ayrıca daha önce yarattığımız bir Timer'ı aynı ID değeri ile yaratmak istersek sistem, eski timer'ı otomatik yok etmekte; aynı ID'ye ilişkin yeni Timer yaratmaktadır (tabi sistem programlama bakımından sistemin-Windows- eski timer'ın handle alanını silmeyip yalnızca periyodunu değiştirmesi de mümkündür). Program sonlandığında Windows, o program tarafından oluşturulmuş tüm timer'ları yok eder.

```
#include <windows.h>

#define ID_TIMER                100


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
```

```
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HBRUSH hRedBrush, hBlueBrush;
        static UINT idTimer;
        static BOOL flag;

        switch (message) {
                case WM_CREATE:
                        hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
                        hBlueBrush = CreateSolidBrush(RGB(0, 0, 255));
                        if (!(idTimer = SetTimer(hWnd, ID_TIMER, 1000, NULL)))
                                return -1;
                        break;
                case WM_TIMER:
                        SetClassLong(hWnd, GCL_HBRBACKGROUND,
                                        (LONG) (flag ? hBlueBrush : hRedBrush));
                        flag = !flag;
                        InvalidateRect(hWnd, NULL, TRUE);
                        break;
                case WM_DESTROY:
                        KillTimer(hWnd, idTimer);
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
#include <windows.h>
#include <stdio.h>
#include <time.h>

#define ID_TIMER                100


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                        int nCmdShow)
{
```

```
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HBRUSH hRedBrush, hBlueBrush;
	static UINT idTimer;
	static BOOL flag;

	switch (message) {
		case WM_CREATE:
			hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
			hBlueBrush = CreateSolidBrush(RGB(0, 0, 255));
			if (!(idTimer = SetTimer(hWnd, ID_TIMER, 1000, NULL)))
				return -1;
			break;
		case WM_TIMER:
			{
				time_t t;
				struct tm *pTime;
				char clock[30];

				SetClassLong(hWnd, GCL_HBRBACKGROUND,
					(LONG) (flag ? hBlueBrush : hRedBrush));
```

```
                        flag = !flag;
                        InvalidateRect(hWnd, NULL, TRUE);

                        t = time(NULL);
                        pTime = localtime(&t);
                        sprintf(clock, "%02d:%02d:%02d", pTime->tm_hour,
                                pTime->tm_min, pTime->tm_sec);
                        SetWindowText(hWnd, clock);
                }
                break;
        case WM_DESTROY:
                KillTimer(hWnd, idTimer);
                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

Sınıf Çalışması: Küçük bir daire çiziniz. Dairenin içi kırmızı olsun. Her saniyede şekli rasgele bir yönde hareket ettiriniz. Şekil, yukarı, sağa yada aşağı doğru hareket etmelidir ve asla çalışma alanının dışına çıkmamalıdır.

Açıklamalar: Bir saniye periyotlu bir timer yaratılır. Her WM_TIMER mesajı oluştuğunda eski şekil silinerek yeni şekil rasgele bir yönde yeniden çizilir. Şekil herhangi bir yönden taştığında öbür uçtan çıkacak şekilde düzenleme yapılır. Ayrıca WM_PAINT mesajında şeklin yeniden çıkması sağlanabilir. bak: ball1.c, ball2.c, ball3.c, ball4.c

/* ball4.c */

```
#include <windows.h>
#include <stdlib.h>
#include <time.h>

#define ID_TIMER                    100
#define CIRCLE_RADIOUS              5
#define MOVE_SIZE                   5

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
```

```
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static UINT timerID;
	static RECT circleRect;
	static HBRUSH hRedBrush, hWhiteBrush;
	static HPEN hWhitePen, hBlackPen;
	static int clientWidth, clientHeight;

	PAINTSTRUCT ps;
	HDC hDC;

	switch (message) {
		case WM_CREATE:
			timerID = SetTimer(hWnd, ID_TIMER, 100, NULL);
			if (timerID == 0)
				return -1;
			hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
			hWhiteBrush = GetStockObject(WHITE_BRUSH);
			hWhitePen = GetStockObject(WHITE_PEN);
			hBlackPen = GetStockObject(BLACK_PEN);
			srand(time(0));
			break;
		case WM_TIMER:
		   {
			hDC = GetDC(hWnd);

			SelectObject(hDC, hWhitePen);
			SelectObject(hDC, hWhiteBrush);
			Ellipse(hDC, circleRect.left, circleRect.top, circleRect.right, circleRect.bottom);

			switch (rand() % 4) {
			   case 0:
				if (circleRect.right < 0)
					circleRect.right = clientWidth,
					circleRect.left = clientWidth - 2 * CIRCLE_RADIOUS;
				else
					OffsetRect(&circleRect, -MOVE_SIZE, 0);
				break;
```

```
                    case 1:
                        if (circleRect.left > clientWidth)
                            circleRect.left = 0, circleRect.right = 2 * CIRCLE_RADIOUS;
                        else
                            OffsetRect(&circleRect, MOVE_SIZE, 0);
                        break;
                    case 2:
                        if (circleRect.top > clientHeight)
                            circleRect.top = 0, circleRect.bottom = 2 * CIRCLE_RADIOUS;
                        else
                            OffsetRect(&circleRect, 0, MOVE_SIZE);
                        break;
                    case 3:
                        if (circleRect.bottom < 0)
                            circleRect.bottom = clientHeight,
                            circleRect.top = clientHeight - 2 * CIRCLE_RADIOUS;
                        else
                            OffsetRect(&circleRect, 0, -MOVE_SIZE);

                        break;
                }


                SelectObject(hDC, hRedBrush);
                SelectObject(hDC, hBlackPen);

                Ellipse(hDC, circleRect.left, circleRect.top, circleRect.right, circleRect.bottom);

                ReleaseDC(hWnd, hDC);
                }
                break;
        case WM_PAINT:
                hDC = BeginPaint(hWnd, &ps);

                SelectObject(hDC, hRedBrush);
                Ellipse(hDC, circleRect.left, circleRect.top, circleRect.right, circleRect.bottom);
                EndPaint(hWnd, &ps);

                break;
        case WM_SIZE:
                {
                    int middleX = LOWORD(lParam) / 2;
                    int middleY = HIWORD(lParam) / 2;

                    circleRect.left = middleX - CIRCLE_RADIOUS;
                    circleRect.top = middleY - CIRCLE_RADIOUS;
                    circleRect.right = middleX + CIRCLE_RADIOUS;
                    circleRect.bottom = middleY + CIRCLE_RADIOUS;

                    clientWidth = LOWORD(lParam);
                    clientHeight = HIWORD(lParam);

                }
                break;
        case WM_DESTROY:
                KillTimer(hWnd, timerID);

                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
```

}

SetTimer fonksiyonunda son parametre, aşağıdaki prototipe uygun bir fonksiyon adresi biçiminde girilebilir:

```
VOID CALLBACK TimerProc(
 HWND hwnd,        // handle of window for timer messages
 UINT uMsg,        // WM_TIMER message
 UINT idEvent,     // timer identifier
 DWORD dwTime      // current system time
);
```

Bu durumda SetTimer fonksiyonunun 1. parametresindeki pencere handle değeri NULL olarak geçilmelidir. Bu parametre NULL olarak geçildiğinde ise TIMER_ID değerini fonksiyon dikkate almaz; yani fonksiyon önerilen ID değerini değil kendi ID değerini verir. SetTimer fonksiyonunun son parametresi NULL geçilmez de bir fonksiyon adresi olarak girilirse, sistem belirlenen periyotta WM_TIMER mesajı göndermek yerine bu fonksiyonu çağırır. (SetTimer, penceresiz fonksiyonlarda kullanılamaz).

Timer (TimerProc) fonksiyonunun çağırılması da yine mesaj kuyruğu yoluyla gerçekleşmektedir. WM_TIMER mesajı kuyruğa bırakılmakta; GetMessage tarafından alınmakta ve DispatchMessage fonksiyonuna sokulduğunda bu fonksiyon, Timer fonksiyonunu çağırmaktadır. Özetle, Timer fonksiyonun çağırılması için yine mesaj kuyruğu ve mesaj döngüsü gerekmektedir. Console uygulamalarında Timer mekanizması maalesef bu biçimde kullanılamamaktadır.

THREADLER

Win32 sistemleri, çok thread'li sistemledir. Prosess, ismine ana thread denilen bir thread'le çalışmaya başlar; diğer thread'ler programın çalışma zamanı sırasında CreateThread API fonksiyonu ile yaratılır. Win32 sistemlerinde çizelgeleme işlemleri, thread temelinde preemptive bir biçimde yapılmaktadır. Yani her quanta süresi dolduğunda bir thread'in çalışmasına ara verilir; sıradaki diğer thread çalıştırılır. Bir thread'in çalışmasına ara verilmesinin başka bir thread'in çalışmaya başlatılması işlemine Thread'ler arası geçiş(context switch) denilmektedir. Geçilen thread, aynı prosesin thread'i olabileceği gibi başka bir thread'in thread'i de olabilir. Aynı prosesin thread'leri, aynı bellek alanı üzerinde çalışırlar. Daha teknik bir anlatımla, geçilen thread de aynı prosesin thread'i ise 2GB'lik kullanıcı alanında bir değişiklik yapılmaz. Eğer farklı bir prosesin thread'i ise düşük anlamlı 2GB'lik alan da diğer prosesin bilgileriyle değiştirilir.

```
HANDLE CreateThread(
 LPSECURITY_ATTRIBUTES lpThreadAttributes,   // pointer to security attributes
 DWORD dwStackSize,                          // initial thread stack size
 LPTHREAD_START_ROUTINE lpStartAddress,      // pointer to thread function
 LPVOID lpParameter,                         // argument for new thread
 DWORD dwCreationFlags,                      // creation flags
 LPDWORD lpThreadId                          // pointer to receive thread ID
);
```

Bir thread temel olarak 3 biçimde sonlanabilir:

1. Thread'in başlangıç fonksiyonunun doğal olarak sonlanmasıyla
2. ExitThread fonksiyonunun çağırılmasıyla. Bu API fonksiyonunu hangi thread akışı çağırırsa o thread sonlandırılmaktadır.
3. Bir thread başka bir thtead'i TerminateThread API fonksiyonu ile sonlandırabilir. Bir thread'in dışarıdan zorla sonlandırılması, son çare olarak uygulanmaktadır.

Bilindiği gibi exit standart C fonksiyonu, hangi sistem için yazılmışsa o sistemin proses sonlandıran fonksiyonunu çağırmaktadır. exit, prosesi sonlandırmadan önce stdio tamponlarını flush eder ve açık tüm

dosyaları kapatır. Win32 sistemlerinde prosesi sonlandıran gerçek fonksiyon, ExitProcess'tir. Standartlara göre main fonksiyonu bittiğinde exit fonksiyonu ile proses sonlandırılmaktadır. Proses sonlandırıldığında prosesin tüm thread'leri de yok edilmektedir.

CreateThread Fonksiyonu
Thread yaratan tek fonksiyon CreateThread fonksiyonudur.

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes,     // pointer to security attributes
  SIZE_T dwStackSize,                           // initial thread stack size
  LPTHREAD_START_ROUTINE lpStartAddress,        // pointer to thread function
  LPVOID lpParameter,                           // argument for new thread
  DWORD dwCreationFlags,                        // creation flags
  LPDWORD lpThreadId                            // pointer to receive thread ID
);
```

Fonksiyonun 1. parametresi Kernel nesnesinin güvenlik bilgileridir, NULL geçilebilir. Fonksiyonun 2. parametresi yaratılacak thread'in stack uzunluğunu belirtir. Bu paramtere 0 olarak geçilirse PE formatında belirtilen stack uzunluğu esas alınır. Microsoft linkerları default olarak stack uzunluğu 1MB olarak almaktadır. Yani bu parametre 0 olarak geçildiğinde dafault stack uzunluğu MB olmaktadır. Fonksiyonun 3. parametresi thread akışının başlatılacağı fonksiyonun başlangıç adresidir. Thread fonksiyonu olarak kullanılacak fonksiyonun aşağıdaki gibi bir parametrik yapıya sahip olması gerekir:

```
DWORD WINAPI ThreadProc( LPVOID lpParameter // thread data );
```

Görüldüğü gibi thread fonksiyonunun parametresi void *, geri dönüş değeri DWORD türündendir. Thread fonksiyonunun çağırma biçimi WINAPI yani __stdcall biçimindedir. Fonksiyonun 4. parametresi, Thread fonksiyonuna geçirilecek parametreyi belirtir. Thread fonksiyonu çağırıldığında buraya yazılan değer parametre olarak aktarılmaktadır. Fonksiyonun 5. parametresi, thread'in yaratım özelliğini belirtir. Bu parametre 0 geçilebilir yada CREATE_SUSPENDED biçiminde geçirilebilir. Eğer 0 geçilirse thread yaratılır yaratılmaz thread çalışmaya başlar. Yok eğer CREATE_SUSPENDED geçilirse thread yaratılır ama hemen çalışmaya başlamaz; çalışmaya başlatmak için ResumeThread fonksiyonunu uygulamak gerekir. Fonksiyonun son parametresi thread'in ID değerinin yerleştirileceği DWORD türünden nesnenin adresidir. CreateThread fonksiyonu başarı durumunda yaratılan thread'in handle değerine, başarısızlık durumunda NULL değerine geri dönmektedir. Thread'in handle değeri, daha sonra thread ile ilgili bir takım işlemlerde kullanılır. Thread'in ID değeri, bu kursta ele alınmayacak olan bazı durumlarda gerekmektedir.

bak: thread.c

```
/* thread.c*/

#include <windows.h>
#include <stdio.h>
#include  <stdlib.h>


DWORD WINAPI ThreadFunc(LPVOID pParam)
{
        int i;

        for (i = 0; i < 100; ++i) {
                printf("%d\n", i);
                Sleep(1000);
        }
```

```
	return 0;
}

int main(void)
{
	HANDLE hThread;
	DWORD dwThreadID;

	printf("main thread!..\n");

	hThread = CreateThread(NULL, 0, ThreadFunc, 0, 0, &dwThreadID);

	if (hThread == NULL) {
		printf("Cannot create thread!..\n");
		exit(EXIT_FAILURE);
	}

	getchar();

	return 0;
}
```

Sleep Fonksiyonu

Çok işlemli ve çok thread'li sistemlerde program akışını bekletmek için meşgul bir döngü oluşturmak yerine thread'in geçici süre çizelge dışında tutulması daha etkin bir yöntemdir. İşte Sleep fonksiyonu hangi thread tarafında çağırılmışsa o thread'i parametresi ile belirtilen ms kadar çizelge dışında bekletir. Bu zaman dolduğunda thread'in yeniden çizelgeye alınması, işletim sisteminin sorumluluğundadır.

VOID Sleep(DWORD dwMilliseconds // sleep time in milliseconds **);**

Parametre olarak INFINITE geçilirse thread süresiz çizelge dışı bırakılır. Parametre olarak 0 özel değeri geçilirse thread geri kalan quanta süresini kullanmadan fakat çizelge dışına da çıkartılmadan thread'ler arası geçiş olur.

ExitThread Fonksiyonu

ExitThread, bir thread akışını sonlandırır.

VOID ExitThread(DWORD dwExitCode // exit code for this thread **);**

Fonksiyonun parametresi, thread'in sonlanma değeridir. Bu değerin özel bir anlamı yoktur. İstenirse yine programcı tarafından elde edilip kullanılmaktadır.

Aslında Thread fonksiyonu bittiğinde de yine thread, ExitThread fonksiyon ile sonlandırılmaktadır.

CreateThrad fonksiyonu ile yeni bir akışın yaratılması aşağıdakine benzer bir mantıkla yapılmaktadır:

HANDLE CreateThread(..., ThreadProc, ...)
{
1. Yeni bir akış ve çizelge elemanı oluşturuldu
2. Yeni akışın başlangıç noktasını (BEGIN) ver.

return yeni yaratılan thread'in handle değeri

BEGIN:

```
        rethread = ThreadProc(...);
        ExitThread(rethread);

}
```

Thread'lerin Beklenmesi ve WaitForXXX Fonksiyoları

Win32 sistemlerinde Kernel nesneleri denilen sistem nesneleri üzerinde belirli olaylar gerçekleşene kadar thread'i bloke eden WaitForSingleObject ve WaitForMultipleObjects isimli iki fonksiyon vardır. Thread'ler de bir Kernel nesnesidir. Bu fonksiyonlar, thread'leri de bekleyebilmektedir. WaitForSingleObject ve WaitForMultipleObjects fonksiyonları aslında bazı ayrıntıları olan genel fonksiyonlardır. Kernel nesneleri ve bu fonksiyonların ayrıntıları kursumuzun kapsamı dışındadır ve "Win32 sistem programlama kursu" nun konusudur.

```
DWORD WaitForSingleObject(
 HANDLE hHandle,          // handle to object to wait for
 DWORD dwMilliseconds     // time-out interval in milliseconds
);
```

Fonksiyonun 1. parametresi, beklenecek Kernel nesnesinin handle değeridir. Biz bu parametreyi yarattığımız thread'in handle değeri olarak geçmeliyiz. 2. parametre zaman aşımı değeridir. Bu değer eğer söz konusu thread sonlanmazsa en kötü olasılıkla beklenecek zaman miktarını belirtmektedir. Bu parametreye INFINITE özel değeri geçilebilir. Bu durumda blokenin çözülmesi ancak thread'in sonlanması ile mümkündür. Fonksiyon başarısızlık durumunda WAIT_FAILED özel değerine geri dönmektedir. Eğer geri dönüş değeri WAIT_OBJECT_0 ise thread sonlandığından dolayı bloke çözülmüştür. WAIT_TIMEOUT ise zaman aşımından dolayı bloke çözülmüştür. O halde yarattığımız bir thread'in sonlanmasını aşağıdaki gibi bekleyebiliriz:

```
hThread = CreateThread(...);
...
...
WaitForSingleObject(hThread, INFINITE);
```

WaitForMultipleObjects fonksiyonu birden fazla Kernel nesnesini aynı anda beklemek için kullanılır.

```
DWORD WaitForMultipleObjects(
 DWORD nCount,                  // number of handles in the handle array
 CONST HANDLE *lpHandles,       // pointer to the object-handle array
 BOOL fWaitAll,                 // wait flag
 DWORD dwMilliseconds           // time-out interval in milliseconds
);
```

Fonksiyonun 1. ve 2. parametreleri, içerisinde Kernel nesnelerinin handle eğerlerinin bulunduğu dizinin uzunluğu ve başlangıç adresidir. Yani biz bekleyeceğimiz thread'lerin handle değerlerini bir diziye yerleştirmeliyiz; o dizinin uzunluğunu ve adresini parametre olarak geçirmeliyiz. Fonksiyon, bu thread'lerin hepsi sonlanana kadar yada herhangi birisi sonlanana kadar bekleme yapabilir. İşte 3. parametre hepsi sonlanana kadar beklenecekse TRUE, herhangi birisi sonlanana kadar beklenecekse FALSE geçilmelidir. Fonksiyonun son parametresi zaman aşımı değeridir; INFINITE geçilebilir. Fonksiyonun geri dönüş değeri dökümanlardan izlenmelidir. bak:thread4-5.c

```
/*thread4.c */
```

```
#include <windows.h>
#include <stdio.h>
#include  <stdlib.h>

#define NTHREADS                10

DWORD WINAPI ThreadFunc(LPVOID pParam)
{
        char *pStr = (char *) pParam;
        int i;

        for (i = 0; i < 10; ++i) {
                printf("%s: %d\n", pStr, i);
                Sleep(500);
        }

        free(pStr);

        return 0;
}

int main(void)
{
        HANDLE hThreads[NTHREADS];
        DWORD dwThreadID;
        int i;
        char *pStr;

        printf("main thread!..\n");

        for (i = 0; i < NTHREADS; ++i) {
                pStr = (char *) malloc(50);
                sprintf(pStr, "Thread (%d)", i + 1);
                hThreads[i] = CreateThread(NULL, 0, ThreadFunc, pStr, 0, &dwThreadID);
        }


        WaitForMultipleObjects(NTHREADS, hThreads, TRUE, INFINITE);

        return 0;
}
```

Thread'ler ve Prosesin Bellek Alanı

Aynı prosese ilişkin thread'ler aynı bellek alanını kullanmaktadır. Thread'ler prosesin data alanını ve heap alanını ortak olarak kullanırlar. Yani tüm thread'ler prosesin aynı statik nesnelerini ve heap alanını kullanmaktadır.

Her thread'in stack'i birbirinden ayrılmıştır. Yerel değişkenler ve parametre değişkenleri stack'te yaratıldığına göre her thread akışı yerel değişkenlerin ve parametre değişkenlerinin farklı bir kopyasını kullanır. Örneğin iki thread aynı fonksiyon üzerinde ilerliyor olsun. Bir thread, fonksiyonun yerel değişkenini değiştirdiğinde o thread, kendi kopyasını değiştiriyor durumdadır. Yani diğer thread bundan etkilenmez. Fakat bir thread bir global değişkeni değiştirirse diğer thread o anda artık onu değişmiş görür. Çünkü global değişkenlerin thread başına ayrı bir kopyası yoktur; proses genelinde tek bir kopyası vardır.

Thread'ler ve Paralel Programlama

Bilgisayarımızda tek bir mikroişlemci varsa gerçekte iki thread aynı anda çalışamaz.

Anahtar Notlar:

Intel'in yeni kuşak mikroişlemcilerinde Hyper threading denilen bir özellik vardır. Hyper threading kabaca tek bir işlemcinin aynı anda birden fazla makine kodunu paralel çalıştırabilmesi anlamına gelir. Tabi bu işlem kısıtlı ölçüde yapılabilmektedir. Hyper threading, birden fazla CPU'nun gösterdiği performansı henüz gösterememektedir. Bilgisayarımızda birden fazla işlemci varsa işletim sistemi, prosesin farklı thread'lerini farklı işlemcilerde aynı anda çalıştırabilmektedir. Bu durumda gerçek bir aynı anda çalışma söz konusu olabilmektedir.

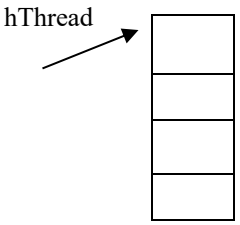
Thread'in Handle Alanı ve Thread'lere İlişkin Önemli Fonksiyonlar

Bir thread'i yarattığımızda CreateThread API fonksiyonu bize bir handle değeri verir. Thread fonksiyonu sonlanınca bu handle alanı otomatik olarak yok edilmez. Örneğin en azından thread'in sonlanış kodu bu handle alanında saklanmaktadır; programı bu çıkış kodunu aldıktan sonra handle alanını yok etmek isteyebilir. Özetle thread'in handle alanının yok edilmesi programcı tarafından ayrıca yapılmalıdır. Bu işlem için **CloseHandle** API fonksiyonu kullanılır.

```
BOOL CloseHandle(
  HANDLE hObject   // handle to object to close
);
```

Eğer programcı isterse thread'i yarattıktan sonra hemen CloseHandle uygulayabilir. Bu durumda handle alanı yok edilmez. Yalnızca bir bayrak tutulur; thread sonlandığında otomatik olarak handle alanı yok edilir.

```
HANDLE hThread;
....
hThread = CrateThrea(...);
CloseHandle(hThread);
```



Thread'in sonlanış kodu GetExitCodeThread API fonksiyonu ile alınabilir.

```
BOOL GetExitCodeThread(
 HANDLE hThread,          // handle to the thread
 LPDWORD lpExitCode       // address to receive termination status
);
```

Fonksiyonun 1. parametresi thread'in handle değeri; 2. parametresi thread'in sonlanış kodunun yerleştirileceği DWORD bir nesnenin adresidir. bak: thread16Ek.c

```
/* thread16Ek.c */

#include <windows.h>
#include <stdio.h>
```

```
#include  <stdlib.h>

#define NTHREADS                10

DWORD WINAPI ThreadFunc(LPVOID pParam)
{

        ExitThread(500);

        return 100;
}

int main(void)
{
        HANDLE hThread;
        DWORD dwThreadID;
        DWORD dwExitCode;

        printf("main thread!..\n");

        hThread = CreateThread(NULL, 0, ThreadFunc, 0, 0, &dwThreadID);
        WaitForSingleObject(hThread, INFINITE);

        GetExitCodeThread(hThread, &dwExitCode);

        printf("%ld\n", dwExitCode);

        return 0;
}
```

Bir thread'e ilişkin zamansal istatistikler GetTimeThread API fonksiyonu ile alınabilir.

```
BOOL GetThreadTimes(
 HANDLE hThread,                // specifies the thread of interest
 LPFILETIME lpCreationTime,     // when the thread was created
 LPFILETIME lpExitTime,         // when the thread was destroyed
 LPFILETIME lpKernelTime,       // time the thread has spent in kernel mode
 LPFILETIME lpUserTime // time the thread has spent in user mode
);
```

Fonksiyonlarda sözü edilen FILETIME yapısı 2 DWORD elemandan oluşan 1.1.1601'den itibaren geçen 100 nano saniyeleri tutar. Bu kullanışsız yapı **FiletimeToSystemTime** fonksiyonu ile **SYSTEMTIME** denilen daha kullanışlı bir yapıya dönüştürülür.

Win32'de Thread'lerin Çizelgelenmesi

Win32 sistemlerinde öncelik sınıflarına ayrılmış döngüsel çizelgeleme (Round Rubrn Scheduling with Priority Class) tekniği kullanılmaktadır. Bu tekniğe göre sistemdeki her thread'in 0–31 arası bir öncelik derecesi vardır. Eşit öncelikli thread'ler ayrı bir sınıf oluşturur. Sistem en yüksek öncelikli sınıfı kendi arasında döngüsel çizelgeler. Eğer bu sınıfta çizelgelenecek hiçbir thread kalmadıysa bir sonraki thread grubunu kendi aralarında çizelgeler. Thread'ler sonlanarak yada bloke olarak çizelgeden çıkabilirler. Bir öncelik grubu üzerinde çizelgeleme yapılırken daha yüksek öncelikli bir thread çizelgeye girdiği zaman sistem, o grubu bırakarak yeniden yüksek gruba geçer. Örneğin sistemde aşağıdaki gibi 6 thread bulunuyor olsun:

T1	T2
18	18

T3 T4 T5
8 8 8

T6
1

Burada önce T1 ve T2 kendi aralarında sırasıyla çalıştırılır. Bunlar çeşitli gerekçelerle çizelge dışı kaldığında bu kez T3, T4, T5 grubu kendi aralarında çalıştırılır. Bunlar da çizelge dışı kaldığında T1 çalışma fırsatı bulabilir. Şimdi 8 öncelikli grubun kendi aralarında çalıştırıldığını düşünelim. T2 thread'inin blokesinin çözülüp sisteme geri döndüğünü varsayalım. Sistem, bu 8'lik grubu bırakarak yeniden T2'yi çizelgelemeye başlar.

Win32 sistemlerinde çok thread'li çalışmalar yaparken diğer thread'leri düşünmemiz gerekir. Thread önceliğimizi yükseltirsek ve hiç bloke olmazsak değer thread'ler çalışma fırsatı bulamayabilir. Örneğin biz arka planda bir olayı bir thread içerisinde for döngüsü ile izliyor olalım. Döngü her yinelendiğinde bir sleep işlemi yapmak bile sistemi rahatlatacaktır.

```
for(;;) {
...
sleep(50);
}
```

Thread Önceliğinin Belirlenmesi

Win32'de thread öncelikleri; prosesin öncelik sınıfı denilen taban bir değerle, thread'in göreli önceliği denilen bir değerin toplanması ile elde edilir. Prosesin öncelik sınıfı SetPriorityClass API fonksiyonu ile değiştirilebilir.

```
BOOL SetPriorityClass(
 HANDLE hProcess,          // handle to the process
 DWORD dwPriorityClass     // priority class value
);
```

Fonksiyonun 1. parametresi prosesin handle değeridir. Kendi prosesimizin handle değerini GetCurrentProcess API fonksiyonu ile alabiliriz:

```
HANDLE GetCurrentProcess(VOID)
```

SetPriorityClass fonksiyonunun 2. parametresi aşağıdakilerden biri olabilir:

IDLE_PRIORITY_CLASS (4)
NORMAL_PRIORITY_CLASS (8)
NORMAL_PRIORITY_CLASS (10)
HIGH_PRIORITY_CLASS (13)
REALTIME_PRIORITY_CLASS (24)

Proses yaratıldığında default öncelik sınıfı NORMAL_PRIORITY_CLASS öncelik sınıfındadır. Thread'in göreli önceliği ise SetThreadPriority fonksiyonu ile belirlenir.

```
BOOL SetThreadPriority(
 HANDLE hThread, // handle to the thread
```

```
 int nPriority // thread priority level
);
```

Fonksiyonun 1. parametresi thread'in handle değeridir. 2. parametre, prosesin öncelik sınıfına hangi değerin toplanıp çıkartılacağını belirtir. Şunlardan biri olabilir:

THREAD_PRIORITY_ABOVE_NORMAL (+1)
THREAD_PRIORITY_BELOW_NORMAL (-1)
THREAD_PRIORITY_HIGHEST (+2)
THREAD_PRIORITY_LOWEST (-2)
THREAD_PRIORITY_NORMAL (0)
THREAD_PRIORITY_IDLE (Eğer prosesin öncelik sınıfı REALTIME ise thread'in toplam önceliğini 16'ya, başka biri ise thread'in önceliğini 1'e çeker.)
THREAD_PRIORITY_TIME_CRITICAL (Eğer prosesin öncelik sınıfı REALTIME ise thread'in toplam önceliğini 32'ye, başka biri ise 16'ya çeker.)

Prosesin öncelik sınıfı ve thread'in göreli öncelik derecesine göre thread'in alacağı toplam öncelik derecesi ilgili dökümanlarda tablo halinde verilmiştir. Yani programcı bu tablodan 0-31 öncelik derecesinin hangisini istediğine karar verir; buna uygun öncelik sınıfı ver thread'in göreli önceliğini tablodan alır.

	Process Priority Class	**Thread Priority Level**
1	IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, HIGH_PRIORITY_CLASS	or THREAD_PRIORITY_IDLE
2	IDLE_PRIORITY_CLASS	THREAD_PRIORITY_LOWEST
3	IDLE_PRIORITY_CLASS	THREAD_PRIORITY_BELOW_NORMAL
4	IDLE_PRIORITY_CLASS	THREAD_PRIORITY_NORMAL
5	Background NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_LOWEST
	IDLE_PRIORITY_CLASS	THREAD_PRIORITY_ABOVE_NORMAL
6	Background NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_BELOW_NORMAL
	IDLE_PRIORITY_CLASS	THREAD_PRIORITY_HIGHEST
7	Foreground NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_LOWEST
	Background NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_NORMAL
8	Foreground NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_BELOW_NORMAL
	NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_ABOVE_NORMAL
9	Foreground NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_NORMAL
	NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_HIGHEST
10	Foreground NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_ABOVE_NORMAL
11	HIGH_PRIORITY_CLASS	THREAD_PRIORITY_LOWEST
	Foreground NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_HIGHEST
12	HIGH_PRIORITY_CLASS	THREAD_PRIORITY_BELOW_NORMAL
13	HIGH_PRIORITY_CLASS	THREAD_PRIORITY_NORMAL
14	HIGH_PRIORITY_CLASS	THREAD_PRIORITY_ABOVE_NORMAL
15	IDLE_PRIORITY_CLASS,	THREAD_PRIORITY_TIME_CRITICAL

	NORMAL_PRIORITY_CLASS, HIGH_PRIORITY_CLASS	or
	HIGH_PRIORITY_CLASS	THREAD_PRIORITY_HIGHEST
16	REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_IDLE
22	REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_LOWEST
23	REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_BELOW_NORMAL
24	REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_NORMAL
25	REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_ABOVE_NORMAL
26	REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_HIGHEST
31	REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_TIME_CRITICAL

O an çalışmakta olan thread'in handle değeri, GetCurrentThread fonksiyonu ile elde edilebilir.

HANDLE GetCurrentThread(VOID)

```
/* thread16Eki1.c */

#include <windows.h>
#include <stdio.h>
#include  <stdlib.h>

#define NTHREADS                10

DWORD WINAPI ThreadFunc(LPVOID pParam)
{

        return 100;
}

int main(void)
{
        SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS);
        SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_TIME_CRITICAL);

        for (;;) {

                Sleep(50);
        }

        return 0;
}
```

Thread'lerin Senkronizasyonu

Thread'lerle işlemler yaparken thread'lerin birbirlerini beklemesi gerekebilir. Tread'ler arası geçiş işlemi herhangi bir zaman preemptive bir biçimde geçekleşir. Bu durumda thread'lerin ortak bir işi bir arada yapabilmesi için bir senkronizasyon oluşturmak gerekebilir. Örneğin global bir bağlı liste olsun ve her iki thread de aynı bağlı listeye eleman ekliyor olsun. Bir thread bağlı listeye eleman eklerken o arada thread'ler arası geçiş oluşsa, geçilen thread de bağlı listeye eleman eklemek istese bağlı liste dağılabilir.

```
int g_i;
int g_x[SIZE];

void Zero(void)
{
        g_x[g_i] = 0;
```

```
        ++g_i;
}
```

İşte bizim bir thread eleman eklerken thread'ler arası geçiş oluşup akış başka bir thread'e geçse bile o thread'in öbür thread'in işini bitirene kadar beklemesi gerekir.

Belirli bir anda yalnızca tek bir thread tarafından işlenmesi gereken kodlara kritik kodlar (critical section) denilmektedir. Kritik kodda thread'ler arası geçişler oluşup akış kaybedilse bile başka bir thread'in kritik koda girmiş olan thread çıkana kadar o koda girmemesi sağlanmalıdır.

Bir kodun atomik olması ise, o kodun çalışması bitene kadar thread'ler arası geçiş olmaması demektir.

Kritik kodlar, global bir takım flag değişkenleri kullanılarak manuel bir biçimde oluşturulamazlar. Örneğin:

```
BOOL g_Flag = FALSE;
...
while (g_Flag)
    ;
g_Flag = TRUE;
...
...                          } Kritik kod
...
g_Flag = FALSE;
```

Bu kod gerekeni yapmaz çünkü ok ile gösterilen yerde thread'ler arası bir geçiş oluşura aynı anda birden fazla thread, kendini kod içinde bulunabilir. Ayrıca yukarıdaki kodun 2. bir dezavantajı da, beklemenin CPU zamanı harcanarak meşgul bir döngü içerisinde yapılmasıdır.

İşte her işletim sisteminin kritik od oluşturmakta kullanılan sistem fonksiyonları vardır. Bu sistem fonksiyonları, thread'ler arası geçiş oluşmasını sağlayan kesmeleri disable ederek bu işlemlerini gerçekleştirir. Win32 sistemlerinde de bu işlemler için kullanılacak pek çok fonksiyon vardır. Bunların en basiti, InitializeCriticalSection, EnterCriticalSection ve LeaveCriticalSection fonksiyonlarıdır. Kritik kod, EnterCriticalSection fonksiyonuyla LeaveCriticalSection fonksiyonu arasındaki koddur.

```
InitializeCriticalSection(...);
...
....
EnterCriticalSection(...);
...
...                          } Kritik kod
...
...
LeaveCriticalSection(...);
```

Bir thread, EnterCriticalSection fonksiyonunu geçtiğinde artık başka bir thread diğer thread LeaveCritialSection ile kritik koddan çıkmadığı sürece blokede bekler.

```
VOID InitializeCriticalSection(
 LPCRITICAL_SECTION lpCriticalSection        // address of critical section object
);
```

```
VOID EnterCriticalSection(
```

```
 LPCRITICAL_SECTION lpCriticalSection        // pointer to critical section object
);
```

```
VOID LeaveCriticalSection(
 LPCRITICAL_SECTION lpCriticalSection        // address of critical section object
);
```

Programcı, CRITICAL_SECTION türünden bir değişken tanımlar ve fonksiyonlara bunun adresini geçirir.

Anahtar Notlar:

Genel olarak C yada C++ standartlarında thread'ler konusunda birşeyler söylenmemiştir. Dolayısıyla C'nin ve C++'ın standart fonksiyonları thread güvenli (Thread Safe) olmak zorunda değildir. Bu fonksiyonları çağıran programcının kodu senkronize etmesi gerekebilir.

GUI Uygulamalarında Thread'lerin Tipik Kullanımı

Bir tuşa basıldığında uzun sürecek bir işlemin başlatıldığını düşünelim. Örneğin bu işlemde bir döngü içerisinde sürekli bir aygıt kontrol ediliyor olsun. Biz, işlemi durduran başka bir tuşa bassak bile işlemi bir etkimiz dokunmaz. Çünkü diğer tuşa basış bilgisini işlenmesi için bile akışın yeniden GetMessage fonksiyonuna gelmesi ve tuşa basım mesajının alınıp işlenmesi gerekir. Halbuki akışımız bir döngü içerisinde arka plan işlemi yapaktadır. Bu, tipik bir problemdir. Çok thread'li bir ortamda bu problemin çok basit bir çözümü vardır. İşlemi başlatan tuşa basıldığında bir thread yaratılır; arka plan işlemi o thread yapar. Böylece asıl thread mesaj döngüsünü işlemeye devam eder. Diğer tuşa basıldığında arka plan işlemi yapan thread sonlandırılarak işlem gerçekleştirilebilir. bak: thread_gui1.c, 2, 3

Mesaj Sistemi Ve Thread'ler

Win32 sistemlerinde her thread'in ayrı bir mesaj kuyruğu bulunur. İster alt pencere olsun ister üst pencere olsun bir thread akışının yarattığı tüm pencerelerin mesajları, o thread'in mesaj kuyruğuna bırakılır. Bu bakımdan thread'ler pencereli ve penceresiz olmak üzere 2'ye ayrılır. Penceresiz thread'lere "Worker Threads" de denilmektedir. Pencereli thread'lere ise "GUI Threads" denilmektedir. Bir thread akışı içerisinde bir pencere yaratılmışsa artık o pencereye ilişkin tüm mesajlar, o thread'in mesaj kuyruğuna bırakılacaktır. GetMessage fonksiyonu ise kendi thread'inin mesajlarını alır. O halde buradan çıkan sonuç şudur:

Biz bir thread'de bir pencere yaratacaksak WinMain fonksiyonunda yaptığımız gibi bir mesaj döngüsü oluşturmalıyız.

Pencereli thread açan programlara tipik örnek Internet Explorer. Bu programda her yeni pencere açtığımızda aslında aynı kod farklı bir thread çalıştırır. bak: thread_multi_gui.c, 2

DİYALOG PENCERELERİ

API programlamada bir form oluşturmak oldukça zahmetlidir. Örneğin bir form üzerinde pek çok kontrol bulunur. Bu kontrollerin hepsinin CreateWindow fonksiyonu ile yaratılması zahmetlidir. Diyalog penceresi bir popup bir pencere ve onun üzerindeki kontrollerden oluşan bir pencere sistemidir. Diyalog pencerelerine ilişkin bir diyalog kaynağı söz konusudur. Diyalog kaynağı, kaynak dosyada belirtilir; sonra DialogBox fonksiyonu ile tüm pencere sistemi bir hamlede yaratılır. Diyalog kaynağında şu belirlemeler yapılmaktadır.

— Diyalog penceresinin ana penceresine ilişkin tanımlamalar
— Kontrollerin diyalog penceresi içerisindeki yerleri, özellikleri vs.

Diyalog Kaynağının Oluşturulması

Diyalog kaynağı, kaynak editörüyle oluşturulabilir yada manuel olarak yazılabilir. Dİyalog penceresi kaynağının genel biçimi şöyledir:

```
<Kaynak ismi> DIALOG row, col, genişlik, uzunluk <pencere biçimleri> CAPTION "Yazı"
{
        PUSHBUTTON "Yazı", Pencere biçimi, row1, col1, genişlik, uzunluk
        EDITTEXT "Yazı", Pencere biçimi, row1, col1, genişlik, uzunluk
        LTEXT "Yazı", Pencere biçimi, row1, col1, genişlik, uzunluk
        ...
}
```

Görüldüğü gibi diyalog penceresi tanımlaması için de tek tek kontroller belirtilmiştir. Aslında bu kontrol anahtar sözcüklerinin yerine tüm kontrolleri kapsayan CONTROL anahtar sözcüğü yerleştirilebilir. . Bu durumda kontrolün türü, sınıf ismine bakılarak anlaşılır. Örneğin:

CONTROL "button", "Ok", BS_PUSHBUTTON, row, col, genişlik, uzunluk

Kontrollerde belirtilen koordinatlar, diyalog penceresinin ana penceresinin çalışma alanına göre; diyalog penceresinin ana penceresinin koordinatları ise masaüstünün sol üst köşesine göredir.

Visual C kaynak editöründe diyalog pencere kaynağı tamamen görsel bir biçimde oluşturulabilmektedir. Kaynak oluşturulduktan sonra ilgili kontrolün üstüne gelinip farenin sağ tuşuna basılıp Properties seçilirse o kontrole ilişkin bilgiler görsel bir biçimde görüntülenir.

Diyalog Pencerelerin Pencere Fonksiyonları

Diyalog penceresi yaratıldığında onun gerçek pencere fonksiyonu User32.dll modülündeki DefDlgProc isimli fonksiyondur. Yani diyalog penceresinin kontrolleri üzerinde gerçekleşen tüm olaylarda DispatchMessage, DefDlgProc fonksiyonunu çağırır. DefDlgProc, sarma fonksiyondur. Bazı klasik işlemleri yaptıktan sonra programcını sağladığı yapay bir pencere fonksiyonunu çağırır. Yani programcı, diyalog penceresi için yapay bir pencere fonksiyon yazar; bu yapay pencere fonksiyonu,i gerçek pencere fonksiyonu olan DefDlgProc tarafından çağırılır. Eğer gerçek pencere fonksiyonunu programcı yazmak zorunda kalsaydı diyalog penceresinde otomatik yapılan işlemleri de onun sağlaması gerekirdi. Halbuki bu işlemleri DefDlgProc yapmakta fakat bizim yapay pencere fonksiyonumuzu çağırarak bize de fırsat tanımaktadır. Diyalog penceresinin yapay pencere fonksiyonunun parametrik yapısı şöyle olmalıdır:

```
BOOL CALLBACK DialogProc(
 HWND hWnd,              // handle to dialog box
 UINT message,           // message
 WPARAM wParam,          //first message parameter
 LPARAM lParam           // second message parameter
);
```

Görüldüğü gibi yapay diyalog pencere fonksiyonunun geri dönüş değeri BOOL türdendir.

Yapay diyalog pencere fonksiyonunda mesajlar işlenirken dikkat edilmesi gereken en önemli nokta, DefWindowProc fonksiyonunun çağırılmamasıdır. Bu fonksiyon zaten gerçek pencere fonksiyonu olan

DefDlgProc tarafından çağırılır. DefDlgProc, yapay pencere fonksiyonu 0 dışı bir değerle geri döndüğünde mesajın işlendiğini düşünür ve DefWindowProc fonksiyonunu çağırmaz; 0 değeri ile geri döndüğünde DefWindowProc fonksiyonunun çağırır. Bu durumda yapay pencere fonksiyonunun işlenmesinde tipik olarak message , switch içerisine alınır; işlenilen mesajlardan TRUE, işlenmeyen mesajlardan FALSE ile geri dönülür.

```
switch (message) {
        case ...:
                ...
                return TRUE;
        ...
        }
return FALSE;
}
```

DialogBox API fonksiyonu

Diyalog penceresinin kaynağını hazırladıktan ve yapay pencere fonksiyonunu yazdıktan sonra artık DialogBox fonksiyonu ile diyalog penceresi açılabilir.

```
int DialogBox(
 HINSTANCE hInstance,     // handle to application instance
 LPCTSTR lpTemplate,      // identifies dialog box template
 HWND hWndParent,         // handle to owner window
 DLGPROC lpDialogFunc     // pointer to dialog box procedure
);
```

Fonksiyonun 1. parametresi, PE formatının yüklenme adresi olan hInstance değeridir. 2. parametre, diyalog kaynağının ismini alır. 3. parametre diyalog penceresinin üst penceresi olacak pencerenin handle değeridir. Fonksiyonun 4. parametresi, diyalog penceresinin yapay pencere fonksiyonunun adresidir. Fonksiyon, EndDialog fonksiyonun geri dönüş değeri ile geri döner.

Diyalog Penceresinin Kapatılması Ve EndDialog Fonksiyonu

Diyalog penceresini sonlandırmak için EndDialog fonksiyonu kullanılır.

```
BOOL EndDialog(
 HWND hDlg,       // handle to dialog box
 int nResult      // value to return
);
```

Fonksiyonun 1. parametresi, diyalog penceresinin handle değeri; 2. parametresi DialogBox API fonksiyonunun geri dönüş değerini oluşturacak değerdir. EndDialog fonksiyonu, diyalog penceresini hemen kapatmaz yalnızca bir bayrağı set eder. Diyalog penceresinin kapatılması, gerçek diyalog pencere fonksiyonu tarafından yapılmaktadır. (bak dialogbox dosyası içi)....

Model ve Modeless Pencere Kavramları

Diyalog pencereleri Model ve Modeless olmak üzere ikiye ayrılır. Diyalog penceresi açıldığında akışın diyalog penceresi kapatılana kadar bekletildiği diyalog pencerelerine Model pencereler denir. Halbuki modeless pencerelerde hem diyalog penceresi açıktır hem de akış, ana mesaj menüsünde devam etmektedir. Örneğin MessageBox penceresi Model, Find&Replace pencereleri Modeless pencerelerdir. Model diyalog

pencereleri DialogBox fonksiyonu ile; Modeless pencereleri ise CreateDialog pencereleri ile yaratılmaktadır. bak: generic.c (newdialog içinde)

Anahtar Notlar:

DialogBox fonksiyonu gerçekte DialogBoxParam fonksiyonuna ilişkin bir makrodur. Eskiden Win16 zamanlarında DialogBox, gerçek bir fonksiyondu ve DialogBoxParam fonksiyonu da yoktu. DialogBoxParam fonksiyonunun 1 parametre fazlalığı vardır.

Diyalog Penceresinin Kapatılması ve DialogBox Fonksiyonunun Geri Dönüş Değeri

Pek çok klasik diyalog penceresinde OK ve Cancel biçiminde iki tuş bulunur. OK, girilen bilgilerin doğruluğunu, Cancel ise girilen bilgilerin dikkate alınmaması gerektiğini belirtir. Bu her iki tuş da diyalog penceresini kapatmaktadır. O halde DiyalogBox API fonksiyonunu çağıran programcı bu tespiti yapabilmelidir. DialogBox API fonksiyonu, EndDialog fonksiyonuna geçilen 2. parametre ile geri dönmektedir. Bu durumda programcı diyalog penceresini duruma göre EndDialog(hWnd, IDOK) yada EndDialog(hWnd, IDCANCEL) biçiminde kapatabilir. Bu durumda OK tuşuna basılarak çıkıldığı şöyle anlaşılabilir:

```
if (DialogBox(...) == IDOK) {
        ...
        ...
}
```

Diyalog penceresinin X kapatma tuşuna basıldığında da sanki Cancel tuşuna basılmış gibi IDCANCEL mesajı ile WM_COMMAND mesajı gönderilir.

MW_INITDIALOG Mesajı

Diyalog penceresinin ana penceresine WM_CREATE mesajı geldiğinde DefDlgProc, yani gerçek diyalog pencere fonksiyonu, bu mesajla yapay diyalog pencere fonksiyonunu çağırmaz. Yani yapay diyalog penceresine WM_CREATE biçiminde bir mesaj gelmez. Fakat gerçek pencere fonksiyonları diyalog kaynağına bakarak kontrolleri yarattıktan sonra yapay pencere fonksiyonunu WM_INITDIALOG isimli mesajla çağırır. WM_INITDIALOG mesajı, her pencereye gönderilen gerçek bir mesaj değildir. Yalnızca yapay diyalog pencerelerine gönderilen sahte bir mesajdır. WM_INITDIALOG mesajı geldiğinde diyalog penceresi ve dolayısıyla kontroller görünür değildir ama tüm pencereler yaratılmıştır. Programcı bu mesajda bir takım ilk işlemleri yapmalıdır. Örneğin tipik olarak listeleme kutusuna elemanları ekleyebilir, seçenek kutularını çarpılayabilir. WM_INITDIALOG, yapay pencere fonksiyonuna gönderilen ilk mesajdır.

Yapay Diyalog Pencere Fonksiyonları İçerisinde Kontroller Üzerinde İşlemler

Yapay diyalog pencere fonksiyonu içerisinde kontrollere mesaj gönderebilmek için onların handle değerlerini bilmemiz gerekir. Oysa bu pencereler bizim tarafımızdan değil DialogBox fonksiyonu tarafından yaratılmıştır. Fakat elimizde gerçek diyalog penceresinin handle değeri ve kontrollerin ID değeri olduğuna göre GetDlgItem fonksiyonu ile kontrollerin handle değerlerini elde edebiliriz. Zaten GetDlgItem ismi, buradaki tipik kullanımdan dolay yanlış bir biçimde oyulmuştur. Bu durumda programcı ne zaman kontrollere mesaj gönderecek olsa GetDlgItem fonksiyonu ile ilgili kontrolün handle değerini elde ederek ilgili işlemi yapabilir. Yada programcı WM_INITDIALOG mesajı içerisinde bir kez bu fonksiyon ile kontrollerin handle değerlerini alıp bunları statik ömürlü nesnelerde saklayabilir.

Diyalog Penceresi Kontrollerinden Bilgilerin Alınması

Diyalog pencerelerinin çoğu form amaçlı kullanılır. Yani kullanıcı çeşitli bilgiler girer. Bu bilgiler programcı tarafından alınarak işlenir. Bu durumda programcı diyalog penceresi kapanmadan önce bilgileri kontrolden alarak taşıması gerekmektedir. Örneğin OK tuşuna basıldığında bilgiler bir yere aktarılabilir.

Diyalog Tabanlı Uygulamalar

Model diyalog pencerelerinde akış nasıl bekletilmektedir? İşte DialogBox fonksiyonu içerisinde başka bir mesaj döngüsü kurulur ve o mesaj döngüsü gelen mesajları işlemeye başlar. Bu mesaj döngüsü kuyrukta diyalog penceresine olmayan mesajları atar ve yalnızca kendisine ilişkin mesajları işler. Diyalog penceresi kapatıldığında bu sefer gerçek mesaj döngüsü çalışmaya devam edecektir.

DialogBox API fonksiyonu içerisinde bir mesaj döngüsü kurulduğuna göre programın hiç ana penceresi olmadan diyalog tabanlı bir uygulama yazılabilir. Yani WinMain fonksiyon içerisinde doğrudan DialogBox API fonksiyonu çağırılır; böylece programın ana penceresi diyalog penceresi olur. Bu durumda diyalog tabanlı bir uygulamanın iskeleti şöyle olur: bak dialogbased.c

/* dialogbased.c */

```
#include <windows.h>
#include "resource.h"

BOOL CALLBACK DialogProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
        return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL, DialogProc);
}

BOOL CALLBACK DialogProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        ...
        return FALSE;
}
```

Burada DialogBox API fonksiyonunun hWnd parametresi NULL olarak geçilmiştir. Bu değer, programın ana penceresinin masaüstü olduğu anlamına gelir.

Diyalog Tabanlı Uygulamalar nerelerde kullanılır?

Diyalog tabanlı uygulamalar tipik olarak küçük programlarda basitlik sağladığı için tercih edilmektedir. Örneğin bir programımız sistemle ilgili bazı ayarlamalar yapacak olsun; diyalog tabanlı yazabiliriz. Örnek olarak Windows'un hesap makinesi tipik bir diyalog tabanlı uygulamadır. Fakat orta ve büyük programların diyalog tabanlı oluşturulması, ayrıntılardan dolayı zordur.

PENCEREYE YAZI YAZAN API FONKSİYONLARI

API düzeyinde pencere içerisine yazı azmakta kullanılan yalnızca iki fonksiyon vardır. Bu fonksiyonlar, **TextOut** ve **DrawText** fonksiyonlarıdır.

Fonksiyonun birinci parametresi, yazma işleminde kullanılacak DC yi belirtir. İkinci ve üçüncü parametreler yazının yazılacağı yerin sol üst köşe koordinatıdır. Dördüncü parametre, yazının başlangıç adresini, nihayet

son parametre de yazılacak karakter sayısını belirtmektedir. Bu fonksiyon NULL karakter('\0') görene kadar değil, son parametrede belirtilen sayı kadar karakter basmaktadır.

Örnek:

```
#include <windows.h>
#include "resource.h"

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
        return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL, DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;

        switch (message) {
                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK && HIWORD(wParam) == BN_CLICKED) {
                                EndDialog(hWnd, 0);
                                return TRUE;
                        }
                        if(LOWORD(wParam) == IDCANCEL && HIWORD(wParam) == BN_CLICKED){
                                EndDialog(hWnd, 0);
                                return TRUE;
                        }
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);

                        TextOut(hDC, 100, 100, "This is a test", sizeof("This is a test") - 1);

                        EndPaint(hWnd, &ps);
                        return TRUE;


        }
        return FALSE;
}
```

TextOut , çok ilkel bir fonksiyondur. Örneğin '\n' gibi karakterlere duyarlı değildir.

```
int DrawText(
  HDC hDC,                  // handle to DC
  LPCTSTR lpString,         // text to draw
  int nCount,               // text length
  LPRECT lpRect,            // formatting dimensions
  UINT uFormat              // text-drawing options
);
```

Fonksiyonun birinci parametresi, kullanılacak DC yi, ikinci parmetresi, yazdıralacak yazıyı belirtir. Üçüncü parametre, yazının uzunluğudur. Eğer bu değer "–1" geçilirse, '\0' görene kadar yazdırma yapılır. **DrawText()** fonksiyonu, bir çerçevesel bölgenin içerisinde hizalama yapmaktadır. Fnksiyonun dördüncü

parametresi, bu çerçevesel bçlgeyi belirtir. Son parametre, hizalamanın nasıl yapılacağını anlatmaktadır. Şunlardan oluşturulabilir:

Value	Description
DT_BOTTOM	Justifies the text to the bottom of the rectangle. This value is used only with the DT_SINGLELINE value.
DT_CALCRECT	Determines the width and height of the rectangle. If there are multiple lines of text, **DrawText** uses the width of the rectangle pointed to by the *lpRect* parameter and extends the base of the rectangle to bound the last line of text. If there is only one line of text, **DrawText** modifies the right side of the rectangle so that it bounds the last character in the line. In either case, **DrawText** returns the height of the formatted text but does not draw the text.
DT_CENTER	Centers text horizontally in the rectangle.
DT_EDITCONTROL	Duplicates the text-displaying characteristics of a multiline edit control. Specifically, the average character width is calculated in the same manner as for an edit control, and the function does not display a partially visible last line.
DT_END_ELLIPSIS	For displayed text, if the end of a string does not fit in the rectangle, it is truncated and ellipses are added. If a word that is not at the end of the string goes beyond the limits of the rectangle, it is truncated without ellipses. The string is not modified unless the DT_MODIFYSTRING flag is specified. Compare with DT_PATH_ELLIPSIS and DT_WORD_ELLIPSIS.
DT_EXPANDTABS	Expands tab characters. The default number of characters per tab is eight. The DT_WORD_ELLIPSIS, DT_PATH_ELLIPSIS, and DT_END_ELLIPSIS values cannot be used with the DT_EXPANDTABS value.
DT_EXTERNALLEADING	Includes the font external leading in line height. Normally, external leading is not included in the height of a line of text.

DT_HIDEPREFIX	**Windows 2000/XP:** Ignores the ampersand (&) prefix character in the text. The letter that follows will not be underlined, but other mnemonic-prefix characters are still processed. For example:
	`input string:  "A&bc&&d"`
	`normal:  "Abc&d"`
	`DT_HIDEPREFIX:  "Abc&d"`
	Compare with DT_NOPREFIX and DT_PREFIXONLY.

DT_INTERNAL	Uses the system font to calculate text metrics.
DT_LEFT	Aligns text to the left.
DT_MODIFYSTRING	Modifies the specified string to match the displayed text. This value has no effect unless DT_END_ELLIPSIS or DT_PATH_ELLIPSIS is specified.
DT_NOCLIP	Draws without clipping. **DrawText** is somewhat faster when DT_NOCLIP is used.
DT_NOFULLWIDTHCHARBREAK	**Windows 98/Me, Windows 2000/XP:** Prevents a line break at a DBCS (double-wide character string), so that the line breaking rule is equivalent to SBCS strings. For example, this can be used in Korean windows, for more readability of icon labels. This value has no effect unless DT_WORDBREAK is specified.

DT_NOPREFIX	Turns off processing of prefix characters. Normally, **DrawText** interprets the mnemonic-prefix character & as a directive to underscore the character that follows, and the mnemonic-prefix characters && as a directive to print a single &. By specifying DT_NOPREFIX, this processing is turned off. For example,
	`input string:  "A&bc&&d"`
	`normal:  "Abc&d"`
	`DT_NOPREFIX:  "A&bc&&d"`
	Compare with DT_HIDEPREFIX and DT_PREFIXONLY.

DT_PATH_ELLIPSIS	For displayed text, replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle. If the string contains backslash (\) characters, DT_PATH_ELLIPSIS preserves as much as possible of the text after the last backslash. The string is not modified unless the DT_MODIFYSTRING flag is specified.

	Compare with DT_END_ELLIPSIS and DT_WORD_ELLIPSIS.

DT_PREFIXONLY	**Windows 2000/XP:** Draws only an underline at the position of the character following the ampersand (&) prefix character. Does not draw any other characters in the string. For example,
	`input string:  "A&bc&&d"`
	`normal:  "Abc&d"`
	`DT_PREFIXONLY:  "  _  "`
	Compare with DT_HIDEPREFIX and DT_NOPREFIX.

DT_RIGHT	Aligns text to the right.
DT_RTLREADING	Layout in right-to-left reading order for bi-directional text when the font selected into the *hdc* is a Hebrew or Arabic font. The default reading order for all text is left-to-right.
DT_SINGLELINE	Displays text on a single line only. Carriage returns and line feeds do not break the line.
DT_TABSTOP	Sets tab stops. Bits 15–8 (high-order byte of the low-order word) of the *uFormat* parameter specify the number of characters for each tab. The default number of characters per tab is eight. The DT_CALCRECT, DT_EXTERNALLEADING, DT_INTERNAL, DT_NOCLIP, and DT_NOPREFIX values cannot be used with the DT_TABSTOP value.
DT_TOP	Justifies the text to the top of the rectangle.
DT_VCENTER	Centers text vertically. This value is used only with the DT_SINGLELINE value.
DT_WORDBREAK	Breaks words. Lines are automatically broken between words if a word would extend past the edge of the rectangle specified by the *lpRect* parameter. A carriage return-line feed sequence also breaks the line.
DT_WORD_ELLIPSIS	Truncates any word that does not fit in the rectangle and adds ellipses. Compare with DT_END_ELLIPSIS and DT_PATH_ELLIPSIS.

DT_VCENTER, **DT_SINGLELINE** ile kullanılır.

Normal olarak DrawText, **'\n'** karakterini gördüğünde ikinci satıra geçer fakat bunun için **DT_SINGLELINE** brlirlemesinin yapılmamış olması gerekir.

```
#include <windows.h>
#include <stdio.h>

#define NRADIO_BUTTONS                  10
#define ID_RADIOBUTTON_BASE             100

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
```

```
		if (!RegisterClass(&wndClass))
			return -1;
	}
	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hRadioButtons[NRADIO_BUTTONS];

	switch (message) {
		case WM_CREATE:
		{
			int i;
			char text[2] = {'A', '\0'};

			for (i = 0; i < 5; ++i) {

			hRadioButtons[i] = CreateWindow("button", text ,
				WS_CHILD|WS_VISIBLE|WS_BORDER|BS_AUTORADIOBUTTON |
				((i == 0) ? WS_GROUP : 0), 100, 100 + 20 * i, 50, 20, hWnd,
				(HMENU) (ID_RADIOBUTTON_BASE + i),
		((LPCREATESTRUCT) lParam)->hInstance, NULL);
			++text[0];
			}

			text[0] = '1';
			for (;i < NRADIO_BUTTONS; ++i) {

			hRadioButtons[i] = CreateWindow("button", text ,
				WS_CHILD|WS_VISIBLE|WS_BORDER|BS_AUTORADIOBUTTON |
				((i == 5) ? WS_GROUP : 0),100, 100 + 20 * i, 50, 20, hWnd,
				(HMENU) (ID_RADIOBUTTON_BASE + i),
		((LPCREATESTRUCT) lParam)->hInstance, NULL);
			++text[0];
			}
		}
		break;
		case WM_LBUTTONDOWN:
		{
			int i;
			HWND hRadio;
			char text[2] = "A";
```

```
                    for (i = 0; i < 5; ++i) {
                    hRadio = GetDlgItem(hWnd, ID_RADIOBUTTON_BASE + i);
                            if (SendMessage(hRadio, BM_GETCHECK, 0, 0))
                                    break;
                    }
                    text[0] = 'A' + i;
                    MessageBox(hWnd, (i == 5)? "Unchecked" :
                    text, "Message", MB_OK);
            }
            break;
            case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
            default:
              return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

TextOut ve DrawText ile yazı yazarken yazınını şekil ve zemin rengi söz konusudur. Yazının zemin rengi ile pencerenin zemin rengi uyuşmazsa yazı güzel gözükmez. Yazıların şekil ve zemin renkleri DC alanında saklanmaktadır. DC yaratıldığında yazıların default zemin rengi beyaz, şekil rengi siyahtır. Yazının şekil ve zemin renkleri, **SetTextColor()** ve **SetBkColor()** fonksiyonlarıyla değiştirilebilir.

```
COLORREF SetTextColor(
  HDC hDC,                  // handle to DC
  COLORREF crColor          // text color
);
```

```
COLORREF SetBkColor(
  HDC hDC,                  // handle to DC
  COLORREF crColor          // text color
);
```

Görüldüğü gibi yazıların zemin ve şekil renklerinin DC deki fırça ve kalemle ilişkisi yoktur. Başka bir değişle yazılar DC deki kalemle yazılmaz.

WINDOWS'UN CONTROL PANELDE BELİRLENEN RENKLERİ

Bilindiği gibi, kullanıcılar "Control Panel(Denetim masası)" den çeşitli öğelerin rengini değişitirebilir. Bu renkler değiştiğinde programımızın bundan etkilenmesi, arzu edilen bir durumdur. Windows, Control Panle'de belilenen renkleri kendi iççerisinde saklamakta, sistem kapatıldığında da bu bilgileri "registery" dosyaysına yazmaktadır. Sistem açıldığında, bu renklerle açılacaktır.

Control panelde belirlenen bu renkleri biz, GetSysColor isimli fonksiyon ile alabiliriz.

```
DWORD GetSysColor(
  int nIndex            // display element
);
```

Fonksiyonun parametresi, kontrol panel deki hangi rengin elde edileceğini belirten sembolik sabittir. Örneğin;

COLOR_WINDOW, pencerenin zemin rengini, **COLOR_CAPTIONTEXT**, pencere başlık yazısının rengini belirtmektedir.

Düğme ile Dialog penceresinin zemin rengi aynıdır ve **COLOR_3DFACE, COLOR_BTNFACE** ile temsil edilir.

Kontrol paneldeki sistem renkleri, **SetSysColors** ile değişrilebilir.

Fonksiyon, istenildiği kadar renkleri değiştirebilir. Bunun içiçn birinci parametre, değiştirilecek renk sayısını belirtmektedir. İkinci parametre ile üçüncü parametre, paralel dizilerdir. İkinci parametreyle belirtilen dizinnin elemanları **COLOR_XXX,** biçimindeki sembolik sabitlerden, üçüncü dizinin elemanları ise renkleri belirten değerlerden oluşur. Örneğin pencerenin zemin rengini sistem genelinde programlama yoluyla değiştirmek isteyelim:

```
#include <windows.h>
#include "resource.h"


BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
        return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL, DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        HDC hDC;
        PAINTSTRUCT ps;
        RECT rect;

        switch (message) {
                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK && HIWORD(wParam) == BN_CLICKED) {
                                int type = COLOR_WINDOW;          //////////////      burada
                                int color = RGB(0, 0, 255);        /////////         zemin renkleri
                                SetSysColors(1, &type, &color);    ////////////      değiştiriliyor.

                                EndDialog(hWnd, 0);
                                return TRUE;
                        }
                        if (LOWORD(wParam) == IDCANCEL && HIWORD(wParam) == BN_CLICKED) {
                                EndDialog(hWnd, 0);
                                return TRUE;
                        }
                case WM_PAINT:
                        hDC = BeginPaint(hWnd, &ps);

                        SetTextColor(hDC, RGB(255, 0, 0));
                        SetBkColor(hDC, RGB(0, 255, 0));


                        GetClientRect(hWnd, &rect);
                        DrawText(hDC, "This is a test\nYes this is a test", -1, &rect, DT_LEFT);


                        EndPaint(hWnd, &ps);
                        return TRUE;
```

```
        }
        return FALSE;
}
```

GetSysColorBrush(), API fonksiyonu doğrudan Kontrol peneldeki renkeler ilişkin bir fırça verir.
Bu işlem aşagıdaki çağırmanın tam eşleniği değildir.

CreateSolidBrush(GetSysColor(nIndex));

CreateSolidBrush her seferinde yeni bir fırça yaratır.Halbuki GetSysColorBrush daha önce yaratılmış bir fırçayı yineleyebilmeltedir.GetSysColorBrush ile alınan fırça DeleteObject ile bırakılmaya çalışılmamalıdır.

Kontrol panelden bir renk değişikliği yapıldığında sistem tüm ana pencerelere WM_SYSCOLORCHANGE mesajını yollar.Böylece programcı bu değişikliği fark ederek bir takım düzenlemeler yapabilir.

Pencereyi Kontrol Panel Renklerine Göre Otomatik Ayarlamak

Sistem renklerine ilişkin otomatik fırça yaratmak için daha kolay bir yol düşünülmüştür. Fırça nesnesine ilişkin handle değeri bilindiği gibi bir göstericidir. Win32'de hiçbir adres bilgisinin yüksek anlamlı WORD değeri 0 olamaz. İşte bir fırçanın handle değerini belirten göstericiye COLOR_XX ile belirtilen sistem renkleri yerleştirilirse bu durum otomatik olarak kontrol panelden o renge ilişkin bir fırçanın yaratılıp kullanılacağı anlamına gelir. Kontrol panel renklerine ilişkin COLOR_XXX sembolik sabitleri 0'dan başlatılmıştır. Halbuki 0 gösterici için NULL adres anlamına gelir. İşte bu yüzden bu değerlerin 1 fazlası kullanılmaktadır. Örneğin:

```
HBrush = (HBRUSH) (COLOR_WINDOW + 1);
```

Burada hBrush değişkenine atanan COLOR_WINDOW + 1 değeri kontrol paneldeki pencerelerin zemin rengine ilişkin fırça rengi anlamına gelmektedir. Bu hBrush değeri kullanıldığı zaman API fonksiyonları COLOR_WINDOW rengine ilişkin bir fırça üretip onu kullanırlar. Bu durumda pencere zemin renginin kontrol panel renklerine göre otomatik olarak değişmesi için yapılacak tek şey şudur:

```
wndClass.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
```

Kontrol panelde renkleri değiştiğinde zemin renginin otomatik değiştirilmesi şöyle de yapılabilir:

```
case WM_SYSCOLORCHANGE:
        SetClassLong(hWnd, GCL_HBRBACKGROUND, COLOR_WINDOW + 1);
        InvalidateRect(hWnd, NULL, TRUE);
        break;
```

Kontrollerin Renklendirilmesi

Kontroller, pencere fonksiyonları Windows'un içerisinde olan alt pencerelerdir. Tüm pencerelerin ister ana pencere olsun ister alt pencere olsun güncelleme alanları vardır ve onlara WM_PAINT mesajı gönderilmektedir. Örneğin içerisinde kontrollerimizin olduğu bir pencereyi minimize durumdan restore (maximize) durumuna geçirelim. Bu durumda WM_PAINT mesajı yalnızca ana pencere için değil tüm kontroller için de gönderilecektir. Bu kontroller kendi çizimlerini kendileri yapacaklardır.

Kontroller, çizim işlemlerinde dışarıdan belirleme yapılabilmesi için şöyle bir yöntem kullanmaktadır: Kontrollere WM_PAINT mesajı geldiğinde kontroller, BeginPaint ile DC'yi elde ettikten sonra üst

pencereye WM_CTLCOLORXXX biçiminde mesaj gönderirler (SendMessage ile). Mesaja yarattıkları DC'yi parametre olarak geçerler. Programcı da bu DC'yi kullanarak çeşitli set işlemleri yapar. mesajdan dönüldüğünde kontrol, çizimleri bu DC ile yapacağından programcının belirlemeleri devreye girmiş olur. WM_CTLCOLORXXX mesajlarında programcı bir fırçanın handle değeri ile geri dönmelidir. Kontrol bu fırçayı, zemini boyamak için kullanacaktır.

Her kontrol için ayrı bir WM_CTLCOLOR mesajı vardır. Örneğin edit kontrolü üst pencereye WM_CTLCOLOREDIT mesajını; statik kontrol WM_CTLCOLORSTATIC mesajını göndermektedir. Tüm bu mesajların parametrik yapısı aynıdır. Mesajın wParam parametresine DC'nin handle değeri; lParam parametresine kontrolün handle değeri yerleştirilir. Programcı da yarattığı fırçanın handle değeri ile geri dönmelidir.

Programcı, ya aynı cinsten tüm kontrolleri aynı renge boyamak ister yada bunların her biri için farklı bir renk kullanmak ister. Örneğin bizim 5 tane edit kontrolümüz olsun. WM_CTLCOLOREDIT bu 5 kontrolün hepsi için gönderilecektir. Kontrole göre renk ayrımı yapacaksak WM_CTLCOLOREDIT mesajının hangi kontrolden gönderildiğini tespit etmemiz gerekir. Programcı, mesajdan elde edilen kontrol handle değerini GetDlgCtrlId fonksiyonu ile ID değerine dönüştürüp switch içerisine sokabilir.

Diyalog pencerelerinin zemin rengi default olarak düğme rengidir. Fakat programcılar bazen bu rengi değiştirmek isteyebilir. Diyalog penceresinin renkleri için de aynı biçimde diyalog penceresinin yapay pencere fonksiyonuna WM_CTLCOLORDLG mesajı gönderilmektedir.

Düğmelerin ve statik kontrolün daha çok diyalog pencereleri üzerinde kullanılacağı düşüncesi ile bunların zemin renkleri diyalog pencereleri ile aynı biçimde alınmıştır. Bu durum özellikle statik kontrolün ana pencere üzerinde kullanıldığı durumlarda zemin rengi uyuşmazlığına yol açmaktadır. Bu nedenle WM_CTLCLOLORSTATIC mesajı kullanılabilir.

Font Kavramı

DrawText ve TextOut fonksiyonlarının yazı biçimi, DC'de belirtilmiş fonta bağlıdır. DC yaratıldığında default font olarak System Font denilen font vardır.

Font, tamamen kalemle fırça gibi bir çizim nesnesidir. Font nesnesi yaratan CreateFont gibi API fonksiyonları vardır. DrawText ve TextOut fonksiyonlarının yaratılmış olan fontları kullanabilmesi için SelectObject API fonksiyonu ile bu fontların, diğer çizim nesneleri gibi DC'ye bağlanması gerekir.

Font, çeşitli bileşenleri olan bir kavramdır. Bir fontun şu bileşenleri vardır:

— Büyüklüğü: Örneğin biz bir yazının büyük karakterlerle görüntülenmesini istiyorsak büyük karakterli bir font belirlemeliyiz. Font karakterlerinin büyüklüğü genellikle dpi (dot per inch) ile belirlenmektedir.
— Font karakterlerinin eşit genişlikte mi farklı genişlikte mi olduğu: Bası fontlarda karakterler eşit genişliktedir. Örneğin Courier fontu böyle bir fonttur. Fakat bazı fontlarda karakterlerin her biri farklı genişlikte olabilmektedir.
— Fontun Yazı karakteristiği: Her fonttaki karakterlerin tipik bir biçimsel özelliği vardır. Örneğin bazı fontlardaki karakterler el yazısı karakterleri gibidir. Font karakteristikleri "Times New Roman", "Arial" gibi isimlerle belirtilmektedir.
— Fontun İkincil Karakteristikleri: Karakterlerin yatık yada kalın çizgili olması da bir font karakteristiğidir. Fakat her fontun italik, bold yada bold-italik biçimleri olmayabilir.

Yazının rengi bir font karakteristiği değildir. Yani biz SelectObject ile yani bir font seçmiş olalım. Onu renklendirmek için yine SetTextColor, SetBkColor gibi API fonksiyonları kullanmak zorunda kalırız.

Font, bir dosya biçiminde bir formata sahip bir bilgidir. CreateFont API fonksiyonunun fontu yaratabilmesi için font karakterlerinin bulunduğu font dosyasının incelemesi gerekir. Windows sistemi yüklendiğinde zaten tipik bazı font dosyaları Windows dizinine çekilmektedir.

Klavye Mesajları

Klavyeden herhangi bir tuşa basıldığında Windows, Klavye Odağı (Keyboard Focus) hangi pencerede ise o pencere için kuyruğa WM_KEYDOWN mesajını bırakır. El tuştan çekildiğinde WM_KEYUP mesajı bırakılır. Basılan tuşa ilişkin bilgiler mesajın wParam ve lParam parametrelerine geçirilir.
WM_KEYDOWN ve WM_KEYUP mesajlarının parametreleri şöyledir:

wParam: basılan tuşun sanal tuş kodu
lParam: bu parametre bitsel bir biçimde yorumlanır.
- [0 - 15] Bitleri: Bu bitler, lParam parametresinin düşük anlamlı 2 byte'ıdır. Burada tuşa basım sayısı bulundurulur. Windows, aynı tuşa birden fazla kez basıldığında bu mesajları ayrı WM_KEYDOWN mesajları olarak kuyruğa bırakabildiği gibi tek bir WM_KEYDOWN mesajı olarak bırakıp bu sayacı arttırabilmektedir. Her ne kadar böyle bir durumun gerçekleşmesi çok seyrek olsa da programcı bu duruma dikkat etmelidir.
- [16 - 23] Bitleri: Bu, lParam'ın düşük anlamlı 3. byte'ıdır. Burada o sisteme ilişkin doğal tarama kodu (scan code) bulunur.
- [24 - 31] Bitleri: Bu bitler burada ele alınmayacaktır. Dökümanlardan izlenebilir.

Sanal Tuş Kodları

Klavye mesajlarının wParam parametresine sistem bağımsız sanal tuş kodları bilgisi yerleştirilir. Sanal tuş kodları, windows.h içerisinde **VK_XXX** sembolik sabitleriyle belirlenmiştir. Programcı, tuşun ne tuşu olduğuna bu kodlara bakarak karar vermelidir. Çeşitli tuşlar şunlardır:

VK_BACK: Backspace
VK_TAB: Tab
VK_RETURN: return
VK_ESCAPE: Escape
VK_PRIOR
VK_PAGEUP ????
VK_NEXT
VK_PAGEDOWN ????
VK_END: End
VK_HOME: Home
VK_LEFT : Sola ok tuşu
VK_RIGHT: Sağa ok tuşu
VK_INSERT: insert
VK_DELETE: delete

ASCII tuşlarının sanal tuş kodları tek tek define edilmemiştir. Onların tuş kodları, büyük harf karşılıklarının ASCII kodlarına eşittir. genericKey1, 2.c.....

```
/* genericKey1.c */

#include <windows.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
```

```
                    int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message) {
                case WM_KEYDOWN:
                        if (wParam == VK_RETURN) {
                                MessageBox(hWnd, "Enter", "Message", MB_OK);

                        }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
/* generickey2.c */

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message) {
                case WM_KEYDOWN:
                        if (wParam == VK_SHIFT) {
                                MessageBox(hWnd, "shift", "Message", MB_OK);

                        }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

Görüldüğü gibi önce Shift sonra a tuşuna basılmış olsa; sonra el bu tuşlardan çekilmiş olsa önce shift tuşu için WM_KEYDOWN, sonra a tuşu için WM_KEYDOWN; sonra çekilme sırasına göre WM_KEYUP mesajları gelir. Programcının, shift durumunu anlaması için özel bayraklar tutması gerekebilir.

WW_SYSKEYDOWN ve WM_SYSKEYUP Mesajları

Bu mesajlar, sistem tuşu diye adlandırılan "Alt" tuşuna basılınca gönderilmektedir. Mesajların parametrik yapıları, yukarıdaki mesajların parametrik yapılarına benzerdir.

GetKeyState Fonksiyonu:

Bu fonksiyon, bir tuşa basıldığındaki klavye durumu hakkında bilgi verir. Bu fonksiyon, çağırıldığı andaki klavye durumunu anlamak için değil o andaki *klavye mesajına ilişkin durumu* anlamakta kullanılır.

```
SHORT GetKeyState(
  int nVirtKey   // virtual-key code
);
```

Fonksiyonun parametresi, durumu öğrenilecek sanal tuş kodudur. Fonksiyonun geri dönüş değerinin en yüksek anlamlı biti, soruşturulacak tuşun mesajın oluştuğu anda basık durumda mı yoksa çekik durumda mı olduğunu anlamak için kullanılır. Eğer basık durumda ise bu bit 1'dir yani geri dönüş değeri negatiftir; değilse bu bit 0'dır yani geri dönüş değeri pozitiftir. En düşük anlamlı biti CapsLock, NumLock gibi tuşların durumunu belirlemekte kullanılır.

Örneğin bir alfabetik karakter tuşuna basıldığında bunun büyük harf mi küçük harf mi olduğunu anlamak için Shift tuşunun durumuna bakmak gerekir. Bu işlem şöyle yapılabilir:

```
case WM_KEYDOWN:
        if (wParam >= 'A' && wParam <= 'Z'){
                if (GetKeyState(VK_SHIFT) < 0)     /* basılı */
                        asciicode = wParam;
                else
                        asciicode = wParam – 'A' + 'a';
                ....
```

```
	}
```

/* generickey3.c */

```
#include <windows.h>


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	char s[2];

	switch (message) {
		case WM_KEYDOWN:
			if (wParam >= 'A' && wParam <= 'Z') {
				s[0]= (GetKeyState(VK_SHIFT) < 0)  ? wParam : wParam - 'A' + 'a';
```

```
                        s[1] = '\0';
                        MessageBox(hWnd, s, "Message", MB_OK);
                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

GetAsyncKeyState Fonksiyonu

Bu fonksiyon, çağırıldığı andaki klavyenin durumuna bakar.

```
SHORT GetAsyncKeyState(
 int vKey // virtual-key code
);
```

Fonksiyonun parametresi ve geri dönüş değeri, GetKeyState fonksiyonu ile aynıdır. Yalnızca bu fonksiyon, mesajın bırakıldığı andaki değil fonksiyonun çağırıldığı andaki klavye durumuna bakar.

WM_CHAR Mesajı

WM_CHAR mesajı daha kolay işlenebilen, biraz daha yüksek seviyeli bir mesajdır. Bu mesaj yalnızca ASCII tuşlarına basıldığında gönderilir. Aslında mesajın gönderilmesi tamamen mesaj döngüsüne yerleştirilen TranslateMessage ile yapılmaktadır. TranslateMessage, shift tuşlarının durumunu kendisi kontrol ederek duruma göre WM_CHAR mesajını kuyruğa bırakır. Dolayısıyla bu mesajın kuyruğa bırakılması için mesaj döngüsünde TranslateMessage fonksiyonunun bulunması gerekir. Örneğin elimizi A tuşuna basıp çektiğimizi düşünelim. Önce basmadan dolayı bir WM_KEYDOWN mesajı kuyruğa bırakılır. Bu mesaj, GetMessage tarafından alınarak TranslateMessage fonksiyonuna verildiğinde TranslateMessage, basılan tuşun bir ASCII tuşu olduğunu anlar ve kuyruğa WM_CHAR mesajını bırakır. Bu olaylar muhtemelen çok hızlı gerçekleşecektir. Yani biz elimizi A tuşundan çektiğimiz zaman WM_CHAR mesajı kuyruğa bırakılmış olur. Bundan sonra Windows kuyruğa WM_KEYUP mesajını bırakır. Yani mesaj sırası şöyle olacaktır: WM_KEYDOWN, WM_CHAR, WM_KEYUP. Bak: generickey4.c, 4a.c -> font (karakter) büyüklüğünü otomatik alıyor

/* generickey4.c */

```
#include <windows.h>


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                   int nCmdShow)
{
    WNDCLASS wndClass;
    HWND hWnd;
    MSG message;

    if (!hPrevInstance) {
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.cbClsExtra = 0;
        wndClass.cbWndExtra = 0;
```

```
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static int y = 0;
	static char s[30];
	HDC hDC;
	static int lineHeight;
	TEXTMETRIC tm;

	switch (message) {
		case WM_CREATE:
			hDC = GetDC(hWnd);
			GetTextMetrics(hDC, &tm);
			lineHeight = tm.tmHeight;
			ReleaseDC(hWnd, hDC);
			break;
		case WM_KEYDOWN:
			hDC = GetDC(hWnd);
			sprintf(s, "WM_KEYDOWN");
			TextOut(hDC, 0, y, s, strlen(s));
			y += lineHeight;
			ReleaseDC(hWnd, hDC);
			break;
		case WM_KEYUP:
			hDC = GetDC(hWnd);
			sprintf(s, "WM_KEYUP");
			TextOut(hDC, 0, y, s, strlen(s));
			y += lineHeight;
			ReleaseDC(hWnd, hDC);
			break;
```

```
		case WM_CHAR:
			hDC = GetDC(hWnd);
			sprintf(s, "WM_CHAR");
			TextOut(hDC, 0, y, s, strlen(s));
			y += lineHeight;
			ReleaseDC(hWnd, hDC);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

/* generickey4a.c */

```
//....

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static int y = 0;
	static char s[30];
	HDC hDC;

	switch (message) {
		case WM_KEYDOWN:
			hDC = GetDC(hWnd);
			sprintf(s, "WM_KEYDOWN");
			TextOut(hDC, 0, y, s, strlen(s));
			y += 30;
			ReleaseDC(hWnd, hDC);
			break;
		case WM_KEYUP:
			hDC = GetDC(hWnd);
			sprintf(s, "WM_KEYUP");
			TextOut(hDC, 0, y, s, strlen(s));
			y += 30;
			ReleaseDC(hWnd, hDC);
			break;
		case WM_CHAR:
			hDC = GetDC(hWnd);
			sprintf(s, "WM_CHAR");
			TextOut(hDC, 0, y, s, strlen(s));
			y += 30;
			ReleaseDC(hWnd, hDC);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

WM_CHAR mesajının parametrik yapısı şöyledir:

wParam : Karakterin Unicode karşılığı
lParam : WM_KEYUP ve WM_KEYDOWN mesajlarındaki lParam ile aynıdır.

TranslateMessage fonksiyonu, her pencere için gönderilen WM_KEYDOWN mesajını WM_CHAR göndermek için işler. Örneğin standart edit kontrolü WM_CHAR kullanılarak yapılmıştır. Eğer biz mesaj döngüsünden TranslateMessage fonksiyonunu kaldırırsak EditBox kontrolüne de WM_CHAR mesajı gitmeyecek ve dolayısıyla bu kontrol de çalışmayacaktır.

WM_CHAR mesajı, shift tuşlarının durumu dikkate alınarak oluşturulmaktadır. Yani örneğin biz shift ve a tuşuna bassak shift tuşu için WM_CHAR oluşmaz. a tuşuna basıldığında TranslateMessage, A için WM_CHAR mesajını oluşturur.

Klavye Odağı

Windows, klavyeden tuşlara basıldığında hangi pencereye klavye mesajlarını göndermektedir? İşte bunu klavye odağı (keyboard focus) ile belirlemektedir. Klavye odağı, bir t anında tek bir pencerede olabilir. Klavye odağı hangi pencerede ise klavye mesajları o pencere için kuyruğa bırakılır.

Bir ana pencere fare ile yada başka bir yöntemle aktif hale getirildiğinde klavye odağı otomatik olarak o ana pencereye verilir. Ancak alt pencereye tıklandığında klavye odağı otomatik olarak alt pencereye Windows tarafından verilmez. alt pencere isterse fare mesajından hareketle klavye odağını kendisi almalıdır.
Klavye odağını ayarlamak için SetFocus fonksiyonu kullanılır.

```
HWND SetFocus(
 HWND hWnd // handle to window to receive focus
);
```

Fonksiyonun parametresi, odağın yerleştirileceği pencerenin handle değeridir. Fonksiyon, daha önce odağa sahip olan pencerenin handle değeri ile geri döner. SetFocus fonksiyonu ile ancak, fonksiyonu çağıran thread ile aynı thread tarafından yaratılmış bir pencereye odak geçilebilir. Yani başka bir deyişle biz bu fonksiyonla odağı başka bir programın penceresine geçiremeyiz.

Klavye odağı, SetFocus ile hangi pencereye geçirilmişse SetFocus fonksiyonu o pencereyi haberdar etmek için SendMessage yoluyla o pencereye WM_SETFOCUS mesajını gönderir. Benzer biçimde bu mesajı göndermeden önce klavye odağını kaybeden pencereye de WM_KILLFOCUS mesajını gönderecektir. Böylece odak alındığında yada kaybedildiğinde alt pencere kendi durumunu görüntüsel olarak yansıtma fırsatını elde eder. bak: generickey6.c, 7

```
/* generickey6.c */

#include <windows.h>


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                        int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
```

```
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "GenericChild";
		wndClass.lpfnWndProc = (WNDPROC) WndChildProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}


	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hChild;

	switch (message) {
		case WM_CREATE:
			hChild = CreateWindow("GenericChild", "",
			WS_CHILD|WS_VISIBLE|WS_BORDER, 100, 100, 200, 200,
					hWnd, (HMENU) 100,
					((LPCREATESTRUCT) lParam)->hInstance, NULL);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
```

```
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}


LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam)
{

    switch (message) {
        case WM_LBUTTONDOWN:
                SetFocus(hWnd);
                break;

        case WM_SETFOCUS:
        {
                HBRUSH hBrush;

                hBrush = CreateSolidBrush(RGB(rand() % 256, rand() % 256, rand() % 256));
                SetClassLong(hWnd, GCL_HBRBACKGROUND, (LONG) hBrush);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

/* generickey7.c */

```
//.....

LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{

    switch (message) {
        case WM_LBUTTONDOWN:
                SetFocus(hWnd);
                break;
        case WM_CHAR:
                MessageBox(hWnd, "Test", "Message", MB_OK);
                break;

        case WM_SETFOCUS:
        {
                HBRUSH hBrush;

                hBrush = CreateSolidBrush(RGB(rand() % 256, rand() % 256, rand() % 256));
                SetClassLong(hWnd, GCL_HBRBACKGROUND, (LONG) hBrush);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

AKTİF PENCERE KAVRAMI

Default olarak başlık kısmı mavi biçimde bulunan pencerelere "aktif pencereler" denir. Alt pencereler aktif pencere olamaz; yalnızca ana pencereler aktif pencere olabilir. Bir ana pencere, temel olarak 3 biçimde aktif hale getirilebilir:

1. Fare ile ana pencereye tıklayarak
2. Klavye ile Alt+Tab tuşları ile
3. API fonksiyonları ile

Aktif hale getirmek ile klavye odağı kavramları arasında 2 türlü ilişki vardır:

1. Bir ana pencere aktif hale getirildiğinde klavye odağı otomatik olarak o ana pencereye verilir.
2. Klavye odağı SetFocus fonksiyonu ile bir alt pencereye verildiğinde onun ana penceresi aktif hale gelir.

Aktif hale gelen ana pencere Z sırasına göre önce gözükür. Bir pencerenin aktif hale gelmesi ile Windows, o ana pencerenin ilişkin olduğu thread'in önceliğini birkaç quanta kadar 1 derece yükseltir. Böylece aktif hale getirilen pencerenin daha etkileşimli olması sağlanır.

Bir pencerenin programlama yoluyla aktif hale geçirilmesi birkaç biçimde olabilmektedir. CreateWindow fonksiyonu ile bir ana pencere yaratıldığında otomatik olarak pencere aktif hale getirilir. Böylece son çalıştırılan programın penceresi aktif halde görüntülenecektir. SetActiveWinow API fonksiyonu ile de bir pencere aktif hale getirilebilir.

```
HWND SetActiveWindow(
  HWND hWnd   // handle to window to activate
);
```

Fonksiyonun parametresi, aktif hale getirilecek pencerenin handle değeridir. Fonksiyon, aktif halini kaybeden pencerenin handle değeri ile geri döner. Pencere aktif hale gelmeden önce ve aktif olma durumunu kaybederken pencerelere WM_ACTIVATE mesajı gönderilir. Bu mesajla pencerenin aktif hale mi geldiği yoksa aktifliğini mi kaybettiği mesajın wParam parametresiyle anlaşılabilir. Eğer bu parametre WA_ACTIVE biçimindeyse pencere aktif hale gelmiştir; WA_INACTIVE biçimindeyse pencere aktifliğini kaybetmiştir. Eğer pencere SetActiveWindow fonksiyon ile değil de fare ile aktif hale getirilmişse wParam, WA_CLICKACTIVE değerini alır. bak: active1.c

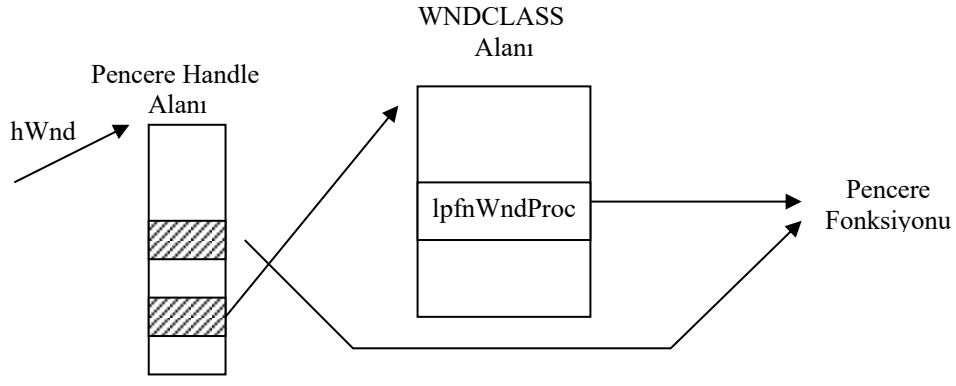
```
/* active1.c */
#include <windows.h>


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                        int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
```

```
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "GenericChild";
		wndClass.lpfnWndProc = (WNDPROC) WndChildProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static HWND hChild;

	switch (message) {
		case WM_CREATE:
			hChild = CreateWindow("GenericChild", "",
				WS_CHILD|WS_VISIBLE|WS_BORDER, 100, 100, 200, 200, hWnd,
				(HMENU) 100, ((LPCREATESTRUCT) lParam)->hInstance, NULL);
			break;
		case WM_ACTIVATE:
			if (wParam == WA_ACTIVE || wParam == WA_CLICKACTIVE) {
				ShowWindow(hWnd, SW_MAXIMIZE);
			}
```

```
                break;
            case WM_DESTROY:
                PostQuitMessage(0);
                break;
            default:
              return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

LRESULT CALLBACK WndChildProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam)
{

        switch (message) {
            case WM_LBUTTONDOWN:
                SetFocus(hWnd);
                break;
            case WM_CHAR:
                MessageBox(hWnd, "Test", "Message", MB_OK);
                break;

            case WM_SETFOCUS:
            {
                HBRUSH hBrush;

                hBrush = CreateSolidBrush(RGB(rand() % 256, rand() % 256, rand() % 256));
                SetClassLong(hWnd, GCL_HBRBACKGROUND, (LONG) hBrush);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

PENCERE FONKSİYONUNUN KANCALANMASI

Bu konuya İngilizce "Window Subclassing" denilmektedir. Bir pencere sınıfı RegisterClass API fonksiyonu ile register ettirildiğinde sistem, WNDCLASS yapısını kendi alanında bir yerde saklar. Sonra bu pencere sınıfından bir pencere yaratıldığında yaratılan pencerenin handle alanında bu pencerenin hangi WNDCLASS yapısı kullanılarak yaratıldığına ilişkin o WNDCLASS yapısını gösteren bir gösterici tutar. Yani hWnd değeri bilindiğinde sistem, pencerenin yaratıldığı WNDCLASS yapısına kolaylıkla erişmektedir. Pencere yaratıldığında ayrıca pencere fonksiyonunun adresi WNDCLASS yapısından alınarak pencere handle alanına da kopyalanmaktadır. Pencere yaratıldığında durum aşağıdaki gibi olmaktadır:



SendMessage ve DispatchMessage fonksiyonları pencere fonksiyonunun adresinin WNDCLASS alanından değil pencere handle alanından almaktadır.
Window Subclassing, bir pencerenin pencere fonksiyonu yerine başka bir fonksiyonun yerleştirilmesi anlamına gelir. 3 biçimde yapılabilir:

1. Pencere handle alanındaki pencere fonksiyonunun adresinin değiştirilmesi: Böyle bir işlem, SendMessage ve DispatchMessage işlemleri için yalnızca o pencerenin etkilenmesini sağlar.
2. WNDCLASS yapısındaki pencere pencerenin adresinin değiştirilmesi: Bu durumda artık bu pencere sınıfı kullanılarak yaratılacak pencerelerin pencere fonksiyonları değiştirilmiş olur.
3. Daha önce yaratılmış pencerelerin Fonksiyonlarının ve WNDCLASS yapısının adresinin değiştirilmesi: Bu durumda bu işlem, hem daha önce yaratılmış olan pencereler hem de yeni yaratılacak pencereler etkilenir.

Pencerenin Handle Alanındaki Pencere Fonksiyonunun Adresinin Değiştirilmesi

Bu işlem, SetWindowLong fonksiyonu ile yapılabilir. Bu fonksiyonun ikinci parametresi GWL_WNDPROC olarak girilir; 3. parametresi de pencere fonksiyonunun adresi olarak girilirse handle alanındaki pencere fonksiyonunun adresi değiştirilebilir. Benzer biçimde pencere fonksiyonunun adresini de biz GetWindowLong fonksiyonu ile elde edebiliriz.

Window subclassing'den amaç pencerenin pencere fonksiyonunun tamamen değiştirmek değil araya girmektir. Bunun için önce eski pencere fonksiyonunun GetWindowLong fonksiyonu ile alınması, sonra yeni pencere fonksiyonunun yerleştirilmesi gerekir.

Yeni Pencere Fonksiyonunda Eski Pencere Fonksiyonunun Çağırılması

Yeni pencere fonksiyonu içerisinde eski pencere fonksiyonunun çağırılması doğrudan fonksiyon göstericisi yoluyla değil de CallWindowProc fonksiyonu ile yapılmalıdır.

```
LRESULT CallWindowProc(
 WNDPROC lpPrevWndFunc,      // pointer to previous procedure
 HWND hWnd,                  // handle to window
 UINT Msg,                   // message
 WPARAM wParam,              // first message parameter
 LPARAM lParam               // second message parameter
);
```

WNDPROC, aşağıdaki gibi bir typedef ismidir:

typedef LRESULT (CALLBACK *WNDPROC) (HWND, UINT, WPARAM, LPARAM)

bak: subclassing1.c -> basılan karakterden 1 sonraki çıkıyor.
bak: subclassing2.c -> alfabetik hangi karaktere basılırsa basılsın a çıkıyor.
bak: subclassing3.c -> basılan karakterin büyük halini basıyor.
bak: subclassing4.c -> noktadan sonra basılan karakterin büyük halini basıyor.
bak: subclassing5.c -> checkbox'a basıldığında orayı boyuyor.

```
/* subclassing1.c */

#include <windows.h>
#include <ctype.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
```

```
LRESULT CALLBACK NewWndProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam);
void SubClass(HWND hWnd);

WNDPROC pOldWndProc;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                        int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hEdit;

        switch (message) {
                case WM_CREATE:
                        hEdit = CreateWindow("edit", "",
                                        WS_CHILD|WS_VISIBLE|WS_BORDER|ES_MULTILINE,
                                        100, 100, 200, 200, hWnd, (HMENU) 100,
                                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);

                        if (hEdit == NULL)
```

```
                        return -1;
                    SetFocus(hEdit);
                    SubClass(hEdit);
                    break;

            case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
            default:
              return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void SubClass(HWND hWnd)
{
        pOldWndProc = (WNDPROC) SetWindowLong(hWnd, GWL_WNDPROC, (LONG) NewWndProc);

        if (pOldWndProc == NULL) {
                MessageBox(NULL, "Cannot get old Wndproc!..", "Message", MB_OK);
                return;
        }
}

LRESULT CALLBACK NewWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{

        switch (message) {
                case WM_CHAR:
                        if (isalpha(wParam))
                                wParam++;
                        break;
        }

        return CallWindowProc(pOldWndProc, hWnd, message, wParam, lParam);
}
```

/* subclassing2.c */

```
//.....

LRESULT CALLBACK NewWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{

        switch (message) {
                case WM_CHAR:
                        if (isalpha(wParam))
                                wParam = 'a';
                        break;
        }

        return CallWindowProc(pOldWndProc, hWnd, message, wParam, lParam);
}
```

/*subclassing3.c */

```
//...
LRESULT CALLBACK NewWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{

        switch (message) {
                case WM_CHAR:
                        wParam = toupper(wParam);
```

```
                break;
        }

        return CallWindowProc(pOldWndProc, hWnd, message, wParam, lParam);
}
```

/* subclassing4.c*/

```
LRESULT CALLBACK NewWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static BOOL flag;

        switch (message) {
                case WM_CHAR:
                        if (wParam == '.')
                                flag = TRUE;
                        else if (isalpha(wParam) && flag) {
                                wParam = toupper(wParam);
                                flag = FALSE;
                        }

                        break;
        }

        return CallWindowProc(pOldWndProc, hWnd, message, wParam, lParam);
}
```

/* subclassing5.c*/

```
//...
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hEdit, hCheckBox;

        switch (message) {
                case WM_CREATE:
                        hEdit = CreateWindow("edit", "",
                                        WS_CHILD|WS_VISIBLE|WS_BORDER|ES_MULTILINE,
                                        100, 100, 200, 200, hWnd, (HMENU) 100,
                                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);
                        hCheckBox = CreateWindow("button", "Test",
                                        WS_CHILD|WS_VISIBLE|BS_CHECKBOX, 10, 10, 50, 40,
                                        hWnd, (HMENU) 100,
                                        ((LPCREATESTRUCT) lParam)->hInstance, NULL);

                        if (hEdit == NULL)
                                return -1;
                        if (hCheckBox == NULL)
                                return -1;
                        SubClass(hCheckBox);

                        break;

                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
void SubClass(HWND hWnd)
{
	pOldWndProc = (WNDPROC) SetWindowLong(hWnd, GWL_WNDPROC, (LONG) NewWndProc);

	if (pOldWndProc == NULL) {
		MessageBox(NULL, "Cannot get old Wndproc!..", "Message", MB_OK);
		return;
	}
}

LRESULT CALLBACK NewWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static BOOL bFlag = FALSE;
	static HBRUSH hRedBrush, hWhiteBrush;

	switch (message) {
		case WM_CREATE:
			hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
			hWhiteBrush = CreateSolidBrush(RGB(255, 255, 255));
			break;
		case WM_LBUTTONDOWN:
		{
			HDC hDC;
			RECT rect;
			char text[30];

			bFlag = !bFlag;
			hDC = GetDC(hWnd);
			GetClientRect(hWnd, &rect);

			GetWindowText(hWnd, text, 30);
			SetBkColor(hDC, bFlag ? RGB(255, 0, 0) : RGB(255, 255, 255));
			DrawText(hDC, text, -1, &rect, DT_RIGHT|DT_VCENTER|DT_SINGLELINE);


			ReleaseDC(hWnd, hDC);

		}


			break;
		case WM_DESTROY:
			DeleteObject(hRedBrush);
			DeleteObject(hWhiteBrush);
			break;
	}

	return CallWindowProc(pOldWndProc, hWnd, message, wParam, lParam);
}
```

Pencere Fonksiyonunun Kancalanmasına Neden Gereksinim Duyulur?

Şüphesiz kendi yazdığımız pencere fonksiyonunu kancalamamıza gerek yoktur. Fakat başkaları tarafından yazılmış olan pencere fonksiyonları kancalanabilir. Örneğin standart kontrollerin pencere fonksiyonlarını kancalayarak faydalı bir takım işlemler yapabiliriz. Bir edit kontrolünü isteğimize uygun bir biçimde bu yöntemle değiştirebiliriz. Bir listbox kontrolündeki girişlerin sayısını sınırlandırabiliriz. Tabi şüphesiz bu yöntemle standart kontrollerin tam olarak isteğimize uygun hale getirilmesi mümkün olmayabilir. Bu durumda isteğimize uygun bir kontrolü tamamen sıfırdan yazmak gerekebilir.

Window subclassing yöntemi ile başka bir programın pencere fonksiyonunu kancalayamayız. Çünkü en azından o başka programın pencere fonksiyonu o programın bellek alanı içerisindedir. O programda o pencere fonksiyonunun adresini alsak bile bizim programımızda bunun bir anlamı olmayabilir.

Başka proseslere gönderilen mesajların ele geçirilmesi için daha yetenekli bir mekanizma kullanılmaktadır. Bu konuya genel olarak İngilizce "Windows Hooking" denilmektedir. Örneğin herhangi bir programdaki yazının üzerine click yapıldığında sözlük programının aktive olarak click yapılan sözcük hakkında açıklama göstermesi gibi tipik bir örnek bu mekanizma işle yapılmaktadır. Windows hook işlemi için SetWindowsHook API fonksiyonu kullanılır.

Pencere Sınıfında Pencere Fonksiyonunun Kancalama İşlemi

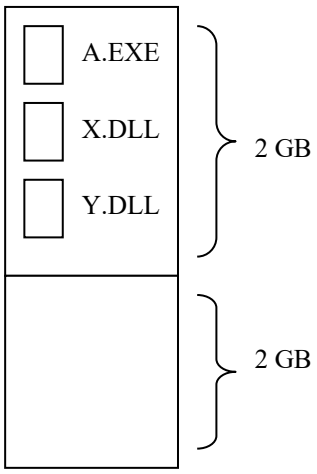
WNDCLASS alanındaki pencere fonksiyonunun adresini değiştirmek ve bu adresi elde etmek için SetClassLong ve GetClassLong API fonksiyonları kullanılmaktadır.

SetClassLong fonksiyonunda 2. parametre GCL_WNDPROC biçiminde girilirse pencere fonksiyonu değiştirilebilir. Bu biçimde pencere fonksiyonu değiştirildiğinde o zamana kadar yaratılmış olan pencereler bu işlemden etkilenmez. Fakat bu işlemden sonra bu sınıf kullanılarak yaratılacak olan pencereler etkilenir.

```
DWORD SetClassLong(
 HWND hWnd,              // handle of window
 int nIndex,             // index of value to change
 LONG dwNewLong          // new value
);
```

DİNAMİK KÜTÜPHANELERİN KULLANIMI

Dinamik kütüphaneden bir çağırma yapıldığında linker, çağırılan fonksiyonu çalıştırılabilen dosyaya yazmaz. Yalnızca PE formatının import tablosu denilen kısmına hangi DLL'deki hangi fonksiyonun çağırıldığı bilgisini yazar. Program çalıştırılmak istenildiğinde yükleyici, çalıştırılabilen dosyanın import tablosunu inceler, bu exe programın hangi DLL'leri kullandığını tespit eder. Bu DLL'leri de exe programlarla birlikte bütünsel 2GB'lik kullanıcı alanına yükler. Örneğin a.exe dosyası, x.dll ve y.dll kütüphanelerini kullanıyor olsun. Yükleme sonrasında aşağıdaki gibi bir durum oluşacaktır:



DLL'den tek bir fonksiyon çağırılıyor olsa bile dll'in tamamı bütünsel olarak yüklenmektedir. Yükleyicinin prosese ilişkin herhangi bir dll'i bulamaması durumunda yükleme işlemi başarısızlıkla sonuçlanır ve program çalışmaz.

Bir prosesin parçalarını oluşturan .EXE ve .DLL dosyalarının her birine "modül" denilmektedir.

EXE dosya ile DLL dosya arasında format bakımından bir fark yoktur. Bu dosyaların her ikisi de PE formatına uygundur. EXE dosyanın DLL'den tek farkı, bir başlangıç noktası içermesidir.

DİNAMİK KÜTÜPHANELERİN OLUŞTURULMASI

Dinamik kütüphanelerin fiziksel olarak oluşturulması oldukça kolaydır. Tek yapılacak şey, bir linker seçeneği ile linker'ın exe değil de dll üretmesini sağlamaktır. Visual C++ derleyici sisteminde bu işlem, new/project menüsünden "win32 dynamic link library" seçilerek yapılabilir.

Dinamik Kütüphanedeki Fonksiyonların Çağırılması Ve PE Formatının Import Ve Export Tabloları

Makine dilinde bir fonksiyonun çağırılması işlemi CALL makine komutu ile yapılmaktadır. CALL makine komutunun doğrudan (direct) ve dolaylı (indirect) olmak üzere 2 biçimi vardır. CALL makine komutunun operandı, fonksiyonun bellek adresidir. Örneğin: CALL 1FC30000

Dolaylı (indirect) CALL komutunda operand, bir bellek adresidir fakat o bellek adresi fonksiyonun bulunduğu bellek adresi değil, fonksiyonun bulunduğu adresi tutan bellek bölgesinin adresidir. Örneğin: CALL [1F358000] : Burada işlemci, 1F358000 adresinden 4 byte çeker ve oradaki fonksiyona dallanır.

Statik kütüphaneden çağırma yapıldığında derleyici ve linker bir işbirliği içinde çağırılan fonksiyonun program yüklendiğindeki yerini tam olarak tespit edebilir. Link edilmiş program bu nedenle doğrudan bir CALL komutu içermektedir.

Anahtar Notlar:

Linker, EXE dosyanın nereye yükleneceğini bilerek kodu üretmektedir. Bu durumda linker, statik kütüphane içerisindeki fonksiyonu EXE dosyasına kendisi yazdığına göre o fonksiyonun EXE dosyanın neresinde olduğunu bilecektir. O halde EXE dosyanın yükleme adresi ile bu değerin toplanması, fonksiyonun gerçek bellek adresini verir. Win32 yükleyicileri, programın nereye yükleneceğine PE formatına bakarak karar vermektedir.

Bir DLL'den fonksiyon çağırıldığında fonksiyonun adresi link işlemi sonrasında bile tespit edilemez. Çünkü DLL'lerin yükleyici tarafından nereye yükleneceği çeşitli nedenlerle değişebilmektedir ve yükleme öncesinde bu bilgi kesin olarak bilinemez. Derleyici, bir DLL'den fonksiyon çağırıldığında fonksiyon için import tablosuna referans eden dolaylı CALL kullanır. Import tablosunun formatı, aşağıdakine benzer biçimdedir:

IMPORT TABLOSU
X.DLL

İsim	Adres

CALL [????????]

Func ...	???????? ...

Y.DLL

İsim	Adres
Sample ...	???????? ...

Görüldüğü gibi import tablosunda DLL içerisindeki fonksiyonların ismi ve adresleri bulunmaktadır. Import tablosundaki adres sütunu, link işlemi sonrasında bile doldurulmamış durumdadır. Bu adres sütunu, yükleyici tarafından doldurulmaktadır.

PE formatının export tablosu da aşağıdakine benzer yapıdadır:

EXPORT TABLOSU

Fonksiyon İsmi	Offset
Func ...	00130018 ...

Görüldüğü gibi export tablosunda fonksiyonların PE formatının başından itibaren kaçıncı byte'da olduğu bilgisi vardır. Yükleyici, EXE dosyanın import tablosunu inceler; oradaki fonksiyonları DLL'lerdeki export tablolarında arar. DLL'leri nereye yüklediğini de bildiğine göre DLL içerisindeki fonksiyonların net adreslerini tespit eder ve bu adreslerle import tablosunu doldurur. Artık herşey yerine oturmuştur.

Görüldüğü gibi EXE dosyası import yoğun, DLL dosyaları da export yoğundur. Fakat her iki dosya da PE formatına uygundur. Yani her iki dosyanın da import ve export tabloları vardır. Örneğin bir DLL'in başka bir DLL'den fonksiyon çağırdığını düşünelim. Bu durumda bu DLL'in hem import hem de export tabloları doludur. Şüphesiz yükleyici, bu DLL'in kullandığı DLL'i de recursive bir biçimde belleğe yükleyecektir. (__declspec(dllexport) void Func(void) ---> fonksiyon linker tarafından export tablosuna yazılır. Fonksiyonun prototipinde yada tanımlamasında yazılır. declspec, anahtar sözcüktür.)

Derleyici ve linker, her fonksiyonu export tablosuna yazmaz. Bir fonksiyonun export tablosuna yazılması için dllexport bildiriminin yapılmış olması gerekir. dllexport bildirimi, fonksiyonun prototipi yada tanımlaması önüne aşağıdaki gibi getirilebilir:

__declspec(dllexport) .....fonksiyon.....

__declspec (decleration specification), Microsoft sistemleri için düşünülmüş standart olmayan bir anahtar sözcüktür. Özetle biz bir DLL yazarken dışarıdan çağırılmasını istediğimiz fonksiyonları dllexport bildirimi ile tanımlamamız gerekir.

Bir DLL içerisinden fonksiyon çağırırken aslında özel bir bildirimde bulunmaya gerek yoktur. Çünkü zaten linker fonksiyonu static kütüphanelerde bulamazsa dinamik kütüphanelerde arar ve fonksiyonun bir DLL fonksiyonu olduğunu tespit eder. Fakat dllimport bildirimi, bu tür durumda derleyicinin ve linker'ın işlemlerini kolaylaştırdığı için işlemleri hızlandırmaktadır. Yani biz bir DLL içerisinden fonksiyon çağıracaksak o fonksiyonun prototipinin önüne dllimport bildirimini yazmakta fayda vardır. Örneğin;

__declspec(dllimport) void Func(void);

dllimport bildirimi yalnızca fonksiyon prototipi önüne getirilebilir, tanımlama ifadesinde kullanılamaz.

DİNAMİK KÜTÜPHANELER İÇİN KAYNAK DOSYALARIN OLUŞTURULMASI

Bir DLL için fonksiyon yazarken fonksiyon prototiplerine hem DLL oluştururken hem de o DLL' i EXE'den kullanırken gereksinim duyulur. Bu durumda bu fonksiyonları bir başlık dosyası içerisine toplamak daha anlamlıdır. Fakat DLL derlemesi yaparken fonksiyon prototiplerinin önünde dllexport bildirimi, EXE için derleme yaparken dllimport bildirimi bulunmalıdır. Bu işlem pratik olarak önişlemci komutlarıyla yapılabilir. Örneğin, MYDLL isminde bir DLL içerisinde Add ve Multiply isminde iki fonksiyon yerleştirecek olalım.

MYDLL.H
```
#ifndef  _MYDLL_H_
#define  _MYDLL_H_

#ifdef  DLLBUILD
#define DLLPROC  __declspec(dllexport)
#else
#define DLLPROC  __declspec(dllimport)
#endif

/*Function prototypes*/

DLLPROC int Add(int a, int b);
DLLPROC int Multiply(int a, int b);

#endif
```

DLL derlemesi yaparken DLLBUILD sembolik sabiti define edilmeli, EXE için derleme yaparken define edilmemelidir.

DLL DOSYASININ IMPORT KÜTÜPHANESİ

Bir DLL oluşturulurken linker DLL'in import kütüphanesi denilen bir .lib dosyası da oluşturur. Bu dosya normal bir statik kütüphane dosyası değildir. Bir DLL'i kullanacak olan programcı onun import kütüphanesini link aşamasında projeye dahil etmelidir. Import kütüphanesinin içerisinde DLL içerisinde bulunan fonksiyonlar hakkında bilgi vardır.

YÜKLEYİCİNİN DLL DOSYASINI ARAMASI

DLL'in yeri import tablosunda yazmaz. Bu nedenle yükleyici, DLL dosyalarını önceden belirlenmiş olan bazı dizinlerde aramaktadır. Bu dizinler sırasıyla şunlardır:

1. EXE dosyanın bulunduğu dizin.
2. O anda bulunulan geçerli dizin.
3. "Windows\System32" dizini.
4. "Windows" dizini
5. Path çevre değişkeniyle belirtilen dizinler.

DLL bu dizinlerden biri içinde değilse problem oluşur. (debug içindeki .dll'i exenin bulunduğu yere kopyala.)

C++' DA SINIFLARIN DLL İÇERİSİNE YERLEŞTİRİLMESİ

C++' da Sınıfın belirli üye fonksiyonları dllexport ve dllimport biçiminde bildirilebilir. Örneğin:

```
#ifndef DLLBUILD
#define DLLPROC __declspec(dllexport)
#else
#define DLLPROC __declspec(dllimport)
#endif

class Sample{
public:
        DLLPROC void Func1();
        void Func2();
        //...
};
```

Burada yalnızca Func1 fonksiyonu export tablosuna yazılmıştır. dolayısıyla biz DLL dışından yalnıza Func1 fonksiyonunu gerçek anlamda çağırabiliriz. Func2 fonksiyonunun çağırılması C++ kurallarına uygun olduğu halde DLL mekanizmasından dolayı link aşamasında problem oluşturacaktır.

Eğer dllexport ve dllimport bildirimleri class anahtar sözcüğü ile sınıf ismi arasına getirilirse sınıfın her bölümdeki tüm üye fonksiyonları için bu bildirim yapılmış kabul edilir. Örneğin;

```
class DLLPROC Sample{
public:
        void Func1();
        void Func2();
        //...
};
```

VISUAL C++ DLL WIZARD

DLL oluşturmayı kolaylaştırmak için yukarıdaki yapılanlara benzer işlemler yapan bir wizard vardır. Bunun için Project/New/Win32 Dynamic Link Library seçilir. Ondan sonra "A DLL that exports some symbols" radyo düğmesi işaretlenir.

DllMain FONKSİYONU

DLL içerisindeki DllMain isimli özel bir fonksiyon, çeşitli durumlarda sistem tarafından çağırılmaktadır. Bu fonksiyon aslında programcı tarafından yazılmak zorunda değildir. Fonksiyonu programcı yazmazsa link işlemi sırasında kütüphanedeki içi boş bir DllMain fonksiyonu DLL dosyasına eklenir. Eğer DllMain fonksiyonu programcı tarafından yazılırsa kütüphanedeki fonksiyon devreye girmez. DllMain fonksiyonunun parametrik yapısı şöyledir:

```
BOOL WINAPI DllMain(
  HINSTANCE hinstDLL,    // handle to DLL module
  DWORD fdwReason,       // reason for calling function
  LPVOID lpvReserved     // reserved
);
```

Fonksiyonun 1. parametresi modülün belleğe yüklenme adresini belirtir. Fonksiyonun 2. parametresi DllMain fonksiyonunu sistemin hangi nedenle çağırdığını belirtir. Şunlardan bir olabilir:

DLL_PROCESS_ATTACH
DLL_THREAD_ATTACH
DLL_THREAD_DETACH
DLL_PROCESS_DETACH

Value	**Meaning**
DLL_PROCESS_ATTACH	Indicates that the DLL is being loaded into the virtual address space of the current process as a result of the process starting up or as a result of a call to **LoadLibrary**. DLLs can use this opportunity to initialize any instance data or to use the **TlsAlloc** function to allocate a thread local storage (TLS) index.
DLL_THREAD_ATTACH	Indicates that the current process is creating a new thread. When this occurs, the system calls the entry-point function of all DLLs currently attached to the process. The call is made in the context of the new thread. DLLs can use this opportunity to initialize a TLS slot for the thread. A thread calling the DLL entry-point function with DLL_PROCESS_ATTACH does not call the DLL entry-point function with DLL_THREAD_ATTACH. Note that a DLL's entry-point function is called with this value only by threads created after the DLL is loaded by the process. When a DLL is loaded using **LoadLibrary**, existing threads do not call the entry-point function of the newly loaded DLL.
DLL_THREAD_DETACH	Indicates that a thread is exiting cleanly. If the DLL has stored a pointer to allocated memory in a TLS slot, it uses this opportunity to free the memory. The system calls the entry-point function of all currently loaded DLLs with this value. The call is made in the context of the exiting thread.
DLL_PROCESS_DETACH	Indicates that the DLL is being unloaded from the virtual address space of the calling process as a result of either a process exit or a call to **FreeLibrary**. The DLL can use this opportunity to call the **TlsFree** function to free any TLS indices allocated by using **TlsAlloc** and to free any thread local data.

DllMain, DLL prosesin adres alanına yüklendiğinde yalnızca bir kez DLL_PROCESS_ATTACH koduyla çağırılır. Benzer biçimde DLL, prosesin adres alanından boşlatılacağı zaman son kez DllMain, DLL_PROCESS_DETACH koduyla çağırılacaktır.

Bir proses içerisinde ne zaman bir thread yaratılsa sistem, o prosesin kullandığı tüm thread'ler için tek tek DllMain fonksiyonunu DLL_THRERAD_ATTACH koduyla çağırır. Böylelikle DLL'ler, yeni bir thread'in yaratılmış olduğunu anlayacaktır. Benzer biçimde bir thread sonlandığında da tüm DLL'ler için tek tek DllMain DLL_THREAD_DETACH koduyla çağırılır. DllMain fonksiyonu yaratılmış olan ve yok edilecek thread akışları tarafından çağırılmaktadır. Bak: dll dosyası

/* mydll.c */

```
#include <windows.h>
#include <stdio.h>
#include "mydll.h"

int Add(int a, int b)
{
	return a + b;
}

int Multiply(int a, int b)
{
	return a * b;
}

BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID reserved)
{
	switch (dwReason) {
		case DLL_PROCESS_ATTACH:
			MessageBox(NULL, "Process attach", "Message", MB_OK);
			break;
		case DLL_PROCESS_DETACH:
			MessageBox(NULL, "Process detach", "Message", MB_OK);
			break;
		case DLL_THREAD_ATTACH:
			MessageBox(NULL, "Thread attach", "Message", MB_OK);
			break;
		case DLL_THREAD_DETACH:
			MessageBox(NULL, "Thread detach", "Message", MB_OK);
			break;
	}

	return TRUE;
}

/* test.c */

#include <windows.h>
#include <stdio.h>
#include "mydll.h"

DWORD CALLBACK ThreadFunc(LPVOID pParam)
{
	int i;

	for (i = 0; i < 10; ++i) {
		printf("%d\n", i);
		Sleep(1000);
	}

	return 0;
}
```

```
int main(void)
{
	HANDLE hThread;
	DWORD dwThreadID;

	printf("%d\n", Add(10, 10));
	printf("%d\n", Multiply(10, 20));

	hThread = CreateThread(NULL, 0, ThreadFunc, 0, 0, &dwThreadID);
	if (hThread == NULL) {
		fprintf(stderr, "Cannot create thread!..\n");
		exit(EXIT_FAILURE);
	}

	WaitForSingleObject(hThread, INFINITE);

	return 0;
}
```

DLL'LER VE KAYNAK KULLANIMI

DLL dosyaları da PE formatına uygun dosyalardır. Yani onların da kaynak bölümleri vardır. Hatta bazı programcılar programın tüm kaynaklarını bir DLL dosyası içerisine yerleştirip oradan kullanmaktadır. Yani bu programcılar DLL'leri sadece kaynakları saklamak için kullanmaktadır (resource only DLL).

Kaynak yükleyen LoadXXX fonksiyonları, PE formatının başlangıç adresi olan hInstance değerini parametre olarak istemektedir. Bu durumda bizim DLL modülünün hInstance değerini bu fonksiyonlara parametre olarak vermemiz gerekir. Bir DLL modülünün yükleme adresi olan hInstance değeri, GetModuleHandle fonksiyonu ile elde edilebilir.

```
HMODULE GetModuleHandle(
 LPCTSTR lpModuleName  // address of module name to return handle for
);
```

Fonksiyon parametre olarak modül dosyasının ismini alır (eğer dosyanın uzantısı yazılmazsa uzantının .dll olduğu varsayılır). Fonksiyonun geri dönüş değeri, DLL'in yükleme adresidir. HMODULE kavramının HINSTANCE kavramından bir farkı yoktur. Eğer parametre olarak NULL geçilirse fonksiyon, EXE dosyanın hInstance değerini verir.

DİNAMİK KÜTÜPHANELERİN DİNAMİK YÜKLENMESİ

Dinamik kütüphaneler istenirse programın çalışmasının belli bir anında LoadLibrary isimli API fonksiyonuyla dinamik olarak yüklenebilirler ve FreeLibrary fonksiyonuyla boşaltılabilirler. Bu yükleme biçimi çok seyrek kullanılmaktadır.

```
HINSTANCE LoadLibrary(
  LPCTSTR lpLibFileName   // address of filename of executable module
);
```

```
BOOL FreeLibrary(
 HMODULE hLibModule // handle to loaded library module
```

```
);
```

LoadLibrary fonksiyonu, dinamik kütüphane dosyasının ismini parametre olarak alır. Eğer dosya ismi \ karakteri kullanılarak bir path ifadesi biçiminde yazılırsa dosya yalnızca belirtilen dizinde aranır. Eğer dosya ismi yalın olarak yazılırsa dosya daha önce belirtilen dizinlerde otomatik olarak aranır. LoadLibrary fonksiyonu başarı durumunda yüklenen DLL dosyasının hInstance değeri ile geri döner. FreeLibrary fonksiyonu ise DLL'in yükleme adresini parametre olarak alıp boşaltma işlemi yapar.

Bir DLL dinamik olarak yüklendikten sonra içerisindeki kaynaklar doğrudan kullanılabilir yada export tablosunda belirtilmiş olan fonksiyonların adresleri elde edilerek o fonksiyonlar çağırılabilir. Bunun için GetProcAddress fonksiyonu kullanılmaktadır.

```
FARPROC GetProcAddress(
 HMODULE hModule,       // handle to DLL module
 LPCSTR lpProcName      // name of function
);
```

Fonksiyonun 1. parametresi modülün yüklenme adresi, 2. parametresi modül içerisinde export edilmiş fonksiyonun ismidir. İsim yerine fonksiyonun sıra numarası (ordinal number) verilebilir ama burada ele alınmayacaktır. Fonksiyonun geri dönüş değeri FarProc ile temsil edilen, geri dönüş değeri ve parametresi void olan bir fonksiyon adresidir. Programcı, fonksiyonun geri dönüş değerini uygun fonksiyon göstericisi türüne dönüştürerek atamayı yapabilir. Fonksiyon başarısız olursa NULL değerine geri dönmektedir. Bu işlemler için typedef bildirimini kullanmak kolaylık sağlayabilir. Örneğin biz sample.dll isimli bir dll'in içerisinde iki sayının toplamıyla geri dönen add isimli fonksiyonun export edildiğini biliyor olalım. Bu fonksiyonu şöyle çağırabiliriz:Örneğin:

```
HINSTANCE hModule;
int (*pAdd) (int, int);

hModule = LoadLibrary("sample.dll");
if (hModule == NULL) {
        ...
}
pAdd = (int (*) (int, int)) GetProcAddress(hModule, "Add");
if (pAdd == NULL) {
        ....
}
```

typedef bildirimiyle işlemler aşağıdaki gibi yapılabilir:

```
typedef int (*PFUNC) (int, int);
pFunc pAdd;
HINSTANCE hModule;
...

pAdd = (pFunc) GetProcAddress (hModule, "Add");
```

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include "mydll.h"
```

```
typedef int (*PFUNC)(int, int);

int main(void)
{
	HINSTANCE hModule;
	PFUNC pAdd, pMultiply;

	if ((hModule = LoadLibrary("..\\mydll\\debug\\mydll.dll")) == NULL) {
		fprintf(stderr, "Cannot load library!..\n");
		exit(EXIT_FAILURE);
	}

	pAdd = (PFUNC) GetProcAddress(hModule, "Add");
	if (pAdd == NULL) {
		fprintf(stderr, "Cannot get address!..\n");
		exit(EXIT_FAILURE);
	}
	pMultiply = (PFUNC) GetProcAddress(hModule, "Multiply");
	if (pAdd == NULL) {
		fprintf(stderr, "Cannot get address!..\n");
		exit(EXIT_FAILURE);
	}

	printf("%d - %d\n", pAdd(100, 200), pMultiply(100, 200));

	FreeLibrary(hModule);


	return 0;
}
 // ama burada compiler'dan statik library'yi eklemeyeceksin
```

LoadLibrary fonksiyonu yalnızca DLL yüklemek için değil, normal bir exe modülünü de dll'miş gibi yüklemek için kullanılabilmektedir.

VISUAL C 6.0 İSKELET API WISARD PROGRAMI

Visual C 6.0 ve .NET derleyicilerinde iskelet API fonksiyonunu yazan bir wisard vardır. Fakat maalesef bu wisard, kötü bir programlama tekniğiyle oluşturulmuştur. Fakat hızlı denemelerde kullanılabilir. Wisard için File/New/Win32 application seçilir. Çıkan menüde "A typical Hello Word application" radyo düğmesi seçilir. Wisard, şu dosyaları üretmektedir:

1. Programın ismi X olmak üzere X.cpp ve X.h dosyaları. Bu dosya, iskelet programı içeren dosyalardır.
2. stdafx.cpp ve stdafx.h dosyaları. windows.h ve diğer bazı dosyalar stdafx.h dosyası içinde include edilmiştir. stdafx.h dosyası zaten x.cpp dosyasından da include edilmiştir. stdafx.cpp dosyasının içi boştur. Gerekirse bazı global tanımlamalar burada yapılabilir.
3. Projenin ismi X olmak üzere X.rc ve resource.h dosyaları.

İskelet API fonksiyonunda pencere başlık yazısı ve pencere sınıfının ismi, string kaynağı olarak kaynaktan alınmıştır. İskelet API programının küçük bir menüsü vardır. Help/About seçildiğinde küçük bir diyalog penceresi çıkartmaktadır. Ayrıca bazı kısayol tuşlarını da kullanmaktadır.

MENÜ İŞLEMLERİNİN AYRINTILARI

Menü elemanları CheckMenuItem fonksiyonu ile checked yada unchecked yapılabilir.

```
DWORD CheckMenuItem(
 HMENU hmenu,          // handle to menu
 UINT uIDCheckItem,    // menu item to check or uncheck
 UINT uCheck           // menu item flags
);
```

Fonksiyonun 1. parametresi, menünün handle değeridir (ana menünün handle değeri GetMenu ile alınabilir.). Fonksiyonun 2. parametresi,menü elemanının ID değeri yada pozisyon numarasını belirtir. Menü işlemleri genel olarak elemanların ID değerleri yada pozisyon numaraları ile yapılabilir. İşlemin ID değeri ile yapılacağını belirmek için fonksiyonun 3. parametresinde MF_BYCOMMAND yada MF_BYPOSITION belirlemesi yapılır. Eğer bu belirlemelerden hiçbiri yapılmamışsa default olarak MF_BYCOMMAND belirlemesinin yapıldığı kabul edilir. Fonksiyonun 3. parametresi, menü elemanı checked yapılacaksa MF_CHECKED, unchecked yapılacaksa MF_UNCHECKED biçiminde girilir. Yukarıda da belirtildiği gibi bu değerler ayrıca MF_BYCOMAND yada MF_BYPOSITION eklenebilir. Örneğin:

```
CheckMenuItem(GetMenu(hWnd), ID_FILE_OPEN,
                MF_BYCOMMAND | MF_CHECKED);
```

Menü elemanının pozisyon numarası en sondaki pop up menünün ilk elemanı 0 olmak üzere belirtilen ardışıl bir sayıdır. Fonksiyonun geri dönüş değeri MF_CHECKED yada MF_UNCHECKED biçiminde olan menü elemanının önceki değeridir.

```
#include <windows.h>
#include <list>
#include "resource.h"

using namespace std;

typedef struct tagPOINT_ELEM {
        POINT point;
        COLORREF color;
} POINT_ELEM;

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnFileSave(const list<POINT_ELEM> &pointList);
void OnFileOpen(list<POINT_ELEM> &pointList);


int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
```

```
		wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	HDC hDC;
	PAINTSTRUCT ps;
	static int prevX, prevY;
	static list<POINT_ELEM> pointList;
	static HPEN hPen;
	static COLORREF curColor;
	static DWORD checkedID = ID_COLOR_BLACK;
	static HMENU hMenu;
	POINT point;

	switch (message) {
		case WM_CREATE:
			curColor = RGB(0, 0, 0);
			hPen = CreatePen(PS_SOLID, 10, curColor);
			hMenu = GetMenu(hWnd);
			CheckMenuItem(hMenu, ID_COLOR_BLACK, MF_CHECKED);
			break;
		case WM_COMMAND:
			switch (LOWORD(wParam)) {
				case ID_FILE_SAVE_MENU:
					OnFileSave(pointList);
					break;
				case ID_FILE_OPEN_MENU:
					OnFileOpen(pointList);
					InvalidateRect(hWnd, NULL, TRUE);
					break;
				case ID_FILE_CLEAR_MENU:
					pointList.clear();
					InvalidateRect(hWnd, NULL, TRUE);
					break;
				case ID_COLOR_BLACK:
```

```
            DeleteObject(hPen);
            curColor = RGB(0, 0, 0);
            hPen = CreatePen(PS_SOLID, 10, curColor);
            CheckMenuItem(hMenu, checkedID, MF_UNCHECKED);
            CheckMenuItem(hMenu, ID_COLOR_BLACK, MF_CHECKED);
            checkedID = ID_COLOR_BLACK;
            break;
    case ID_COLOR_RED:
            DeleteObject(hPen);
            curColor = RGB(255, 0, 0);
            hPen = CreatePen(PS_SOLID, 10, curColor);
            CheckMenuItem(hMenu, checkedID, MF_UNCHECKED);
            CheckMenuItem(hMenu, ID_COLOR_RED, MF_CHECKED);
            checkedID = ID_COLOR_RED;
            break;
    case ID_COLOR_BLUE:
            DeleteObject(hPen);
            curColor = RGB(0, 0, 255);
            hPen = CreatePen(PS_SOLID, 10, curColor);
            CheckMenuItem(hMenu, checkedID, MF_UNCHECKED);
            CheckMenuItem(hMenu, ID_COLOR_BLUE, MF_CHECKED);
            checkedID = ID_COLOR_BLUE;
            break;
    case ID_COLOR_GREEN:
            DeleteObject(hPen);
            curColor = RGB(0, 255, 0);
            hPen = CreatePen(PS_SOLID, 10, curColor);
            CheckMenuItem(hMenu, checkedID, MF_UNCHECKED);
            CheckMenuItem(hMenu, ID_COLOR_GREEN, MF_CHECKED);
            checkedID = ID_COLOR_GREEN;

            break;
    case ID_COLOR_CUSTOM:
            {
                    CHOOSECOLOR cc = {sizeof(cc)};
                    COLORREF colors[16];
                    cc.hwndOwner = hWnd;
                    cc.hInstance = (HWND) GetWindowLong(hWnd,
                                    GWL_HINSTANCE);
                    cc.lpCustColors = colors;
                    ChooseColor(&cc);

                    DeleteObject(hPen);
                    hPen = CreatePen(PS_SOLID, 10, cc.rgbResult);
                    CheckMenuItem(hMenu, checkedID, MF_UNCHECKED);
                    CheckMenuItem(hMenu, ID_COLOR_CUSTOM,
                                    MF_CHECKED);
                    checkedID = ID_COLOR_CUSTOM;
            }
            break;
    case ID_BACKCOLOR_CUSTOM:
            {
                    CHOOSECOLOR cc = {sizeof(cc)};
                    HBRUSH hBrush;
                    COLORREF colors[16];

                    cc.hwndOwner = hWnd;
                    cc.hInstance = (HWND) GetWindowLong(hWnd,
                                    GWL_HINSTANCE);
                    cc.lpCustColors = colors;
                    ChooseColor(&cc);

                    hBrush = CreateSolidBrush(cc.rgbResult);
```

```
                                DeleteObject((HBRUSH) SetClassLong(hWnd,
                                        GCL_HBRBACKGROUND, (LONG) hBrush));
                                InvalidateRect(hWnd, NULL, TRUE);
                        }
                        break;

        }
        break;
case WM_LBUTTONDOWN:
        prevX = LOWORD(lParam);
        prevY = HIWORD(lParam);
        break;

case WM_MOUSEMOVE:
        {
                if (!(wParam & MK_LBUTTON))
                        break;
                point.x = prevX;
                point.y = prevY;

                POINT_ELEM pointElem;
                pointElem.point = point;
                pointElem.color = curColor;
                pointList.push_back(pointElem);

                hDC = GetDC(hWnd);
                SelectObject(hDC, hPen);
                MoveToEx(hDC, prevX, prevY, NULL);
                LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
                prevX = LOWORD(lParam);
                prevY = HIWORD(lParam);

                point.x = prevX;
                point.y = prevY;

                pointElem.point = point;
                pointElem.color = curColor;
                pointList.push_back(pointElem);

                ReleaseDC(hWnd, hDC);
        }
                break;
case WM_PAINT:
        {
                HPEN hPen;
                hDC = BeginPaint(hWnd, &ps);

                list<POINT_ELEM>::iterator iter = pointList.begin();

                while (iter != pointList.end()) {
                        hPen = CreatePen(PS_SOLID, 10, iter->color);
                        SelectObject(hDC, hPen);
                        MoveToEx(hDC, iter->point.x, iter->point.y, NULL);
                        ++iter;
                        LineTo(hDC, iter->point.x, iter->point.y);
                        ++iter;
                        DeleteObject(hPen);
                }

                EndPaint(hWnd, &ps);

        }
        break;
```

```
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

void OnFileSave(const list<POINT_ELEM> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "wb")) == NULL) {
                MessageBox(NULL, "Cannot create file!..", "Error", MB_OK);
                return;
        }

        list<POINT_ELEM>::const_iterator iter = pointList.begin();

        for (; iter != pointList.end(); ++iter) {
                fwrite(&*iter, 1, sizeof(POINT_ELEM), f);
        }

        fclose(f);
}

void OnFileOpen(list<POINT_ELEM> &pointList)
{
        FILE *f;

        if ((f = fopen("paint.dat", "rb")) == NULL) {
                MessageBox(NULL, "Cannot open file!..", "Error", MB_OK);
                return;
        }

        pointList.clear();

        POINT_ELEM pointElem;
        for (;;) {
                fread(&pointElem.point, 1, sizeof(POINT_ELEM), f);
                if (feof(f))
                        break;
                pointList.push_back(pointElem);
        }
}
```

Menü elemanları, EnableMenuItem fonksiyonu ile etkisiz yada pasif duruma geçirilebilir.

```
BOOL EnableMenuItem(
 HMENU hMenu,           // handle to menu
 UINT uIDEnableItem,    // menu item to enable, disable, or gray
 UINT uEnable           // menu item flags
);
```

Fonksiyonun 1. parametresi menünün handle değeri, 2. parametresi menü elemanının ID değeri yada pozisyon numarasıdır. 3. parametre, MF_BYCOMMAND yada MF_BYPOSITION değerlerinin yanısıra MF_DISABLED, MF_ENABLED, MF_GRAYED içerebilir. MF_GRAYED, MF_DISABLED biçimini

kapsamakla birlikte aynı zamanda menü elemanını griye dönüştürür. Fonksiyonun geri dönüş değeri, menü elemanının önceki değeridir.

```
//...

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	int wmId, wmEvent;
	PAINTSTRUCT ps;
	HDC hdc;
	TCHAR szHello[MAX_LOADSTRING];
	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
	static HMENU hMenu;

	switch (message)
	{
		case WM_CREATE:
			hMenu = GetMenu(hWnd);
			EnableMenuItem(hMenu, ID_FILE_CLOSEX, MF_GRAYED);
			break;

		case WM_COMMAND:
			wmId    = LOWORD(wParam);
			wmEvent = HIWORD(wParam);
			// Parse the menu selections:
			switch (wmId)
			{
				case IDM_ABOUT:
				  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
						(DLGPROC)About);
				  break;
				case IDM_EXIT:
				  DestroyWindow(hWnd);
				  break;
				case ID_FILE_OPENX:
					EnableMenuItem(hMenu, ID_FILE_CLOSEX, MF_ENABLED);
					EnableMenuItem(hMenu, ID_FILE_OPENX, MF_GRAYED);
					break;
				case ID_FILE_CLOSEX:
					EnableMenuItem(hMenu, ID_FILE_OPENX, MF_ENABLED);
					EnableMenuItem(hMenu, ID_FILE_CLOSEX, MF_GRAYED);
					break;


				default:
				  return DefWindowProc(hWnd, message, wParam, lParam);
			}
			break;
		case WM_PAINT:
			hdc = BeginPaint(hWnd, &ps);
			// TODO: Add any drawing code here...
			RECT rt;
			GetClientRect(hWnd, &rt);
			DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
			EndPaint(hWnd, &ps);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
			return DefWindowProc(hWnd, message, wParam, lParam);
  }
```

```
  return 0;
}

//...
```

POPUP MENÜLERİNİN OLUŞTURULMASI

Ana menüden bağımsız olarak pencerenin herhangi bir yerinde fare tuşlarına basıldığında bir popup menü görüntülenmesi işlemine çok sık rastlanmaktadır. Popup menüler dinamik yolla yada kaynak yoluyla oluşturulabilir. Dinamik yöntemle oluşturma işlemi, sonraki bölümde ele alınacaktır.

Windows sistemlerinde tüm pulldown menü sisteminin bir handle değeri olduğu gibi, her popup menünün de ayrı bir handle değeri vardır. Kaynak işlemleriyle ancak pulldown menü sistemi oluşturulabilmektedir. Pulldown menü sisteminin bir popup menüsüne ilişkin handle, GetSubMenu fonksiyonu ile elde edilebilir.

```
HMENU GetSubMenu(
 HMENU hMenu,      // handle to menu
 int nPos          // menu item position
);
```

Fonksiyonun 1. parametresi pulldown menü sisteminin handle değeri, 2. parametresi pulldown menü sisteminin hangi popup penceresinin handle değerinin elde edileceğini belirten 0 orijinli bir indeks numarasıdır.

Kaynak kullanarak bir popup menü oluşturmak için kaynakta önce bir pulldown menü oluşturulur, sonra LoadMenu fonksiyonu ile pulldown menü kaynağı yüklenerek GetSubMenu fonksiyonu ile onun popup menü handle değeri elde edilir.

Popup menü, istenildiği zaman TrackPopupMenu fonksiyonu ile çıkartılabilir.

```
BOOL TrackPopupMenu(
  HMENU hMenu,          // handle to shortcut menu
  UINT uFlags,          // screen-position and mouse-button flags
  int x,                // horizontal position, in screen coordinates
  int y,                // vertical position, in screen coordinates
  int nReserved,        // reserved, must be zero
  HWND hWnd,            // handle to owner window
  CONST RECT *prcRect   // ignored
);
```

Fonksiyonun 1. parametresi popup menünün handle değeridir. 2. parametre menünün belirlenen noktaya göre hizalanma biçimini belirtir. Bu parametre, TPM_CENTERALIGN, RPM_LEFTALIGN ve TPM_RIGHTALIGN değerlerinden biri olabilir.

Fonksiyonun 3. ve 4. parametreler hizalama noktasını belirtir. Bu nokta, popup menüsünün sol üst köşesini belirmektedir. Bu değerler, çalışma alanı değil, masaüstü orijinlidir. Bu nedenle programcının fare mesajlarından aldığı koordinat bilgisini masaüstü orijinli biçime dönüştürmesi gerekir. Bu işlemler, ClientToScreen ve ScreenToClient fonksiyonlarıyla yapılmaktadır.

Fonksiyonun 5. parametresi kullanılmamaktadır. 0 biçiminde bırakılabilir. Fonksiyonun 6. parametresi popup menü penceresinin üst penceresi olacak pencerenin handle değeridir. Son parametre ise kullanılmamaktadır. NULL geçilebilir.

Kaynaktan hareketle popup menu çıkartılmak için şu işlemler yapılmalıdır:

1. Kaynakta bir pulldown menü oluşturulur. Pulldown menüye bir popup penceresi eklenir.
2. Pulldown menü bütünsel olarak LoadMenu fonksiyonu ile yüklenir ve pulldoen menünün handle değeri elde edilir.
3. GetSubMenu fonksiyonuyla pulldown menünün popup penceresinin handle değeri alınır,
4. TrackPopupMenu fonksiyonuyla popup menü açılır.

```
// apiwizard.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR     lpCmdLine,
                     int  nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_APIWIZARD, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_APIWIZARD);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}
```

```
//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style          = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc    = (WNDPROC)WndProc;
        wcex.cbClsExtra     = 0;
        wcex.cbWndExtra     = 0;
        wcex.hInstance      = hInstance;
        wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_APIWIZARD);
        wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName   = (LPCSTR)IDC_APIWIZARD;
        wcex.lpszClassName  = szWindowClass;
        wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);
```

```
  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND      - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY      - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
        static HMENU hContextMenu, hMenu;

        switch (message)
        {
                case WM_CREATE:
                        hMenu = GetMenu(hWnd);
                        EnableMenuItem(hMenu, ID_FILE_CLOSEX, MF_GRAYED);
                        hContextMenu = GetSubMenu(LoadMenu(hInst,
                                        MAKEINTRESOURCE(IDR_CONTEXT_MENUS)), 0);
                        break;
                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                                (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                case ID_FILE_OPENX:
                                        EnableMenuItem(hMenu, ID_FILE_CLOSEX, MF_ENABLED);
                                        EnableMenuItem(hMenu, ID_FILE_OPENX, MF_GRAYED);
                                        break;
                                case ID_FILE_CLOSEX:
                                        EnableMenuItem(hMenu, ID_FILE_OPENX, MF_ENABLED);
                                        EnableMenuItem(hMenu, ID_FILE_CLOSEX, MF_GRAYED);
                                        break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);
                        // TODO: Add any drawing code here...
                        RECT rt;
                        GetClientRect(hWnd, &rt);
                        DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                        EndPaint(hWnd, &ps);
                        break;
```

```
                case WM_RBUTTONDOWN:
                        {
                                POINT pt;

                                pt.x = LOWORD(lParam);
                                pt.y = HIWORD(lParam);
                                ClientToScreen(hWnd, &pt);

                        TrackPopupMenu(hContextMenu, TPM_LEFTALIGN, pt.x, pt.y, 0, hWnd, NULL);
                        }
                        break;


                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

//...
```

Ayrıca Windows'ta farenin sağ tuşuna basıldığında WM_RBUTTONDOWN mesajının yanısıra WM_CONTEXTMENU isimli başka bir mesaj daha oluşturulmaktadır. Bu mesaja geçirilen parametreler default masaüstü orijinli olduğu için bir dönüştürme yapılmasına gerek yoktur.

```
case WM_CONTEXTMENU:
        TrackPopupMenu(hContextMenu, TPM_LEFTALIGN, LOWORD(lParam),
                        HIWORD(lParam),0, hWnd, NULL);
        break;
```

Popup menülerden eleman seçildiğinde tamamen seçim işlemi ana menüden yapılmış gibi yine menü elemanının ID değeri ile WM_COMMAND mesajı gönderilir.

```
case ID_FIRSTPOPUP_ELMA:
        MessageBox(hWnd, "Elma", "Message", MB_OK);
        break;
case ID_FIRSTPOPUP_ARMUT:
        MessageBox(hWnd, "Armut", "Message", MB_OK);
        break;
case ID_FIRSTPOPUP_AVOKADO:
        MessageBox(hWnd, "Avokado", "Message", MB_OK);
        break;
```

```
// multi_context_menu.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING      100

#define NMOUSE_REGIONS      5

// Global Variables:
HINSTANCE hInst;                                // current instance
```

```
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];    // The title bar text


typedef struct tagMOUSE_REGION {
        int left, top, right, bottom;
        HMENU hMenu;
} MOUSE_REGION;

MOUSE_REGION   g_regions[] = {
        {100, 100, 200, 200}, {0, 0, 100, 100}, {300, 300, 400, 400},
        {200, 200, 400, 300}, {400, 100, 600, 150},
};

// Foward declarations of functions included in this code module:
ATOM            MyRegisterClass(HINSTANCE hInstance);
BOOL            InitInstance(HINSTANCE, int);
LRESULT CALLBACK   WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK   About(HWND, UINT, WPARAM, LPARAM);
void OnRButtonDown(HWND hWnd, int x, int y);

int APIENTRY WinMain(HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPSTR     lpCmdLine,
                int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_MULTI_CONTEXT_MENU, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_MULTI_CONTEXT_MENU);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
```

```
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style          = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc    = (WNDPROC)WndProc;
        wcex.cbClsExtra     = 0;
        wcex.cbWndExtra     = 0;
        wcex.hInstance      = hInstance;
        wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_MULTI_CONTEXT_MENU);
        wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName   = (LPCSTR)IDC_MULTI_CONTEXT_MENU;
        wcex.lpszClassName  = szWindowClass;
        wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
```

```
//
//  WM_COMMAND      - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY      - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
        int i;
        HMENU hPullDownMenu;

        switch (message)
        {
                case WM_CREATE:
                        hPullDownMenu = LoadMenu(hInst,
                                                MAKEINTRESOURCE(IDR_MENU_REGIONS));
                        for (i = 0; i < NMOUSE_REGIONS; ++i) {
                                g_regions[i].hMenu = GetSubMenu(hPullDownMenu, i);
                        }
                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                        (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_RBUTTONDOWN:
                        OnRButtonDown(hWnd, LOWORD(lParam), HIWORD(lParam));
                        break;

                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);

                        for (i = 0; i < NMOUSE_REGIONS; ++i)
                                Rectangle(hdc, g_regions[i].left, g_regions[i].top, g_regions[i].right,
                                        g_regions[i].bottom);

                        EndPaint(hWnd, &ps);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
```

```
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                                return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
   return FALSE;
}

void OnRButtonDown(HWND hWnd, int x, int y)
{
        RECT rect;
        POINT pt;
        int i;


        for (i = 0; i < NMOUSE_REGIONS; ++i) {
                rect.left = g_regions[i].left;
                rect.right = g_regions[i].right;
                rect.top = g_regions[i].top;
                rect.bottom = g_regions[i].bottom;
                pt.x = x;
                pt.y = y;

                if (PtInRect(&rect, pt)) {
                        ClientToScreen(hWnd, &pt);
                        TrackPopupMenu(g_regions[i].hMenu, TPM_LEFTALIGN, pt.x, pt.y, 0, hWnd,
                                        NULL);
                        return;
                }

        }
}
```

Menülerin Programın Çalışma Zamanında Oluşturulması

Menüler, tamamen programın çalışma zamanında dinamik olarak oluşturulabilir. bir Pull Down Menu boş olarak CreateMenu API fonksiyonuyla yaratılabilir. Boş bir Popup Menu ise CreatePopupMenu fonksiyonuyla yaratılmaktadır. Pull Down Menu çubuğu ya da Popup Menu yaratıldıktan sonra InsertMenu, AppendMenu gibi fonksiyonlarla menüye eleman eklenebilir. Menü elemanı yazıdan oluşan basit bir eleman olabileceği gibi bir Popup Menu olabilir.

```
HMENU CreatePopupMenu(VOID);
```

```
HMENU CreateMenu(VOID);
```

AppendMenu fonksiyonu şöyledir:

```
BOOL AppendMenu(
```

```
 HMENU hMenu,            // handle to menu to be changed
 UINT uFlags,            // menu-item flags
 UINT uIDNewItem,        // menu-item identifier or handle to dropdown menu or submenu
 LPCTSTR lpNewItem       // menu-item content
);
```

Fonksiyonun birinci parametresi menünün handle değeridir. İkinci parametresi MF_XXX biçiminde çeşili sembolik sabitlerden oluşturulmaktadır. Örneğin, MF_STRING, menü elemanının düz bir yazı olacağını belirtmektedir. MF_POPUP, menü elemanının bir Popup Menu olacağını belirtir. MF_BITMAP ise menü elemanının bir icon görüntüsünde olacağını belirtir. Fonksiyonun üçüncü parametresi, ekelenen menü elemanının ID'sini belirtir. Eğer menü elemanı Popup ise buraya Popup Menu' nün handle değeri girilmelidir. Nihayet son parametre menü elemanının yazısını belirtir. Bu fonksiyon menü elemanının menünün sonuna eklemektedir.

```
// multi_context_menu.cpp : Defines the entry point for the application.
//...

typedef struct tagMOUSE_REGION {
        int left, top, right, bottom;
        HMENU hMenu;
} MOUSE_REGION;

MOUSE_REGION   g_regions[] = {
        {100, 100, 200, 200}, {0, 0, 100, 100}, {300, 300, 400, 400},
        {200, 200, 400, 300}, {400, 100, 600, 150},
};

// Foward declarations of functions included in this code module:
ATOM  MyRegisterClass(HINSTANCE hInstance);
BOOL   InitInstance(HINSTANCE, int);
LRESULT CALLBACK  WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK  About(HWND, UINT, WPARAM, LPARAM);
void OnRButtonDown(HWND hWnd, int x, int y);

//...


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
        int i;
        HMENU hPullDownMenu;

        switch (message)
        {
                case WM_CREATE:
                        hPullDownMenu = LoadMenu(hInst,
                                                MAKEINTRESOURCE(IDR_MENU_REGIONS));
                        for (i = 0; i < NMOUSE_REGIONS; ++i) {
                                g_regions[i].hMenu = GetSubMenu(hPullDownMenu, i);
                        }
                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
```

```
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                                (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_RBUTTONDOWN:
                        OnRButtonDown(hWnd, LOWORD(lParam), HIWORD(lParam));
                        break;
                case WM_LBUTTONDOWN:
                        {
                                HMENU hMenu = GetMenu(hWnd);
                                HMENU hFilePopupMenu = GetSubMenu(hMenu, 0);
                                HMENU hNewPopup = CreatePopupMenu();

                                AppendMenu(hNewPopup, MF_STRING, 500, "&Test");
                                AppendMenu(hNewPopup, MF_STRING, 600, "&Rest");

                                AppendMenu(hFilePopupMenu, MF_POPUP, (UINT) hNewPopup, "Test");

                        }
                        break;

                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);

                        for (i = 0; i < NMOUSE_REGIONS; ++i)
                                Rectangle(hdc, g_regions[i].left, g_regions[i].top, g_regions[i].right,
                                        g_regions[i].bottom);

                        EndPaint(hWnd, &ps);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
 //...
void OnRButtonDown(HWND hWnd, int x, int y)
{
        RECT rect;
        POINT pt;
        int i;

        for (i = 0; i < NMOUSE_REGIONS; ++i) {
                rect.left = g_regions[i].left;
                rect.right = g_regions[i].right;
                rect.top = g_regions[i].top;
                rect.bottom = g_regions[i].bottom;
                pt.x = x;
                pt.y = y;
```

```
                if (PtInRect(&rect, pt)) {
                        ClientToScreen(hWnd, &pt);
                        TrackPopupMenu(g_regions[i].hMenu, TPM_LEFTALIGN, pt.x, pt.y, 0, hWnd,
                                        NULL);
                        return;
                }
        }
}
```

InsertMenu fonksiyonunun prototipi de şöyledir:

```
BOOL InsertMenu(
 HMENU hMenu,           // handle to menu
 UINT uPosition,        // menu item that new menu item precedes
 UINT uFlags,           // menu item flags
 UINT uIDNewItem,       // menu item identifier or handle to dropdown menu or submenu
 LPCTSTR lpNewItem      // menu item content
);
```

Bu fonksiyonun AppendMenu fonksiyonundan tek farkı bu fonksiyonun sona ekleme yerine herhangi bir pozisyona ekleme yapabilmesidir. Diğer parametreler aynı anlamdadır.

Diğer menü fonksiyonları MSDN dokümanlarından takip edilebilir.

ÇİZİM İŞLEMLERİNİN AYRINTILARI

API programlamada kullanılan klasik çizim kütüphanesine GDI kütüphanesi denilmektedir. GDI kütüphanesinin fonksiyonları gdi32dll modülü içerisindedir. Bu modül, Windows'un bir parçasıdır. Yani diğer API fonksiyonlarında olduğu gibi GDI fonksiyonlarını da kullanabilmek için özel bir şey yapmaya gerek yoktur. Son yıllarda Microsoft tarafından farklı bir tasarımla yeni bir çizim kütüphanesi daha oluşturulmuştur. Bu çizim kütüphanesine GDI+ denilmektedir. GDI+, temelde .NET ortamı için düşünülmüş bir kütüphanedir. Bu kütüphanenin bulunduğu dll, XP haricinde diğer Windows sistemlerinin doğal bir parçası değildir. .NET frameWork kurulduğunda sisteme çekilmektedir. GDI+ genel olarak GDI sisteminden daha gelişmiş bir kütüphanedir.

Region İŞLEMLERİ

Dikdörtgensel olmak zorunda olmayan alanlar topluluğuna Region denilmektedir. Region, HRGN türüyle temsil edilir.

Region oluşturmak için bir grup CreateXXXRgn fonksiyonu vardır. Örneğin CreateEllipticRgn elips biçiminde bir region oluşturmakta, CreateRectRgn, dikdörtgensel bir region oluşturmaktadır.

```
HRGN CreateEllipticRgn(
 int nLeftRect,         // x-coord of the upper-left corner of the bounding rectangle
 int nTopRect,          // y-coord of the upper-left corner of the bounding rectangle
 int nRightRect,        // x-coord of the lower-right corner of the bounding rectangle
 int nBottomRect        // y-coord of the lower-right corner of the bounding rectangle
);
```

```
HRGN CreateRectRgn(
 int nLeftRect,         // x-coordinate of region's upper-left corner
 int nTopRect,          // y-coordinate of region's upper-left corner
```

```
 int nRightRect,        // x-coordinate of region's lower-right corner
 int nBottomRect        // y-coordinate of region's lower-right corner
);
```

Region, pek çok amaçla kullanılabilmektedir. Örneğin bir noktanın bir region içerisinde olup olmadığı, PtInRegion fonksiyonu ile tespit edilebilir.

```
BOOL PtInRegion(
  HRGN hrgn,        // handle to region
  int X,            // x-coordinate of point
  int Y             // y-coordinate of point
);
```

Fonksiyonun geri dönüş değeri 0 dışı ise nokta region'ın içerisinde; 0 ise içerisindedir.
// hittesting.cpp

```
// hittesting.cpp : Defines the entry point for the application.
//
//.....

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	int wmId, wmEvent;
	PAINTSTRUCT ps;
	HDC hdc;
	static HRGN hRegion;
	TCHAR szHello[MAX_LOADSTRING];
	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch (message)
	{
		case WM_CREATE:
			hRegion = CreateEllipticRgn(100, 100, 200, 200);
			break;

		case WM_COMMAND:
			wmId    = LOWORD(wParam);
			wmEvent = HIWORD(wParam);
			// Parse the menu selections:
			switch (wmId)
			{
				case IDM_ABOUT:
				  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
						(DLGPROC)About);
				  break;
				case IDM_EXIT:
				  DestroyWindow(hWnd);
				  break;
				default:
				  return DefWindowProc(hWnd, message, wParam, lParam);
			}
			break;
		case WM_PAINT:
			hdc = BeginPaint(hWnd, &ps);

			Ellipse(hdc, 100, 100, 200, 200);

			EndPaint(hWnd, &ps);
			break;
```

```
		case WM_LBUTTONDOWN:
			if (PtInRegion(hRegion, LOWORD(lParam), HIWORD(lParam)))
				MessageBox(hWnd, "Ok", "Message", MB_OK);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
			return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}
```

```
// hittesting.cpp : Defines the entry point for the application.
//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	int wmId, wmEvent;
	PAINTSTRUCT ps;
	HDC hdc;
	static HRGN hRegion;
	TCHAR szHello[MAX_LOADSTRING];
	POINT points[] = {{100, 100}, {100, 150}, {150, 300}, {170, 230}};


	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch (message)
	{
		case WM_CREATE:
			hRegion = CreatePolygonRgn(points, 4, WINDING);
			break;

		case WM_COMMAND:
			wmId    = LOWORD(wParam);
			wmEvent = HIWORD(wParam);
			// Parse the menu selections:
			switch (wmId)
			{
				case IDM_ABOUT:
				  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
						(DLGPROC)About);
				  break;
				case IDM_EXIT:
				  DestroyWindow(hWnd);
				  break;
				default:
				  return DefWindowProc(hWnd, message, wParam, lParam);
			}
			break;
		case WM_PAINT:
			hdc = BeginPaint(hWnd, &ps);

			Polygon(hdc, points, 4);

			EndPaint(hWnd, &ps);
			break;
		case WM_LBUTTONDOWN:
			if (PtInRegion(hRegion, LOWORD(lParam), HIWORD(lParam)))
				MessageBox(hWnd, "Ok", "Message", MB_OK);
```

```
                break;
            case WM_DESTROY:
                PostQuitMessage(0);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// ....
```

FillRgn fonksiyonu bir region'ın işçini boyamakta kullanılır. Prototipi aşağıdaki gibidir:

```
BOOL FillRgn(
 HDC hdc,           // handle to device context
 HRGN hrgn,         // handle to region to be filled
 HBRUSH hbr         // handle to brush used to fill the region
);
```

```
// hittesting.cpp

//...

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    static HRGN hRegion;
    TCHAR szHello[MAX_LOADSTRING];
    POINT points[] = {{100, 100}, {100, 150}, {150, 300}, {170, 230}};
    static HBRUSH hRedBrush;


    LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

    switch (message)
    {
        case WM_CREATE:
            hRegion = CreatePolygonRgn(points, 4, WINDING);
            hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
            break;

        case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
                case IDM_ABOUT:
                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                            (DLGPROC)About);
                  break;
                case IDM_EXIT:
                  DestroyWindow(hWnd);
                  break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
            }
```

```
                    break;
                case WM_PAINT:
                    hdc = BeginPaint(hWnd, &ps);

                    Polygon(hdc, points, 4);
                    FillRgn(hdc, hRegion, hRedBrush);

                    EndPaint(hWnd, &ps);
                    break;
                case WM_LBUTTONDOWN:

                    if (PtInRegion(hRegion, LOWORD(lParam), HIWORD(lParam)))
                        MessageBox(hWnd, "Ok", "Message", MB_OK);
                    break;
                case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
//...
```

Birden fazla region, CombineRgn fonksiyonu ile birleştirilebilir.

```
int CombineRgn(
 HRGN hrgnDest,     // handle to destination region
 HRGN hrgnSrc1,     // handle to source region
 HRGN hrgnSrc2,     // handle to source region
 int fnCombineMode // region combining mode
);
```

Bu fonksiyon, iki region'ı birleştirerek başka bir region elde eder. Fonksiyonun 2. ve 3. parametreleri birleştirilecek region'lar; 1. parametresi hedef region'dır. Fonksiyonun son parametresi, birleştirmenin nasıl yapılacağını belirtmektedir. Fonksiyonun son parametresi RGN_AND ise, iki region'ın kesişim kümesi hedef region olur. RGN_COPY ise, ikinci parametresiyle belirtilen region, 1. parametreyle belirtilen region'a kopyalanır. RGN_DIFF ise, 2. parametre ile belirtilen region'dan 3. parametreyle belirtilen region'n çıkartılması ile elde edilen küme hedef region'a atanır. RGN_OR, iki region'ın birleşim kümesinden region elde eder. RGN_XOR, iki kümenin birleşiminden kesişimi olan kümenin çıkartılmasıyla elde edilen alandan region oluşturur.

OffsetRgn Fonksiyonu

Bu fonksiyon, bir region'ı x ve y değeri kadar ötelemek için kullanılır.

```
int OffsetRgn(
 HRGN hrgn,         // handle to region
 int nXOffset,      // offset along x-axis
 int nYOffset       // offset along y-axis
);
```

PENCERENİN AKTİF ÇİZİM ALANI

Pencereler, dikdörtgensel alanlardır. Normal olarak Windows, pencerenin tümünü aktif çizim alanı olarak kullanır. Fakat istenirse pencerenin aktif çizim alanı dikdörtgensel alanı olmaktan çıkartılıp belirli bir region yapılabilir. Bu durumda Windows pencere görüntüsünü çizeceği zaman tüm dikdörtgensel alanı değil o dikdörtgensel alan içindeki region'ı çizer. Yuvarlak pencereler böyle oluşturulmaktadır. Pencerenin aktif çizim alanını değitirmek için SetWindowRgn fonksiyonu kullanılmaktadır.

```
int SetWindowRgn(
  HWND hWnd,     // handle to window whose window region is to be set
  HRGN hRgn,     // handle to region
  BOOL bRedraw   // window redraw flag
);
```

Fonksiyonun 1. parametresi pencerenin handle değeri, 2. parametresi region, 3. parametresi ise görüntünün o anda tazelenip tazelenmeyeceğini belirtmektedir; TRUE geçilebilir. Söz konusu region'daki noktalar, çalışma alanının sol üst köşesi orijinli değil pencerenin sol üst köşesi orijinlidir.

```
case WM_CREATE:
        {
                HRGN hRegion;

                hRegion = CreateEllipticRgn(100, 100, 200, 200);
                SetWindowRgn(hWnd, hRegion, TRUE);


        }
        break;
```

```
switch (message)
        {
        case WM_CREATE:
                        {
                                HRGN hRegion;

                                hRegion = CreateEllipticRgn(100, 100, 200, 200);
                                SetWindowRgn(hWnd, hRegion, TRUE);


                        }
                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                          (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
```

```
                break;
        case WM_PAINT:
                hdc = BeginPaint(hWnd, &ps);
                // TODO: Add any drawing code here...
                RECT rt;
                GetClientRect(hWnd, &rt);
                DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                EndPaint(hWnd, &ps);
                break;
        case WM_LBUTTONDOWN:
                {
                        char s[100];

                        sprintf(s, "%d - %d", LOWORD(lParam), HIWORD(lParam));
                        MessageBox(hWnd, s, "Message", MB_OK);

                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
  }
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static prevX, prevY;
        static int X, Y;

        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        {
                                HRGN hRegion;
                                RECT rect;

                                hRegion = CreateEllipticRgn(100, 100, 200, 200);
                                SetWindowRgn(hWnd, hRegion, TRUE);
                                GetWindowRect(hWnd, &rect);
                                X = rect.left;
                                Y = rect.top;


                        }
                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
```

```
                                        (DLGPROC)About);
                                    break;
                                case IDM_EXIT:
                                    DestroyWindow(hWnd);
                                    break;
                                default:
                                    return DefWindowProc(hWnd, message, wParam, lParam);
                            }
                            break;
                        case WM_PAINT:
                            hdc = BeginPaint(hWnd, &ps);
                            // TODO: Add any drawing code here...
                            RECT rt;
                            GetClientRect(hWnd, &rt);
                            DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                            EndPaint(hWnd, &ps);
                            break;
                        case WM_LBUTTONDOWN:
                            {
                                prevX = LOWORD(lParam);
                                prevY = HIWORD(lParam);
                            }
                            break;
                        case WM_MOUSEMOVE:
                            if (wParam & MK_LBUTTON) {
                                X += LOWORD(lParam) - prevX;
                                Y += HIWORD(lParam) - prevY;
                                MoveWindow(hWnd, X, Y, 300, 300,TRUE);
                                prevX = LOWORD(lParam);
                                prevY = HIWORD(lParam);
                            }
                            break;
                        case WM_DESTROY:
                            PostQuitMessage(0);
                            break;
                        default:
                            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

---

içine yazı yazalım:

```
// CircleWindow.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"
#include <stdio.h>

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar
text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
LRESULT CALLBACK  About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_CIRCLEWINDOW, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_CIRCLEWINDOW);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra         = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_CIRCLEWINDOW);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
```

```
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_CIRCLEWINDOW;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    0, 0,300, 300, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND      - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY      - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static prevX, prevY;
        static int X, Y;

        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        {
```

```
                        HRGN hRegion;
                        RECT rect;

                        hRegion = CreateEllipticRgn(100, 100, 200, 200);
                        SetWindowRgn(hWnd, hRegion, TRUE);

                }
                break;

        case WM_COMMAND:
                wmId    = LOWORD(wParam);
                wmEvent = HIWORD(wParam);
                // Parse the menu selections:
                switch (wmId)
                {
                        case IDM_ABOUT:
                          DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                  (DLGPROC)About);
                          break;
                        case IDM_EXIT:
                          DestroyWindow(hWnd);
                          break;
                        default:
                          return DefWindowProc(hWnd, message, wParam, lParam);
                }
                break;
        case WM_PAINT:
                {
                        hdc = BeginPaint(hWnd, &ps);

                RECT rt = {100, 100, 200, 200};

                DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                EndPaint(hWnd, &ps);
                }
                        break;
        case WM_LBUTTONDOWN:
                {
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);
                }
                break;
        case WM_MOUSEMOVE:
                if (wParam & MK_LBUTTON) {
                        X += LOWORD(lParam) - prevX;
                        Y += HIWORD(lParam) - prevY;
                        MoveWindow(hWnd, X, Y, 300, 300,TRUE);
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);
                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
	switch (message)
	{
		case WM_INITDIALOG:
				return TRUE;

		case WM_COMMAND:
			if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
			{
				EndDialog(hDlg, LOWORD(wParam));
				return TRUE;
			}
			break;
	}
   return FALSE;
}
```

---

içine buton koyalım:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	int wmId, wmEvent;
	PAINTSTRUCT ps;
	HDC hdc;
	TCHAR szHello[MAX_LOADSTRING];
	static prevX, prevY;
	static int X, Y;
	static HWND hButton;


	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch (message)
	{
		case WM_CREATE:
			{
				HRGN hRegion;
				RECT rect;

				hRegion = CreateEllipticRgn(100, 100, 200, 200);
				SetWindowRgn(hWnd, hRegion, TRUE);
				hButton = CreateWindow("button", "Ok",
						BS_PUSHBUTTON|WS_CHILD|WS_VISIBLE,
						100, 100, 50,30, hWnd, (HMENU) 100, hInst, NULL);

			}
			break;

		case WM_COMMAND:
			wmId    = LOWORD(wParam);
			wmEvent = HIWORD(wParam);
			// Parse the menu selections:
			switch (wmId)
			{
				case IDM_ABOUT:
				  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
						(DLGPROC)About);
				  break;
				case IDM_EXIT:
				  DestroyWindow(hWnd);
				  break;
				default:
```

```
                        return DefWindowProc(hWnd, message, wParam, lParam);
                }
                break;
        case WM_PAINT:
                {
                        hdc = BeginPaint(hWnd, &ps);

                RECT rt = {100, 100, 200, 200};

                DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                EndPaint(hWnd, &ps);
                }
                        break;
        case WM_LBUTTONDOWN:
                {
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);
                }
                break;
        case WM_MOUSEMOVE:
                if (wParam & MK_LBUTTON) {
                        X += LOWORD(lParam) - prevX;
                        Y += HIWORD(lParam) - prevY;
                        MoveWindow(hWnd, X, Y, 300, 300,TRUE);
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);
                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

---

içindeki butona klik yapınca pencereyi kapatsın.....:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static prevX, prevY;
        static int X, Y;
        static HWND hButton;


        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        {
                                HRGN hRegion;
                                RECT rect;

                                hRegion = CreateEllipticRgn(100, 100, 200, 200);
                                SetWindowRgn(hWnd, hRegion, TRUE);
```

```
                        hButton = CreateWindow("button", "Ok",
                                WS_PUSHBUTTON|WS_CHILD|WS_VISIBLE,
                                120, 100, 50,30, hWnd, (HMENU) 100, hInst, NULL);

                }
                break;

        case WM_COMMAND:
                wmId    = LOWORD(wParam);
                wmEvent = HIWORD(wParam);
                // Parse the menu selections:
                switch (wmId)
                {
                case ID_BUTTON:
                        if (HIWORD(wParam) == BN_CLICKED) {
                                DestroyWindow(hWnd);
                        }
                        break;
                        case IDM_ABOUT:
                          DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                (DLGPROC)About);
                          break;
                        case IDM_EXIT:
                          DestroyWindow(hWnd);
                          break;
                        default:
                          return DefWindowProc(hWnd, message, wParam, lParam);
                }
                break;
        case WM_PAINT:
                {
                        hdc = BeginPaint(hWnd, &ps);

                        RECT rt = {100, 50, 200, 200};

                        DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                        EndPaint(hWnd, &ps);
                }
                        break;
        case WM_LBUTTONDOWN:
                {
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);
                }
                break;
        case WM_MOUSEMOVE:
                if (wParam & MK_LBUTTON) {
                        X += LOWORD(lParam) - prevX;
                        Y += HIWORD(lParam) - prevY;
                        MoveWindow(hWnd, X, Y, 300, 300,TRUE);
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);
                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

içini boyayıp yazıyı küçültelim. bu arada sınıfı değiştirdik:

```
// CircleWindow.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"
#include <stdio.h>

#define MAX_LOADSTRING 100
#define ID_BUTTON                   100


// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];    // The title bar text

// Foward declarations of functions included in this code module:
ATOM                                MyRegisterClass(HINSTANCE hInstance);
BOOL                                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_CIRCLEWINDOW, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_CIRCLEWINDOW);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}
```

```
//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style          = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc    = (WNDPROC)WndProc;
        wcex.cbClsExtra     = 0;
        wcex.cbWndExtra     = 0;
        wcex.hInstance      = hInstance;
        wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_CIRCLEWINDOW);
        wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground  = CreateSolidBrush(RGB(255, 0, 0));
        wcex.lpszMenuName   = (LPCSTR)IDC_CIRCLEWINDOW;
        wcex.lpszClassName  = szWindowClass;
        wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
                      0, 0,300, 300, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
```

```
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND      - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY      - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static prevX, prevY;
        static int X, Y;
        static HWND hButton;

        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        {
                                HRGN hRegion;
                                RECT rect;

                                hRegion = CreateEllipticRgn(100, 100, 200, 200);
                                SetWindowRgn(hWnd, hRegion, TRUE);
                                hButton = CreateWindow("button", "Ok",
                                        BS_PUSHBUTTON|WS_CHILD|WS_VISIBLE,
                                        120, 100, 50,30, hWnd, (HMENU) 100, hInst, NULL);

                        }
                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                        case ID_BUTTON:
                                if (HIWORD(wParam) == BN_CLICKED) {
                                        DestroyWindow(hWnd);
                                }
                                break;
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                                (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_PAINT:
```

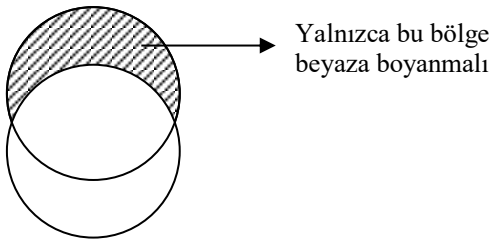
```
                {
                        hdc = BeginPaint(hWnd, &ps);

                        RECT rt = {100, 80, 200, 200};

                        DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                        EndPaint(hWnd, &ps);
                }
                        break;
        case WM_LBUTTONDOWN:
                {
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);
                }
                break;
        case WM_MOUSEMOVE:
                if (wParam & MK_LBUTTON) {
                        X += LOWORD(lParam) - prevX;
                        Y += HIWORD(lParam) - prevY;
                        MoveWindow(hWnd, X, Y, 300, 300,TRUE);
                        prevX = LOWORD(lParam);
                        prevY = HIWORD(lParam);
                }
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
   return FALSE;
}
```

DC'NİN CLIP (kesim) ALANI

Windows'un GDI çizim fonksiyonları aslında yalnızca DC'nin kesim alanına çizim yapabilmektedir. DC'nin kesin alanı (clip region), DC handle alanı içerisine saklanan bir bilgidir. DC, BeginPaint fonksiyonu ile elde edilmişse default kesim alanı güncelleme alanı (update region), GetDc fonksiyonu ile elde edilmişse default kesim alanı tüm çalışma alanıdır (client area). Fakat DC alındıktan sonra belirli bir region DC'nin kesim alanı yapılabilir. Böylece biz artık pencerenin neresine çizmek istersek isteyelim bu işlemden yalnızca kesim alanındaki bölge etkilenir.

GDI çizim fonksiyonları, önce parametreleriyle belirtilen koordinatlara dayalı olarak boyanacak pixelleri belirler. Kesim alanı dışındaki pixelleri bu kümeden atar; yalnızca kesim alanı içerisine düşen pixelleri gerçek anlamda boyar.

DC'nin kesim alanı özellikle zemin ve şekil renginin farklı olduğu durumlarda şekil kaydırmaları için kullanılmaktadır. Örneğin kırmızı bir dairenin fare mesajları yardımıyla beyaz bir zemin üzerinde taşınması işleminde eski daire yeriyle yeni sdaire yeri ile kesişimi olan bölgenin gereksiz bir biçimde zemin rengiyle boyamaması gerekir.



Bir DC'nin region'ı SelectClipRgn veya ExtSelectClipRgn fonksiyonlarıyla değiştirilebilir. ExtSelectClipRgn fonksiyonu, SelectClipRgn fonksiyonunun daha gelişmiş bir biçimidir.

```
int ExtSelectClipRgn(
 HDC hdc,      // handle to device context
 HRGN hrgn, // handle to region
 int fnMode    // region-selection mode
);
```

Fonksiyonun 1. parametresi kesim alanı değiştirilecek DC'nin handle değeridir. 2. parametre programcının oluşturduğu region'ı belirtir. 3. parametre, bu region ile DC'deki region arasında nasıl bir işlem yapılacağını belirlemektedir. Son parametre, RGN_AND RGN_COPY, RGN_DIFF, RGN_OR ve RGN_XOR değerlerinden biri olabilir. Eğer bu değer RGN_COPY ise parametreyle belirtilen region DC'nin kesim alanı yapılır; yani bu durumda fonksiyon tamamen SelectClipRgn gibi davranır.

Kaydırınca titreyen versiyon:

```
// ClipRegion.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];    // The title bar text
RECT g_circle = {100, 100, 200, 200};


// Foward declarations of functions included in this code module:
ATOM            MyRegisterClass(HINSTANCE hInstance);
BOOL            InitInstance(HINSTANCE, int);
LRESULT CALLBACK  WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK  About(HWND, UINT, WPARAM, LPARAM);
```

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_CLIPREGION, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_CLIPREGION);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style              = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra         = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon              = LoadIcon(hInstance, (LPCTSTR)IDI_CLIPREGION);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_CLIPREGION;
```

```
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND      - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY      - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static HBRUSH hRedBrush, hWhiteBrush;
        static int prevX, prevY;
        static HPEN redPen, whitePen;


        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
```

```
            hWhiteBrush = (HBRUSH) GetStockObject(WHITE_BRUSH);
            redPen = CreatePen(0, PS_SOLID, RGB(255, 0, 0));
            whitePen = (HPEN) GetStockObject(WHITE_PEN);

            break;

    case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
                    case IDM_ABOUT:
                      DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                    (DLGPROC)About);
                      break;
                    case IDM_EXIT:
                      DestroyWindow(hWnd);
                      break;
                    default:
                      return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
    case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);


            SelectObject(hdc, hRedBrush);
            SelectObject(hdc, redPen);

            Ellipse(hdc, g_circle.left, g_circle.top, g_circle.right, g_circle.bottom);

            EndPaint(hWnd, &ps);
            break;

    case WM_LBUTTONDOWN:
            prevX = LOWORD(lParam);
            prevY = HIWORD(lParam);
            break;

    case WM_MOUSEMOVE:
            {

                    if (wParam & MK_LBUTTON) {
                            HDC hDC = GetDC(hWnd);

                            SelectObject(hDC, whitePen);
                            Ellipse(hDC, g_circle.left, g_circle.top, g_circle.right,
                                    g_circle.bottom);
                            OffsetRect(&g_circle, LOWORD(lParam) –
                                    prevX,HIWORD(lParam) - prevY);

                            SelectObject(hDC, hRedBrush);
                            SelectObject(hDC, redPen);

                            Ellipse(hDC, g_circle.left, g_circle.top, g_circle.right,
                                    g_circle.bottom);

                            prevX = LOWORD(lParam);
                            prevY = HIWORD(lParam);
                    }

            }
```

```
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                                return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
    return FALSE;
}
```

Kaydırınca titremeyen versiyon (region'la yapılan):

```
// ClipRegion.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];    // The title bar text
RECT g_circle = {100, 100, 200, 200};


// Foward declarations of functions included in this code module:
ATOM            MyRegisterClass(HINSTANCE hInstance);
BOOL            InitInstance(HINSTANCE, int);
LRESULT CALLBACK  WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK  About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPSTR     lpCmdLine,
                int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;
```

```
	// Initialize global strings
	LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
	LoadString(hInstance, IDC_CLIPREGION, szWindowClass, MAX_LOADSTRING);
	MyRegisterClass(hInstance);

	// Perform application initialization:
	if (!InitInstance (hInstance, nCmdShow))
	{
		return FALSE;
	}

	hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_CLIPREGION);

	// Main message loop:
	while (GetMessage(&msg, NULL, 0, 0))
	{
		if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProc;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_CLIPREGION);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= (LPCSTR)IDC_CLIPREGION;
	wcex.lpszClassName	= szWindowClass;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

	return RegisterClassEx(&wcex);
}

//
```

```
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND       - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY         - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static HBRUSH hRedBrush, hWhiteBrush;
        static int prevX, prevY;
        static HPEN redPen, whitePen;
        static HRGN hRegion1, hRegion2;


        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        hRedBrush = CreateSolidBrush(RGB(255, 0, 0));
                        hWhiteBrush = (HBRUSH) GetStockObject(WHITE_BRUSH);
                        redPen = CreatePen(0, PS_SOLID, RGB(255, 0, 0));
                        whitePen = (HPEN) GetStockObject(WHITE_PEN);

                        break;
```

```
case WM_COMMAND:
        wmId    = LOWORD(wParam);
        wmEvent = HIWORD(wParam);
        // Parse the menu selections:
        switch (wmId)
        {
                case IDM_ABOUT:
                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                        (DLGPROC)About);
                  break;
                case IDM_EXIT:
                  DestroyWindow(hWnd);
                  break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        break;
case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);

        SelectObject(hdc, hRedBrush);
        SelectObject(hdc, redPen);

        hRegion1 = CreateEllipticRgn(g_circle.left, g_circle.top, g_circle.right,
                        g_circle.bottom);
        ExtSelectClipRgn(hdc, hRegion1, RGN_COPY);

        Ellipse(hdc, g_circle.left, g_circle.top, g_circle.right, g_circle.bottom);

        EndPaint(hWnd, &ps);
        break;

case WM_LBUTTONDOWN:
        prevX = LOWORD(lParam);
        prevY = HIWORD(lParam);
        break;

case WM_MOUSEMOVE:
        {

                if (wParam & MK_LBUTTON) {
                        RECT tempRect = g_circle;
                        HDC hDC = GetDC(hWnd);


                        hRegion1 = CreateEllipticRgn(g_circle.left, g_circle.top,
                                g_circle.right, g_circle.bottom);
                        ExtSelectClipRgn(hDC, hRegion1, RGN_COPY);

                        OffsetRect(&g_circle, LOWORD(lParam) –
                                prevX,HIWORD(lParam) - prevY);

                        hRegion2 = CreateEllipticRgn(g_circle.left, g_circle.top,
                                g_circle.right, g_circle.bottom);

                        ExtSelectClipRgn(hDC, hRegion2, RGN_DIFF);

                        SelectObject(hDC, whitePen);

                        Ellipse(hDC, tempRect.left, tempRect.top, tempRect.right,
                                tempRect.bottom);
```

```
                                        ExtSelectClipRgn(hDC, hRegion2, RGN_COPY);

                                        SelectObject(hDC, hRedBrush);
                                        SelectObject(hDC, redPen);

                                        Ellipse(hDC, g_circle.left, g_circle.top, g_circle.right,
                                                g_circle.bottom);

                                        prevX = LOWORD(lParam);
                                        prevY = HIWORD(lParam);
                                }

                        }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
   return FALSE;
}
```

Grafiksel Yol İşlemleri

Region bir bölge anlatan kavramdır. Region oluşturmak için düzgün şekillerden faydalanılır. Örneğin, bir daire bir poligon region olabilir, ya da bunların keşisim ya da birleşimlerinden region oluşturulabilir. Fakat düzgün olmayan şekillerden region oluşturmak mümkün değildir. Region' ın yanısıra grafiksel yol denilen region benzeri başka bir kavram daha kullanılmaktadır. Grafiksel yol işlemleri için bir grup API fonksiyonu kullanılmaktadır. Grafiksel yol regionda olduğu gibi dışarıda oluşturulup daha sonra DC alanına bağlanmamaktadır, zaten doğrudan DC alanında oluşturulmaktadır. Grafiksel yol kullanımı oldukça basittir. BeginPath ile EndPath fonksiyonları arasında programcının çağırdığı fonksiyonlardaki şekiller ucuca eklenerek bir yol oluşturulur.

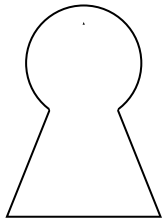
BOOL BeginPath(HDC hdc // handle to device context **);**

BOOL EndPath(HDC hdc // handle to device context **);**

BeginPath ile EndPath arasında kullanılacak fonksiyonlar şunlardır:

AngleArc	**LineTo**	**Polyline**
Arc	**MoveToEx**	**PolylineTo**
ArcTo	**Pie**	**PolyPolygon**
Chord	**PolyBezier**	**PolyPolyline**
CloseFigure	**PolyBezierTo**	**Rectangle**
Ellipse	**PolyDraw**	**RoundRect**
ExtTextOut	**Polygon**	**TextOut**

CloseFigure fonksiyonu son çizimin son noktası ile ilk noktayı birleştirerek şekli kapatır ve yeni bir şekle geçiş sağlar. EndPath fonksiyonu da işlemi bitirmeden önce şekli kapatmaktadır. Örneğin, anahtar deliği biçiminde bir kapalı şekil oluşturmak istesek, önce bir yay sonra bir doğru çizeriz ve CloseFigure ya da EndPath işlemiyle şekli kapatırız.



Bir grafiksel yol (path), SelectClipPath fonksiyonu ile DC'nin Clip Region'ı olarak oluşturulabilir.

```
BOOL SelectClipPath(
 HDC hdc,     // handle of device context
 int iMode    // clipping mode
);
```

Fonksiyonun 1. parametresi seçilecek DC'nin handle değeri, 2. parametresi seçime ilişkin moddur. Mod, ExtSelectClipRgn fonksiyonundaki gibidir. Eğer istenirse PathToRegion fonksiyonuyla DC'deki grafiksel yol, bir region durumuna getirilebilir.

```
HRGN PathToRegion( HDC hdc // handle to device context );
```

SİMGELERİN (icon) ÇALIŞMA ALANINA ÇİZİLMESİ

Handle değeri bilinen bir simge, DrawIcon API fonksiyonuyla çalışma alanına çizilebilir.

```
BOOL DrawIcon(
 HDC hDC,          // handle to device context
 int X,            // x-coordinate of upper-left corner
 int Y,            // y-coordinate of upper-left corner
 HICON hIcon       // handle to icon to draw
);
```

Fonksiyonun 1. parametresi DC'nin handle değeri, 2. ve 3. parametreleri simgenin sol üst köşe koordinatlarıdır. Son parametre, simgenin handle değeridir. Eğer simge kaynakta belirtildiyse LoadIcon fonksiyonu ile yüklenir. Eğer kaynakta belirtilmediyse LoadImage fonksiyonu kullanılarak kaynak yüklenip handle değeri alınabilir.

```
HICON LoadIcon(
 HINSTANCE hInstance,     // handle to application instance
 LPCTSTR lpIconName       // icon-name string or icon resource identifier
```

```
);
```

```
HANDLE LoadImage(
 HINSTANCE hinst,        // handle of the instance containing the image
 LPCTSTR lpszName,       // name or identifier of image
 UINT uType,             // type of image
 int cxDesired,          // desired width
 int cyDesired,          // desired height
 UINT fuLoad             // load flags
);
```

```
// icon.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];                                    // The title bar
text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_ICON, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_ICON);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
```

```
                DispatchMessage(&msg);
            }
        }

        return msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_ICON);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_ICON;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;
  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
```

```
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND      - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY      - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static HICON hIconTurk;
        static HRGN hRgn;

        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        hIconTurk = LoadIcon(hInst, MAKEINTRESOURCE(IDI_ICON_TURK));
                        hRgn = CreateEllipticRgn(100, 100, 132, 132);
                        if (hIconTurk == NULL || hRgn == NULL)
                                return -1;
                        break;
                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                                (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);

                        SelectClipRgn(hdc, hRgn);

                        DrawIcon(hdc, 100, 100, hIconTurk);

                        EndPaint(hWnd, &ps);
```

```
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                                return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
    return FALSE;
}
```

BELLEK DC'LERİ

DC (Device Context), yalnızca ekrana ilişkin bir kavram değildir; daha genel bir kavramdır. DC'nin ilişkili olduğu birim ekran, yazıcı yada bellek olabilir. Örneğin GetDC, GetWindowDC, BeginPaint gibi fonksiyonlar, ekrana ilişkin DC verirler. Bir DC belleğe ilişkinse yapılan çizimler de bellekte kalır. Fakat BitBlt gibi fonksiyonlarla bellekteki görüntü ekrana aktarılabilir. Bir DC'nin pixel çözünürlüğü, renk çözünürlüğü gibi çeşitli özellikleri de vardır. Verilen bir ekran DC'siyle uyumlu (yani aynı renk çözünürlüğünde) bir bellek DC'si CreateComptibleDC fonksiyonuyla elde edilebilir.

```
HDC CreateCompatibleDC(
 HDC hdc      // handle to the device context
);
```

Fonksiyon, bir görüntü DC'sini parametre olarak alır, onunla uyumlu bir bellek DC'si verir.

Bir bellek DC'si yaratıldığında henüz bir zemini yoktur. Bir zemin oluşturmak gerekir. Bellek DC'sinin zemini bir bitmap olabilir. O halde bir bitmap yaratıp SelectObject fonksiyonuyla seçerek zemin oluşturulabilir. Bitmap, HBITMAP türüyle temsil edilen bir çizim nesnesidir. Bitmap kaynaktan LoadBitmap fonksiyonuyla yüklenebilir yada doğrudan programın çalışma zamanı sırasında oluşturulabilir. Bitmap'in de renk çözünürlüğü söz konusudur. O anki DC ile uyumlu bir bitmap, CreateCompatibleBitmap fonksiyonuyla oluşturulabilir.

```
HBITMAP CreateCompatibleBitmap(
 HDC hdc,     // handle to device context
 int nWidth,  // width of bitmap, in pixels
```

```
 int nHeight    // height of bitmap, in pixels
);
```

Fonksiyonun 1. parametresi DC'nin handle değeri, 2. ve 3. parametreleri zemin olarak oluşturulacak bitmap'in genişlik ve yükseklik değerleridir.

BitBlt Fonksiyonu

Bu fonksiyon, çizim konusunun en önemli fonksiyonlarından biridir. Bir DC'nin belirli bir bölgesini başka bir DC'ye kopyalamakta kullanılır.

```
BOOL BitBlt(
 HDC hdcDest,        // handle to destination device context
 int nXDest,         // x-coordinate of destination rectangle's upper-left corner
 int nYDest,         // y-coordinate of destination rectangle's upper-left // corner
 int nWidth,         // width of destination rectangle
 int nHeight,        // height of destination rectangle
 HDC hdcSrc,         // handle to source device context
 int nXSrc,          // x-coordinate of source rectangle's upper-left corner
 int nYSrc,          // y-coordinate of source rectangle's upper-left corner
 DWORD dwRop         // raster operation code
);
```

Fonksiyonu 1. parametresi, kopyalamanın yapılacağı hedef DC'nin handle değeridir. 2. ve 3. parametreler dikdörtgensel bölgenin kopyalanacağı hedef DC'deki sol üst köşeyi belirtir. Sonraki iki parametre, kopyalanacak dikdörtgensel bölgenin genişlik ve yüksekliğidir. Sonraki parametreler sırasıyla kaynak DC ve onun kopyalanacağı dikdörtgensel bölgenin sol üst köşesidir. Kaynak DC ile hedef DC aynı olabilir. BitBlt fonksiyonunun son parametresi kullanılacak raster işlemini belirtir. Raster işlemleri çeşitli olabilir. Fakat bu parametre SRCCOPY biçiminde geçilirse kaynaktan hedefe kopyalama yapılır.

```
// icon.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];                                    // The title bar
text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
```

```
	HACCEL hAccelTable;

	// Initialize global strings
	LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
	LoadString(hInstance, IDC_ICON, szWindowClass, MAX_LOADSTRING);
	MyRegisterClass(hInstance);

	// Perform application initialization:
	if (!InitInstance (hInstance, nCmdShow))
	{
		return FALSE;
	}

	hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_ICON);

	// Main message loop:
	while (GetMessage(&msg, NULL, 0, 0))
	{
		if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProc;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_ICON);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= (LPCSTR)IDC_ICON;
	wcex.lpszClassName	= szWindowClass;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

	return RegisterClassEx(&wcex);
}

//
```

```
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND       - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY         - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static HICON hIconTurk;
        static HRGN hRgn;

        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        hIconTurk = LoadIcon(hInst, MAKEINTRESOURCE(IDI_ICON_TURK));
                        hRgn = CreateEllipticRgn(100, 100, 115, 115);
                        if (hIconTurk == NULL || hRgn == NULL)
                                return -1;
                        break;
                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
```

```
                {
                        case IDM_ABOUT:
                          DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                  (DLGPROC)About);
                          break;
                        case IDM_EXIT:
                          DestroyWindow(hWnd);
                          break;
                        default:
                          return DefWindowProc(hWnd, message, wParam, lParam);
                }
                break;
        case WM_PAINT:
                hdc = BeginPaint(hWnd, &ps);

                DrawIcon(hdc, 0, 0, hIconTurk);
                BitBlt(hdc, 100, 100, 32, 32, hdc, 0, 0, SRCCOPY);
                EndPaint(hWnd, &ps);
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
   return FALSE;
}
```

Son parametredeki raster işlemi kaynak DC'deki bir pixelin hedef DC'deki pixel ile nasıl işleme sokulacağını belirtir. SCRCOPY, hiçbir işleme sokmadan doğrudan kopyalamak için kullanılır.

Bir bitmap görüntü çizdirmenin adımları şunlardır:

1. Bitmap kaynaktan yada LoadImage fonksiyonuyla yüklenerek handle değeri elde edilir.
2. CreateCompetibleDC ile bir bellek DC'si yaratılır ve görüntülenecek bitmap SelectObject ile seçilerek bellek DC'sinin zemini yapılır.
3. WM_PAINT mesajında BitBlt fonksiyonu ile bellek DC'sinden görüntü DC'sine kopyalama yapılarak çizim gerçekleştirilir.

Handle değeri bilinen bir bitmap'in genişlik ve yüksekliği GetObject fonksiyonuyla elde edilebilir.

```
int GetObject(
 HGDIOBJ hgdiobj,        // handle to graphics object of interest
 int cbBuffer,           // size of buffer for object information
 LPVOID lpvObject        // pointer to buffer for object information
);
```

Aslında bu fonksiyon yalnızca bitmap'in değil, başka çizimlerinde bilgisini almaktadır. Fonksiyonun birinci parametresi bitmap'in handle değeri, ikinci parametresi üçüncü parametresiyle belirtilen yapının uzunluğu son parametresi ise hangi nesneye ilişkin bilgiler elde edeceksek o nesneye özgü yapının adresidir. Bitmap için kullanılacak yapı BITMAP isimli yapıdır. Örneğin;

```
BITMAP bmp;
GetObject(hBitmap, sizeof(bmp), &bmp);
```

Bitmapin genişliği BITMAP yapısının bmWidht ve bmHeight elemanlarından alınabilir. Örneğin, bitmap çizdirmek için WM_CREATE mesdajında şunlar yapılabilir.

```
        case WM_CREATE:
        {
                HDC hDC = GetDC(hWnd);
                hMemoryDC = CreateCompatibleDC(hDC);
                ReleaseDC(hWnd, hDC);
                hBitmap = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP1));
                SelectObject(hMemoryDC, hBitmap);
                GetObject(hBitmap, sizeof(bmp), &bmp);
        }
                break;
```

WM_PAINT mesajında da şu işlemler yapılır.

```
        case WM_PAINT:
                hdc = BeginPaint(hWnd, &ps);
                BitBlt(hdc, 0, 0, bmp.bmWidth, bmp.bmHeight, hMemoryDC, 0, 0, SRCCOPY);
                EndPaint(hWnd, &ps);
                break;
```

```
// BitmapApp.cpp : Defines the entry point for the application.

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];                                    // The title bar
text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
```

```
{
	// TODO: Place code here.
	MSG msg;
	HACCEL hAccelTable;

	// Initialize global strings
	LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
	LoadString(hInstance, IDC_BITMAPAPP, szWindowClass, MAX_LOADSTRING);
	MyRegisterClass(hInstance);

	// Perform application initialization:
	if (!InitInstance (hInstance, nCmdShow))
	{
		return FALSE;
	}

	hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_BITMAPAPP);

	// Main message loop:
	while (GetMessage(&msg, NULL, 0, 0))
	{
		if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return msg.wParam;
}

//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProc;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_BITMAPAPP);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= (LPCSTR)IDC_BITMAPAPP;
	wcex.lpszClassName	= szWindowClass;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

	return RegisterClassEx(&wcex);
}

//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);
```

```
   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}

//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        static HBITMAP hBitmap;
        static HDC hMemoryDC;
        static BITMAP bmp;

        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        {
                                HDC hDC = GetDC(hWnd);
                                hMemoryDC = CreateCompatibleDC(hDC);
                                ReleaseDC(hWnd, hDC);
                                hBitmap = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP1));
                                SelectObject(hMemoryDC, hBitmap);
                                GetObject(hBitmap, sizeof(bmp), &bmp);

                        }
                        break;


                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                                (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);

                        BitBlt(hdc, 0, 0, bmp.bmWidth, bmp.bmHeight, hMemoryDC, 0, 0, SRCCOPY);

                        EndPaint(hWnd, &ps);
                        break;
```

```
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                                return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
    return FALSE;
}
```

```
// BitmapApp.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];                                    // The title bar
text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPSTR     lpCmdLine,
                int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_BITMAPAPP, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
```

```
		return FALSE;
	}

	hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_BITMAPAPP);

	// Main message loop:
	while (GetMessage(&msg, NULL, 0, 0))
	{
		if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return msg.wParam;
}

//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProc;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance, (LPCTSTR)IDI_BITMAPAPP);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= (LPCSTR)IDC_BITMAPAPP;
	wcex.lpszClassName	= szWindowClass;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

	return RegisterClassEx(&wcex);
}

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	int wmId, wmEvent;
	PAINTSTRUCT ps;
	HDC hdc;
	static HBITMAP hBitmap;
	static HDC hMemoryDC;
	static BITMAP bmp;

	TCHAR szHello[MAX_LOADSTRING];
	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch (message)
	{
		case WM_CREATE:
			{
				HDC hDC = GetDC(hWnd);
				hMemoryDC = CreateCompatibleDC(hDC);
				ReleaseDC(hWnd, hDC);
				hBitmap = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP1));
				SelectObject(hMemoryDC, hBitmap);
				GetObject(hBitmap, sizeof(bmp), &bmp);

			}
			break;

		case WM_COMMAND:
			wmId    = LOWORD(wParam);
			wmEvent = HIWORD(wParam);    // Parse the menu selections:
			switch (wmId)
			{
				case IDM_ABOUT:
				  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
						(DLGPROC)About);
				  break;
				case IDM_EXIT:
				  DestroyWindow(hWnd);
				  break;
				default:
				  return DefWindowProc(hWnd, message, wParam, lParam);
			}
			break;
		case WM_PAINT:
			hdc = BeginPaint(hWnd, &ps);

			BitBlt(hdc, 0, 0, bmp.bmWidth, bmp.bmHeight, hMemoryDC, 0, 0, SRCCOPY);

			EndPaint(hWnd, &ps);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
			return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch (message)
```

```
        {
                case WM_INITDIALOG:
                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
   return FALSE;
}
```

```
// BitmapApp.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];                                    // The title bar
text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPSTR     lpCmdLine,
                int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_BITMAPAPP, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_BITMAPAPP);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
```

```
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }

        return msg.wParam;
}



//

ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style              = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra         = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon              = LoadIcon(hInstance, (LPCTSTR)IDI_BITMAPAPP);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_BITMAPAPP;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        static HBITMAP hBitmap;
```

```
        static HDC hMemoryDC;
        static BITMAP bmp;
        static HRGN hRegion;

        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        {
                                HDC hDC = GetDC(hWnd);
                                hMemoryDC = CreateCompatibleDC(hDC);
                                ReleaseDC(hWnd, hDC);
                                hBitmap = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP1));
                                SelectObject(hMemoryDC, hBitmap);
                                GetObject(hBitmap, sizeof(bmp), &bmp);
                                hRegion = CreateEllipticRgn(0, 0, bmp.bmWidth, bmp.bmHeight);

                        }
                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                                (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);

                        SelectClipRgn(hdc, hRegion);
                        BitBlt(hdc, 0, 0, bmp.bmWidth, bmp.bmHeight, hMemoryDC, 0, 0, SRCCOPY);

                        EndPaint(hWnd, &ps);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

//...
```

Standart Diyalog Pencerelerinin Kullanımı

Windows programlarında çok sık karşılaştığımız renk seçme diyalog penceresi dosya seçme diyalog penceresi, bul ve değiştir diyalog penceresi gibi diyalog pencereleri aslında API fonksiyonlarıyla çıkartılan standart pencerelerdir.

Renk Seçme Diyalog Penceresinin Kullanımı

Renk seçme diyalog penceresi ChooseColor API fonksiyonuyla çıkartılır.

```
BOOL ChooseColor(
 LPCHOOSECOLOR lpcc  // pointer to structure with initialization  data
);
```

Fonksiyon CHOOSECOLOR isimli bir yapının adresini alır. Programcı yapının bazı elemanlarını doldurur, fonksiyon da seçilen rengi yapının bir elemanına yazar.

```
typedef struct {
 DWORD lStructSize;
 HWND hwndOwner;
 HWND hInstance;
 COLORREF rgbResult;
 COLORREF* lpCustColors;
 DWORD Flags;
 LPARAM lCustData;
 LPCCHOOKPROC lpfnHook;
 LPCTSTR lpTemplateName;
} CHOOSECOLOR;
```

CHOOSECOLOR yapısının lStructSize elemanı yapının uzunluğunu alır. Yapının hwndOwner elemanı açılan diyalog penceresinin üst penceresi olacak pencerenin handle değeri, hInstance elemanı ise PE formatının yüklenme adresini alır. Diyalog penceresi kapatıldıktan sonra seçilen renk yapının rgbResult elemanından alınır.

Yapının COLORREF türünden lpCustColors gösterici elemanı, COLORREF cinsinden 16 elemanlı bir dizinin başlangıç adresini tutmalıdır. Programcının custom colors olarak seçtiği renkler bu dizide saklanır ve diyalog penceresi ilk açıldığında buradaki renkler görüntülenir. Yapının Flags elemanı CC_XXX biçimindeki sembolik sabitlarden oluşur. Örneğin, CC_RGBINIT, yapının rgbResult elemanında belirtilen rengi başlangıçtaki seçim rengi olarak gösterir. CC_FULLOPEN, diyalog penceresinin tam olarak açılmasını sağlar. Yapının diğer eleamnları özel konulara ilişkindir, 0 geçilebilir. ChooseColor fonksiyonunun geri dönüş değeri diyalog penceresinden OK tuşuyla çıkılmışsa 0 dışı bir değer, CANCEL tuşuyla çıkılmışsa 0 değeridir.

```
// colordlg.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"
#include <commdlg.h>

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
```

```
TCHAR szWindowClass[MAX_LOADSTRING];                    // The title bar
text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_COLORDLG, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_COLORDLG);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}

//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style              = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra         = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon              = LoadIcon(hInstance, (LPCTSTR)IDI_COLORDLG);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_COLORDLG;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);
```

```
	return RegisterClassEx(&wcex);
}

//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	int wmId, wmEvent;
	PAINTSTRUCT ps;
	HDC hdc;
	static COLORREF custColors[16];
	CHOOSECOLOR cc = {sizeof(cc)};

	TCHAR szHello[MAX_LOADSTRING];
	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch (message)
	{
		case WM_COMMAND:
			wmId    = LOWORD(wParam);
			wmEvent = HIWORD(wParam);
			// Parse the menu selections:
			switch (wmId)
			{
			case ID_FILE_COLOR:
				cc.hInstance = (HWND) hInst;
				cc.Flags = CC_RGBINIT;
				cc.hwndOwner = hWnd;
				cc.lpCustColors = custColors;
				cc.rgbResult = RGB(255, 255, 255);
				if (ChooseColor(&cc)) {
					HBRUSH hBrush = CreateSolidBrush(cc.rgbResult), hOldBrush;

					hOldBrush = (HBRUSH) SetClassLong(hWnd,
						GCL_HBRBACKGROUND, (LONG) hBrush);
					DeleteObject(hOldBrush);
					InvalidateRect(hWnd,NULL, TRUE);
				}


				break;
				case IDM_ABOUT:
				  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
```

```
                                        (DLGPROC)About);
                                break;
                            case IDM_EXIT:
                                DestroyWindow(hWnd);
                                break;
                            default:
                                return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);
                        // TODO: Add any drawing code here...
                        RECT rt;
                        GetClientRect(hWnd, &rt);
                        DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                        EndPaint(hWnd, &ps);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

Dosya Seçme Diyalog Penceresinin Kullanımı

Dosya seçme diyalog penceresi Open File ve Save As File olmak üzere iki biçimdedir. Open File penceresi GetOpenFileName fonksiyonu ile Save As File Name diyalog penceresi GetSaveFileName fonksiyonu ile açılır.

```
BOOL GetOpenFileName(
 LPOPENFILENAME lpofn  address of structure with initialization data
);
```

```
BOOL GetSaveFileName(
 LPOPENFILENAME lpofn address of structure with initialization data
);
```

Her iki fonksiyon da OPENFILENAME türünden bir yapının adresini alır. Bu yapı maalesef çok uzun bir yapıdır, fakat yapının az sayıda elemanı önemlidir. Diğer elemanlara 0 girildiğinde default değer anlamına gelir. Böylece programcı yapının elemanlarını önce sıfırlayıp sonra yalnızca önemli elemanlara değer atayabilir. Yapının önemli elemanları şunlardır:

DWORD lStructSize: Bu elemana yapının uzunluğu girilmelidir.

HWND hwndOwner: Bu elemana açılacak diyalog penceresinin üst penceresi olacak pencerenin handle değeri girilmelidir.

HINSTANCE hInstance: PE formatının belleğe yüklenme adresi girilmelidir.

LPCTSTR lpstrFilter: Bu elamana filtreleme yazısının adresi girilmelidir. Fitreleme yazısı, her biri NULL karakterlerle biten fakat sonu iki NULL karakterle biten çiftlerden oluşur. Çiftin birinci elemanı, filtreleme

combo box'ında çıkacak olan yazıyı; ikinci elemanı, gerçek filtreleme joker karakterlerini içerir. Örneğin aşağıdaki yazıda iki çift belirlenmiştir:

"All Files(*.*)\0 *.*\0 Bmp Files(*.bmp)\0 *.bmp\0"

birinci çift ikinci çift

DWORD nFilterIndex: Bu eleman, diyalog penceresi açıldığında hangi filtreleme yazısının default olarak gösterileceğini belirtir. 0 orijinlidir.

LPTSTR lpstrFile, DWORD nMaxFile: Yapının bu iki elemanı, seçilen dosya isminin yerleştirileceği adresi ve o dizinin uzunluğunu belirtir. Windows'ta en büyük dosya ismi uzunluğu 260 karakterdir. Buraya seçilen dosyaya ilişkin tüm path ismi yerleştirilmektedir. BlpstrFile elemanına başlangıçta yerleştirilecek yazı, default olarak diyalog penceresinde görüntülenebilir.

LPTSTR lpstrFileTitle, DWORD nMaxFileTitle: Bu iki eleman sırasıyla yalnızca dosya isminin ve uzantısının yerleştirileceği dizinin adresini ve uzunluğuna alır.

LPCTSTR lpstrInitialDir: Diyalog penceresi açıldığında başlangıçta görüntülenecek dizini belirtir.

LPCTSTR lpstrTitle: Bu eleman, diyalog penceresini başlık kısmında görüntülenecek yazıyı belirler.

DWORD Flags: Bu eleman, OFN_XXX biçimindeki sembolik sabitlerden oluşturulmaktadır.

Örneğin, File Dialog penceresini açmak için basit bir kod şöyle olabilir:

```
#include <commdlg.h>

switch (message)
{
        .....
        case ID_FILE_OPENX:
                OnOpenFile(hWnd);
                break;
        .....
}

void OnOpenFile(HWND hWnd)
{
        OPENFILENAME ofn = {sizeof(ofn)};
        char fileName[MAX_PATH] = "";

        ofn.hInstance = hInst;
        ofn.hwndOwner = hWnd;
        ofn.lpstrFilter = "All Files(*.*)\0*.*\0Text Files (*.txt)\0*.txt\0";
        ofn.lpstrFile = fileName;
        ofn.nMaxFile = MAX_PATH;

        if (GetOpenFileName(&ofn)) {
                MessageBox(hWnd, fileName, "Message", MB_OK);

        }
```

}

SaveAs Dialog penceresi de aynı biçimde fakat GetSaveFileName fonksiyonuyla açılmaktadır.

```
// FileDialog.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"
#include <commdlg.h>

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];                                    // The title bar
text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

void OnOpenFile(HWND hWnd);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_FILEDIALOG, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_FILEDIALOG);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
```

```
}

//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style		= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProc;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon		= LoadIcon(hInstance, (LPCTSTR)IDI_FILEDIALOG);
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= (LPCSTR)IDC_FILEDIALOG;
	wcex.lpszClassName	= szWindowClass;
	wcex.hIconSm		= LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

	return RegisterClassEx(&wcex);
}

//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	int wmId, wmEvent;
	PAINTSTRUCT ps;
	HDC hdc;
	TCHAR szHello[MAX_LOADSTRING];
	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	switch (message)
	{
		case WM_COMMAND:
			wmId    = LOWORD(wParam);
			wmEvent = HIWORD(wParam);
			// Parse the menu selections:
			switch (wmId)
			{
				case ID_FILE_OPENX:
```

```
                                OnOpenFile(hWnd);
                                break;

                        case IDM_ABOUT:
                          DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                  (DLGPROC)About);
                          break;
                        case IDM_EXIT:
                          DestroyWindow(hWnd);
                          break;
                        default:
                          return DefWindowProc(hWnd, message, wParam, lParam);
                    }
                    break;
            case WM_PAINT:
                    hdc = BeginPaint(hWnd, &ps);
                    // TODO: Add any drawing code here...
                    RECT rt;
                    GetClientRect(hWnd, &rt);
                    DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                    EndPaint(hWnd, &ps);
                    break;
            case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
            default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

//...
void OnOpenFile(HWND hWnd)
{
        OPENFILENAME ofn = {sizeof(ofn)};
        char fileName[MAX_PATH] = "";

        ofn.hInstance = hInst;
        ofn.hwndOwner = hWnd;
        ofn.lpstrFilter = "All Files(*.*)\0*.*\0Text Files (*.txt)\0*.txt\0";
        ofn.lpstrFile = fileName;
        ofn.nMaxFile = MAX_PATH;

        if (GetOpenFileName(&ofn)) {
                MessageBox(hWnd, fileName, "Message", MB_OK);

        }

}
```

STANDART C FONKSİYONLARINA BENZER API FONKSİYONLARI

Çok kullanılan standart C fonksiyonlarının bazıları birer API fonksiyonu biçiminde de yazılmıştır. Programcının standart C fonksiyonları yerine API fonksiyonlarını tercih etmesi daha uygundur. Çünkü bu fonksiyonlar boşuna .exe içerisinde yer kaplamazlar. Önemli standart C fonksiyonlarına benzer API fonksiyonları şunlardır:

API Fonksiyonu	Standart C Fonksiyonu
lstrlen	strlen

lstrcpy	strcpy
lstrcpyn	strncpy
lstrcmpi	stricmp
lstrcmp	strcmp
lstrcat	strcat
CopyMemory	memcpy
ZeroMemory	?
FillMemory	memset
MoveMemory	memmove

MDI UYGULAMALARI

Bir ana pencerenin aynı biçimde davranan alt pencerelerden oluşan uygulamalara MDI (Multiple Document Interface) uygulamaları denilmektedir. MDI uygulamaları eskiden çok tercih edilmekteydi. Fakat Microsoft daha sonra taktik değiştirerek kendi ürünlerini MDI biçimi yerine bir programın birden fazla kez çalıştırılması biçimine dönüştürmüştür. Fakat hala MDI uygulamaları popülerliğini korumaktadır. Windows API programlama sistemi MDI uygulamalarını bir grup API fonksiyonuyla desteklemektedir.

MDI Terminolojisi

Programın ana penceresine frame penceresi (Frame Window) denilmektedir. Frame penceresinin çalışma alanını kaplayan gri renkli alt pencereye ise client penceresi (Client Window) denilmektedir. Nihayet kullanıcı ile etkileşen asıl işin yapıldığı pencerelere doküman pencereleri (Document Windows) denilmektedir. Client penceresi, frame penceresinin alt penceresi doküman pencereleri ise client penceresinin alt pencereleri durumundadır.

Bir MDI uygulamasında frame penceresine ilişkin pencere fonksiyonu ve doküman pencerelerine ilişkin pencere fonksiyonu programcı tarafından yazılır. Doküman pencerelerinin pencere fonksiyonu aynıdır, böylece bunların davranışları da aynı olur. Client penceresi doküman pencereleri ile ana pencere arasında ara birim oluşturan bir penceredir. Client penceresinin pencere fonksiyonu User32.dll içerisinde hazır bir biçimde bulunmaktadır. Client penceresi ana pencere ile doküman pencereleri arasındaki iletişimi sağlar. Örneğin, doküman pencerelerinde olaylar gerçekleştiğinde bu pencereler client penceresine mesaj gönderir, client penceresi de ana pencereye mesaj göndererek ana pencerenin bu işlemden haberdar olmasını sağlar.

Bir MDI uygulamasının yazım aşamaları özetle şöyledir:

1. Programcı ana pencere için ve doküman pencereleri için iki ayrı pencere sınıfı register eder. Yani bu pencerelerin pencere fonksiyonlarını programcı yazacaktır.
2. Ana pencerenin WM_CREATE mesajında MDICLIENT sınıf ismi kullanılarak, client penceresi yaratılır. Client penceresinin ana pencerenin çalışma alanını kaplaması client penceresi tarafından otomatik yapılan bir işlemdir.
3. Doküman pencereleri client penceresine mesaj gönderilerek client penceresine yarattırılır.

MDI İşleminin Ayrıntıları

Bir MDI uygulamasının ana penceresinde işlenmeyen mesajlar DefWindowProc fonksiyonuyla değil, DefFrameProc fonksiyonuyla işlenir.

```
LRESULT DefFrameProc(
 HWND hWnd,              // handle to MDI frame window
 HWND hWndMDIClient,     // handle to MDI client window
 UINT uMsg,              // message
```

```
 WPARAM wParam,        // first message parameter
 LPARAM lParam         // second message parameter
);
```

Benzer biçimde doküman pencerelerine ,ilişkin pencere fonksiyonlarında işlenmeyen mesajlar DefMDIChildProc fonksiyonuna işlettirilmelidir.

```
LRESULT DefMDIChildProc(
 HWND hWnd,            // handle to MDI child window
 UINT uMsg,            // message
 WPARAM wParam,        // first message parameter
 LPARAM lParam         // second message parameter
);
```

Doküman pencerelerinin yaratılması için client penceresine WM_MDICREATE mesajı gönderilmelidir.

```
WM_MDICREATE
wParam = 0;                                          // not used; must be zero
lParam = (LPARAM) (LPMDICREATESTRUCT) lpmdic;        // creation data
```

Mesajın wParam parametresi kullanılmaz, sıfırlanmalıdır. lParam parametresi MDICREATESTRUCT isimli bir yapının adresini alır. Yani programcı bu türden bir yapı nesnesi tanımlamalı onun içini doldurmalıdır.

```
typedef struct tagMDICREATESTRUCT {
        LPCTSTR szClass;
        LPCTSTR szTitle;
        HANDLE hOwner;
        int x;
        int y;
        int cx;
        int cy;
        DWORD style;
        LPARAM lParam;
} MDICREATESTRUCT;
```

Aslında MDICREATESTRUCT yapısının elemanları adeta CreateWindow fonksiyonunun parametrelerini oluşturmaktadır.

Doküman pencereleri yaratılırken ayrıca WNDCLASS yapısının cdwndExtra elemanı 0 yerine sizeof(HANDLE) kadar olmalıdır.

Ana pencerenin WM_CREATE mesajında client penceresi yaratılırken CreateWindow fonksiyonunun son parametresi CLIENTCREATESTRUCT biçiminde bir yapının adresi olarak girilmelidir. CreateWindow fonksiyonunun son parametresi WM_CREATE mesajına geçirilmektedir.

CLIENTCREATESRUCT yapısı, aşağıdaki gibi iki elemanlı bir yapıdır:

```
typedef struct tagCLIENTCREATESTRUCT { // ccs
   HANDLE hWindowMenu;
   UINT   idFirstChild;
} CLIENTCREATESTRUCT;
```

Yapının hWindowMenu elemanı, ilgili doküman penceresi aktif hale geldiğinde gösterilecek menüyü belirtmektedir. Menu kullanımı söz konusu değilse NULL geçilebilir. Yapının idFirstChild elemanı, doküman pencereleri için bir ilk taban değer belirtmektedir. Her doküman penceresinde bu id değeri otomatik 1 arttırılır. Ayrıca client penceresini yaratırken pencere biçimi olarak WS_CLIPCHILDREN biçiminde kullanmak faydalıdır.

client pencere oluşturalım:

```
#include <windows.h>

#define ID_FIRST_CHILD                  1000

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK DocumentWndProc(HWND hWnd, UINT message, WPARAM wParam,
                                        LPARAM lParam);

HINSTANCE g_hInstance;
HWND g_hClientWnd;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                        int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = sizeof(HANDLE);
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Document";
                wndClass.lpfnWndProc = (WNDPROC) DocumentWndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }
```

```
	g_hInstance = hInstance;

	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	CLIENTCREATESTRUCT clientCreate;

	switch (message) {
		case WM_CREATE:
			clientCreate.hWindowMenu = NULL;
			clientCreate.idFirstChild = ID_FIRST_CHILD;

			g_hClientWnd = CreateWindow("MDICLIENT", NULL,
					WS_CHILD|WS_VISIBLE|WS_CLIPCHILDREN,
						0, 0, 0, 0, hWnd, (HMENU) 1, g_hInstance, &clientCreate);
			if (g_hClientWnd == NULL)
				return -1;
		break;

		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefFrameProc(hWnd, g_hClientWnd, message, wParam, lParam);
	}
	return 0;
}

LRESULT CALLBACK DocumentWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	return DefMDIChildProc(hWnd, message, wParam, lParam);
}
```

Döküman Pencerelerindeki Bilgilerin İşlenmesi

Doküman pencerelerinin pencere fonksiyonlarının davranışları aynıdır. Fakat bu durum, bu doküman pencereleri üzerinde oluşan dataların aynı olacağı anlamına gelmez. Örneğin doküman penceresinin pencere fonksiyonu bir yaz-boz tahtası uygulaması olsun. Doküman penceresinin WM_PAINT mesajında bu çizimlerin yapılabilmesi için her doküman penceresine ilişkin dataların ayrı bir biçimde tutulması gerekir.

Bunu sağlamanın en pratik yolu doküman pencerelerinin tek olan handle değerlerinden faydalanmaktır. Örneğin datalar handle değerinden hareketle o handle değerine ilişkin bir veri yapısında saklanabilir. WM_PAINT mesajında da handle değerinden hareketle ilgili pencerenin bilgilerine erişilebilir. Yani pencere fonksiyonunun kodları aynıdır fakat pencere fonksiyonuna geçirilen hWnd değeri farklı olduğu için bu mesajlarda olayın gerçekleştiği pencere üzerinde işlemler yapılabilir.

SetWindowLong Ve GetWindowLong Fonksiyonlarının Ayrıntıları

Pencerenin handle alanında programcı kendisinin kullanabileceği alanlar ayırabilmektedir. Bu alanların ayrılması, pencere sınıfı register ettirilirken yapılır. WNDCLASS yapısının cbWndExtra elemanı pencerenin handle alanında kullanım için ne kadar yer ayrılacağını belirtmektedir. Örneğin biz buraya 12 değerini girersek bu sınıftan yaratılacak tüm pencerelerin handle alanlarında 12 byte yer ayrılacaktır. Ayrılan bu yerlere SetWindowLong fonksiyonuyla bilgi yerleştirilebilir. Bu bilgiler GetWindowLong fonksiyonuyla alınabilir. SetWindowLong fonksiyonunda 2. parametredeki GWL_XXX değerleri yerine 0, 4, 8 gibi index değerleri girilirse o indexten itibaren 4 byte'lık tahsis edilmiş alana değer girilebilir. Yani buraya 4 girmek demek ayırdığımız 12 byte'lık alanın 4 – 8 byte arasına değer yerleştirmek demektir. GetWindowLong fonksiyonu da benzer biçimde çalışmaktadır.

İşte biz MDI uygulamalarında örneğin her doküman penceresinin çizimlerini farklı bir bağlı listede tutacaksak bağlı listenin ilk düğümünü gösteren göstericiyi pencerenin handle alanında tutabiliriz. Böylece pencere fonksiyonuna geçirilen hWnd değerinden hareketle her pencerenin kendi datasına aynı kodla erişilebilir.

Client Penceresine Gönderilen Mesajlar

Programcı client penceresine WM_MDIXXX biçiminde mesajlar göndererek bazı faydalı işlemleri yapabilir. Örneğin, client penceresine WM_MDICASCADE mesajı gönderilirse client penceresi, doküman pencerelerini kademeli biçime sokar.

Client penceresine WM_MDITILE mesajı gönderildiğinde client penceresi doküman pencerelerini çalışma alanını bölüşecek biçime getirir. Mesajın wParam parametresi bölüşümün nasıl yapılacağını belirtmektedir. Bu değer MDITILE_HORIZONTAL, MDITILE_SKIPDISABLED, MDITILE_VERTICAL biçiminde olabilir. lParam parametresi kullanılmamaktadır, sıfırlanmalıdır.

WM_MDIACTIVATE mesajı programlama yoluyla belirli bir doküman penceresini aktif hale getirmek için kullanılır. Mesajın wParam parametresine aktif hale getirilecek olan doküman penceresinin handle değeri girilir. lParam parametresi kullanılmaz, sıfırlanmalıdır.

---

bak: genericMDI4 -> dosyası

```
//genericMDI.cpp

#include <windows.h>
#include <list>
#include "resource.h"

#define ID_FIRST_CHILD                  1000

using namespace std;

typedef struct tagWNDINFO {
        list<POINT> pointList;
} WNDINFO;

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
```

```
LRESULT CALLBACK DocumentWndProc(HWND hWnd, UINT message, WPARAM wParam,
                                                        LPARAM lParam);

HINSTANCE g_hInstance;
HWND g_hClientWnd;

 int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                        int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
                wndClass.lpszClassName = "Generic";
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = sizeof(WNDINFO *);
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = NULL;
                wndClass.lpszClassName = "Document";
                wndClass.lpfnWndProc = (WNDPROC) DocumentWndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        g_hInstance = hInstance;

        hWnd = CreateWindow("Generic", "Generic App",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
```

```
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	CLIENTCREATESTRUCT clientCreate;
	MDICREATESTRUCT mdiCreate;


	switch (message) {
		case WM_CREATE:
			clientCreate.hWindowMenu = NULL;
			clientCreate.idFirstChild = ID_FIRST_CHILD;

			g_hClientWnd = CreateWindow("MDICLIENT", NULL,
					WS_CHILD|WS_VISIBLE|WS_CLIPCHILDREN,
					0, 0, 0, 0, hWnd, (HMENU) 1, g_hInstance, &clientCreate);
			if (g_hClientWnd == NULL)
				return -1;
			break;
		case WM_COMMAND:
			switch (LOWORD(wParam)) {
				case ID_FILE_NEWX:
					mdiCreate.szClass = "Document";
					mdiCreate.x = CW_USEDEFAULT;
					mdiCreate.y = CW_USEDEFAULT;
					mdiCreate.cx = CW_USEDEFAULT;
					mdiCreate.cy = CW_USEDEFAULT;
					mdiCreate.style = 0;
					mdiCreate.lParam = 0;
					mdiCreate.hOwner = g_hInstance;
					mdiCreate.szTitle = "Test";

					SendMessage(g_hClientWnd, WM_MDICREATE, 0,
							(LPARAM) &mdiCreate);
					break;
				case ID_WINDOW_CASCADEX:
					SendMessage(g_hClientWnd, WM_MDICASCADE, 0, 0);
					break;
				case ID_WINDOW_TILEX :
					SendMessage(g_hClientWnd, WM_MDITILE,
							MDITILE_VERTICAL, 0);
					break;
			}
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefFrameProc(hWnd, g_hClientWnd, message, wParam, lParam);
	}
	return 0;
}

LRESULT CALLBACK DocumentWndProc(HWND hWnd, UINT message, WPARAM wParam,
					LPARAM lParam)
{
	static int prevX, prevY;
	HDC hDC;
	WNDINFO *pWndInfo;
```

```
PAINTSTRUCT ps;

switch (message) {
	case WM_CREATE:
		pWndInfo = new WNDINFO;
		SetWindowLong(hWnd, 0, (LONG) pWndInfo);
		break;

	case WM_LBUTTONDOWN:
		prevX = LOWORD(lParam);
		prevY = HIWORD(lParam);
		break;

	case WM_MOUSEMOVE:
		{
			POINT pt;

			if (!(wParam & MK_LBUTTON))
				break;

			pWndInfo = (WNDINFO *) GetWindowLong(hWnd, 0);


			pt.x = prevX;
			pt.y = prevY;

			pWndInfo->pointList.push_back(pt);

			pt.x = LOWORD(lParam);
			pt.y = HIWORD(lParam);

			pWndInfo->pointList.push_back(pt);

			hDC = GetDC(hWnd);
			MoveToEx(hDC, prevX, prevY, NULL);
			LineTo(hDC, LOWORD(lParam), HIWORD(lParam));

			prevX = LOWORD(lParam);
			prevY = HIWORD(lParam);

			ReleaseDC(hWnd, hDC);
		}
		break;
	case WM_PAINT:
		{
			hDC = BeginPaint(hWnd, &ps);

			pWndInfo = (WNDINFO *) GetWindowLong(hWnd, 0);

			list<POINT>::iterator iter = pWndInfo->pointList.begin();

			while (iter != pWndInfo->pointList.end()) {
				MoveToEx(hDC, iter->x, iter->y, NULL);
				++iter;
				LineTo(hDC, iter->x, iter->y);
				++iter;
			}

			EndPaint(hWnd, &ps);
		}
		break;

	default:
```

```
                return DefMDIChildProc(hWnd, message, wParam, lParam);

        }
        return 0;
}
```

Kaydırma Çubuklarının Kullanımı
Kaydırma çubukları, iki biçimde bulunabilir:

1. Pencerenin doğal kaydırma çubukları biçiminde
2. Bir alt pencere yani kontrol biçiminde

Pencerenin doğal kaydırma çubukları, pencere yaratılırken WS_HSCROLL ve WS_VSCROLL pencere biçimleri kullanılarak yaratılır. Bunlar ayrı bir kontrol değildir; pencerenin bir parçasıdır. Yani ayrı birer handle değerleri yoktur. Kontrol biçiminde kaydırma çubukları CreateWindow fonksiyonunda "scrollbar" sınıf ismiyle yaratılırlar. Kaydırma çubuğu kontrolünde şu işlemler yapılabilir:

1. Çubuğun ucundaki oklara tıklanabilir.
2. Yürütecin iki ara bölgesine tıklanabilir.
3. Yürüteç, sürüklenerek bırakılabilir.

Kaydırma çubuğu, yalnızca mesaj gönderen basit bir kontroldür. Yoksa kaydırma işleminin kendisini yapmaz.

Kaydırma Çubuklarının Kullanımı

Pencerenin doğal kaydırma çubukları ile kontrol biçimindeki kaydırma çubuklarının kullanımı benzerdir. Öncelikle kaydırma çubuğunun değişim aralığını minimum ve maksimum olarak set etmek gerekir. Eğer bu set işlemi yapılmazsa minimum = 0; maximum = 100 anlaşılır. Bu işlem SetScrollRange API fonksiyonuyla yapılmaktadır.

```
BOOL SetScrollRange(
 HWND hWnd,        // handle to window with scroll bar
 int nBar,         // scroll bar flag
 int nMinPos,      // minimum scrolling position
 int nMaxPos,      // maximum scrolling position
 BOOL bRedraw      // redraw flag
);
```

Fonksiyonun 1. parametresi eğer kaydırma çubuğu pencerenin doğal kaydırma çubuğu ise pencerenin handle değeri, eğer kontrol biçimindeki kaydırma çubuğu söz konusu ise onun handle değeridir. Fonksiyonun 2. parametresi SB_CTL, SB_HORZ yada SB_VERT değerlerinden birini içerir. Eğer ayarlanacak çubuk pencerenin doğal düşey kaydırma çubuğu ise SB_VERT; pencerenin doğal yatay kaydırma çubuğu ise SB_HORZ; kontrol biçiminde bir kaydırma çubuğu ise SB_CTL biçiminde girilmelidir. Fonksiyonun sonraki iki parametresi min ve max değerleridir. Son parametre, görüntünün o anda tazelenip tazelenmeyeceğini belirtir, TRUE geçilebilir.

Yürütecin Konumlandırılması

Yürütecin konumlandırılması SetScrollPos fonksiyonuyla yapılmaktadır. yürütecin sürüklenip bırakılması sonucunda bile yürüteç otomatik konumlandırılmaz.

```
int SetScrollPos(
 HWND hWnd,        // handle to window with scroll bar
 int nBar,         // scroll bar flag
 int nPos,         // new position of scroll box
 BOOL bRedraw      // redraw flag
);
```

Birinci parametre pencerenin handle değeri, ikinci parametre SB_HORZ, SB_VERT yada SB_CTL değerlerinden biridir. Fonksiyonun 3. parametresi konumlandırma yapılacak pozisyondur. Son parametre, TRUE geçilmelidir.

Yürütecin Konumunun Alınması

Yürütecin o anki konumu GetScrollPos fonksiyonu ile alınabilir.

```
int GetScrollPos(
 HWND hWnd,        // handle to window with scroll bar
 int nBar          // scroll bar flags
);
```

Kaydırma Çubuklarının Gönderdiği Mesajlar

Kaydırma çubukları, pencereye WM_HSCROLL ve WM_VSCROLL mesajlarını göndermektedir. Mesajın HIWORD(wParam) parametresi kaydırma çubuğunun yeni olması gereken pozisyonunu verir. LOWORD(wParam), mesajın hangi işlem sonucunda geldiğini belirtmektedir. Bu değer şunlar olabilir:

— Eğer kaydırma çubuğunun oklarına klik yapılmışsa ve yatay kaydırma çubuğu söz konusu ise SB_LEFT, SB_RIGHT; düşey kaydırma çubuğu söz konusu ise SB_UP, SB_DOWN
— Eğer yürütecin aralarına klik yapılmışsa ve yatay kaydırma çubuğu söz konusu ise SB_PAGERIGHT, SB_PAGELEFT; eğer düşey kaydırma çubuğu söz konusu ise SB_PAGEDOWN, SB_PAGEUP
— Yürüteç hareket ettirildiği sürece SB_THUMBTRACK mesajı gönderilir.
— Yürüteç sürüklenip bırakıldığında SB_THUMBPOSITION gönderilmektedir.

Value	Meaning
SB_ENDSCROLL	Ends scroll.
SB_LEFT	Scrolls to the upper left.
SB_RIGHT	Scrolls to the lower right.
SB_LINELEFT	Scrolls left by one unit.
SB_LINERIGHT	Scrolls right by one unit.
SB_PAGELEFT	Scrolls left by the width of the window.
SB_PAGERIGHT	Scrolls right by the width of the window.
SB_THUMBPOSITION	The user has dragged the scroll box (thumb) and released the mouse button. The *nPos* parameter indicates the position of the scroll box at the end of the drag operation.
SB_THUMBTRACK	The user is dragging the scroll box. This message is sent repeatedly until the user releases the mouse button. The *nPos* parameter indicates the position that the scroll box has been dragged to.

SB_THUMBTRACK ve SB_THUMBPOSITION mesajlarında programcı HIWORD(wParam) değerinden yürütecin konumunu alabilir.

Özetle programcının kaydırma çubuğu mesajlarını işlemesi için aşağıdaki gibi bir switch oluşturması gerekir:

case WM_HSCROLL:

```
	switch (LOWORD(wParam)) {
		case SB_LINELEFT:
			...
			break;
		case SB_LINERIGHT:
			...
			break;
		case SB_PAGEUP:
			...
			break;
		case SB_PAGEDOWN:
			...
			break;
		case SB_THUMBTRACK:
			...
			break;
		case SB_THUMBPOSITION:
			...
			break;
	}
	break;
```

Programcının bu mesajlara karşı iki şey yapması gerekir:

1. Yürüteci konumlandırması
2. Kaydırma işleminde ne yapılması isteniyorsa onları yapması

bak: scrollbar

//scrollbar.c

```
#include <windows.h>


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
			int nCmdShow)
{
	WNDCLASS wndClass;
	HWND hWnd;
	MSG message;

	if (!hPrevInstance) {
		wndClass.style = CS_HREDRAW | CS_VREDRAW;
		wndClass.cbClsExtra = 0;
		wndClass.cbWndExtra = 0;
		wndClass.hInstance = hInstance;
		wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
		wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
		wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
		wndClass.lpszMenuName = NULL;
		wndClass.lpszClassName = "Generic";
		wndClass.lpfnWndProc = (WNDPROC) WndProc;
		if (!RegisterClass(&wndClass))
			return -1;
	}
```

```
	hWnd = CreateWindow("Generic", "Generic App",
		WS_OVERLAPPEDWINDOW|WS_HSCROLL|WS_VSCROLL,
		CW_USEDEFAULT,
		0,
		CW_USEDEFAULT,
		0,
		NULL,
		NULL,
		hInstance,
		NULL);

	if (!hWnd)
		return -1;

	ShowWindow(hWnd, SW_RESTORE);
	UpdateWindow(hWnd);
	while (GetMessage(&message, 0, 0, 0)) {
		TranslateMessage(&message);
		DispatchMessage(&message);
	}
	return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static int pos;

	switch (message) {
		case WM_LBUTTONDOWN :
			pos += 5;
			SetScrollPos(hWnd, SB_VERT, pos, TRUE);
			break;
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		default:
		  return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

Scroll yada kaydırma çubukları scrollbar sınıf ismiyle kontrol biçiminde de yaratılabilir. bak: controlscroll.cpp

```
// controlscroll.cpp : Defines the entry point for the application.

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
```

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_CONTROLSCROLL, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_CONTROLSCROLL);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_CONTROLSCROLL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_CONTROLSCROLL;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
```

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        static HWND hScroll;
        static int pos;

        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        hScroll = CreateWindow("scrollbar", "", WS_CHILD|WS_VISIBLE|SBS_HORZ,
                                                100, 100, 400, 30, hWnd, (HMENU) 100 , hInst, NULL);
                        SetScrollRange(hScroll, SB_CTL, 0, 255, TRUE);
                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                          (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_HSCROLL :
                        switch(LOWORD(wParam)) {
                                case SB_LINELEFT :
                                        // ...
```

```
                                        break;
                                case SB_LINERIGHT :
                                        // ...
                                        break;
                                case SB_ENDSCROLL :
                                        // ...
                                        break;
                                case SB_RIGHT :
                                        // ...
                                        break;
                                case SB_LEFT :
                                        break;
                                case SB_PAGERIGHT :
                                        // ...
                                        break;
                                case SB_PAGELEFT :
                                        // ...
                                        break;
                                case SB_THUMBPOSITION :
                                        SetScrollPos(hScroll, SB_CTL, HIWORD(wParam), TRUE);
                                        break;
                                case SB_THUMBTRACK :
                                        {
                                                RECT rect;
                                                HBRUSH hBrush =
                                                        CreateSolidBrush(RGB(HIWORD(wParam), 0, 0));
                                                HDC hDC = GetDC(hWnd);
                                                GetClientRect(hWnd, &rect);
                                                FillRect(hDC, &rect, hBrush);
                                                DeleteObject(hBrush);
                                                ReleaseDC(hWnd, hDC);
                                        }
                                break;
                        }
                        break;
                        case WM_LBUTTONDOWN :
                                pos += 5;
                                SetScrollPos(hScroll, SB_CTL, pos, TRUE);
                                break;
                        case WM_PAINT:
                                hdc = BeginPaint(hWnd, &ps);
                                Rectangle(hdc, 100, 100, 150, 150);
                                SetWindowOrgEx(hdc, 500, 500, NULL);

                                Rectangle(hdc, 500, 500, 600, 600);

                                EndPaint(hWnd, &ps);
                                break;
                        case WM_DESTROY:
                                PostQuitMessage(0);
                                break;
                        default:
                                return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
```

```
                case WM_INITDIALOG:
                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
  return FALSE;
}
```

Kontrol biçiminde kaydırma çubuğu yaratılırken kaydırma çubuğunun yatay yada düşey olacağı SBS_VERT ve SBS_HORZ biçimleriyle belirtilmelidir.

Anahtar Notlar:

WS_CLIPCHILDREN pencere biçimi, bir pencerenin alt pencerelerini clip alanından atmak için kullanılır. Böylelikle pencerenin tamamını boyadıkça alt pencereler bundan etkilenmez.

PENCERE ORİJİNİNİN DEĞİŞTİRİLMESİ

Aslında pencere pozitif ve negatif bölgelere sahip olan geniş bir alandır. BeginPaint fonksiyonu ile dc alındığında çalışma alanının sol üst köşesi default olarak (0, 0) koordinatındadır. Aslında negatif bölgelere çizim yapılabilir. Fakat çizilenler görülmeyecektir. İşte SetWindowOrgEx fonksiyon ile çalışma alanının sol üst köşesi herhangi bir koordinatta olacak şekilde öteleme yapılabilir.

```
BOOL SetWindowOrgEx(
 HDC hdc,           // handle of device context
 int X,             // new x-coordinate of window origin
 int Y,             // new y-coordinate of window origin
 LPPOINT lpPoint    // address of structure receiving original origin
);
```

Fonksiyonun 1. parametresi, orijini değiştirilecek pencerenin pencere DC'sinin handle değeri, 2. ve 3. parametreleri yeni orijin noktasının değerleridir. Son parametre, eski orijin noktasının yerleştirileceği Point yapısının adresidir. NULL geçilebilir.

Programcı, önce pencere orijinini değiştirip sonra çizim yapmalıdır. Başka bir deyişle orijinin değişmesinin etkisi, orijinin değiştirilmesinden sonra yapılan fonksiyonlar için geçerlidir. Yani eski çizimler görülmeye devam eder; yeni çizimler yeni orijine göre çizilir.

SCROLLVİEW UYGULAMASI

Herhangi bir görüntünün kaydırma çubukları kaydırmanın pratik bir yöntemi vardır. Bunun çin WM_PAINT mesajında pencereye sığmasa bile bütün çizimler yapılır. Fakat bu çizimlerden önce kaydırma çubuklarının o anki durumuna göre SetWindowOrgEx fonksiyonu çağırılır. Kaydırma çubuklarıyla oynandığında InvalidateRect işlemi yapılır. WM_PAINT mesajı oluşacaktır. WM_PAINT mesajında da SetWindowOrgEx, kaydırma çubuklarının durumuna göre işlemini yapar. Yani sonuç olarak her defasında çizimin hepsi yapılmaktadır ama yalnızca belirli bir bölümü görüntülenmektedir. bak: autoscroll.cpp

```
// autoscroll.cpp : Defines the entry point for the application.

#include "stdafx.h"
#include "resource.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_AUTOSCROLL, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_AUTOSCROLL);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);
```

```
        wcex.style          = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc    = (WNDPROC)WndProc;
        wcex.cbClsExtra     = 0;
        wcex.cbWndExtra     = 0;
        wcex.hInstance      = hInstance;
        wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_AUTOSCROLL);
        wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName   = (LPCSTR)IDC_AUTOSCROLL;
        wcex.lpszClassName  = szWindowClass;
        wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW|WS_VSCROLL,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        static int vertPos;

        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_CREATE:
                        SetScrollRange(hWnd, SB_VERT, 0, 15 * 500, TRUE);
                        break;
                case WM_VSCROLL:
                        {
                                switch(LOWORD(wParam)) {
                                        case SB_LINEUP :
                                                vertPos  -= 15;
                                                break;
                                        case SB_LINEDOWN :
```

```
                        vertPos  += 15;
                        break;
                case SB_ENDSCROLL :
                        // ...
                        break;
                case SB_RIGHT :
                        // ...
                        break;
                case SB_LEFT :
                        // ...
                        break;
                case SB_PAGEUP:
                        vertPos  -= 15 * 10;
                        break;
                        // ...
                        break;
                case SB_PAGEDOWN :
                        vertPos  += 15 * 10;
                        break;
                        // ...
                        break;
                case SB_THUMBPOSITION:
                        vertPos = HIWORD(wParam);
                        break;
                case SB_THUMBTRACK:
                        // ...
                        break;
                }
                SetScrollPos(hWnd, SB_VERT, vertPos, TRUE);
                InvalidateRect(hWnd, NULL, TRUE);
        }
        break;

case WM_COMMAND:
        wmId    = LOWORD(wParam);
        wmEvent = HIWORD(wParam);
        // Parse the menu selections:
        switch (wmId)
        {
                case IDM_ABOUT:
                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                (DLGPROC)About);
                  break;
                case IDM_EXIT:
                  DestroyWindow(hWnd);
                  break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        break;
case WM_PAINT:
        {
                int i;
                char text[30];

                hdc = BeginPaint(hWnd, &ps);

                SetMapMode(hdc, MM_LOENGLISH);
                SetWindowOrgEx(hdc, 0, 1000, NULL);

                //tWindowOrgEx(hdc, 0, vertPos, NULL);
                for (i = 0; i < 500; ++i) {
                        wsprintf(text, "%d", i);
```

```
                                        TextOut(hdc, 0, i * 15, text, strlen(text));
                                }


                                EndPaint(hWnd, &ps);
                        }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                        return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
   return FALSE;
}
```

MAPPING MODE KAVRAMI

Mapping mode, x y artırımlarının yönünü ve GDI sistemindeki bir birimin bir birimin ne anlama geleceğini belirten bir kavramdır. Default mapping mode, MM_TEXT modudur. Bu modda, bir birim bir pixel'dir. x artırımı sağa doğru, y artırımı ise aşağı doğru yapılmaktadır. MM_TEXT modu, bilgisayar için doğal bir mod olsa da bazı bakımlardan kullanışsız kalmaktadır. Örneğin kartezyen koordinat sistemi ile bir uyuşumsuzluk söz konusudur. Kullanılan mapping modlar şunlardır:

Mapping Mode	Birim	X artımı	Y artımı
MM_TEXT	1 pixel	Sağ	Aşağı
MM_LOMETRIC	0.1 mm	Sağ	Yukarı
MM_HIMETRIC	0.01 mm	Sağ	Yukarı
MM_LOENGLISH	0.01 in.	Sağ	Yukarı
MM_HIENGLISH	0.001in.	Sağ	Yukarı
MM_TWIPS	1/1440 in	Sağ	Yukarı
MM_ISOTROPIC	ayarlanabilir. fakat x = y	Ayarlanabilir	Ayarlanabilir
MM_ANISOTROPIC	ayarlanabilir. fakat x != y	Ayarlanabilir	Ayarlanabilir

Şüphesiz buradaki birimler monitörün büyüklüğüne göre değişebilir. Aslında o andaki monitördeki bir pixel'in boyutları elde edilerek monitörden bağımsız bir biçimde aynı uzunluklarda çizim yapmak mümkün olabilir. Buradaki birimler, sistem fontunu referans almaktadır. Fakat bu birimler monitörlerde tutmasa bile bugünkü modern yazıcılarda tutmaktadırlar. MM_ISOTROPIC ve MM_ANISOTROPIC modlarda bir

birimin ne olacağı programcı tarafından ayarlanabilmektedir. MM_ISOTROPIC modda, x ve y eksenindeki birimler aynı olmak zorundadır.

Zoom yapan programların aslında basit bir mantığı vardır. Çizimler, izotropik modlarda yapılabilir. Sonra bir birimin anlamı değiştirilerek büyütme ve küçültme sağlanabilir. Örneğin 10 birim 1 pixel'e karşı geliyorsa sonra biz bunu 5 birimin 1 pixel'e karşı geleceği şekle değiştirirsek tüm şekiller 2 kat büyüyecektir.

Mapping mode, SetMapMode fonksiyonu ile değiştirilebilir.

```
int SetMapMode(
 HDC hdc,            // handle of device context
 int fnMapMode       // new mapping mode
);
```

Fonksiyonun 1. parametresi dc'nin handle değeri, 2. parametresi değiştirilecek mapping modun değeridir. Bu parametreye MM_XXX biçiminde sembolik sabitler geçilebilir. Fonksiyon, önceki mapping mode değerine geri dönmektedir. Mapping mode değiştirildiğinde önceki çizimler bu işlemden etkilenmez; değiştirmeden sonraki çizimler bu işlemden etkilenir.

Mapping mode değiştirildikten sonra artık SetWindowOrgEx fonksiyonu bile yeni mapping moda göre çalışacaktır.

Fare mesajlarına geçirilen x, y değerleri her zaman pixel cinsindendir. Zaten mapping mode ve pencere orijini kavramı dc'ye ilişkin olduğuna göre başka bir olasılık da söz konusu olamaz. Fakat bazen fare mesajlarından elde edilen x, y değerlerinin o anki dc'deki mapping mode ve pencere orijinine göre mantıksal sisteme dönüştürülmesi gerekebilir. Bunun için DPtoLP ve LPtoDP fonksiyonları kullanılır.

```
BOOL DPtoLP(
  HDC hdc,                  // handle to device context
  LPPOINT lpPoints,         // pointer to array of points
  int nCount                // count of points
);
```

```
BOOL LPtoDP(
 HDC hdc,            // handle of device context
 LPPOINT lpPoints, // array of points
 int nCount          // count of points
);
```

DPtoLP, fiziksel koordinattan mantıksal koordinata, LPtoDP ise mantıksal koordinattan fiziksek koordinata dönüştürme yapar. Fonksiyonlar, dönüştürülecek n tane noktayı bir dizi içerisine alır; hepsini dönüştürerek aynı dizide üstüne yazar.

VIEW PORT KAVRAMI

View port kavramı ile yapılan şeylerin çoğu, pencere orijini değiştirilerek de yapılabilir. Fakat bu iki kavram aynı değildir. View port, (0, 0) orijinli ekrandaki herhangi bir pixel'e çekmek için kullanılır. SetWindowOrgEx fonksiyonundaki birimler, o andaki mapping moda bağlı olarak anlamlandırılır. Halbuki view port değiştiren fonksiyonların birimi her zaman pixel'dir.

Viewport, SetViewportOrgEx fonksiyonuyla değiştirilebilir.

```
BOOL SetViewportOrgEx(
 HDC hdc,            // handle of device context
 int X,              // new x-coordinate of viewport origin
 int Y,              // new y-coordinate of viewport origin
 LPPOINT lpPoint     // address of structure receiving original origin
);
```

Fonksiyonunun birinci parametresi DC'nin handle değeri, ikinci ve üçüncü parametreleri Viewport orijininin çekileceği pixeli belirtmektedir. Fonksiyonun son parametresi eski Viewport orijininin yerleştirileceği POINT yapısının adresidir, bu parametre NULL geçilebilir. bak graf

```
// graf.cpp : Defines the entry point for the application.

#include "stdafx.h"
#include "resource.h"
#include <math.h>

#define MAX_LOADSTRING 100

#define ORG_X                   200
#define ORG_Y                   200
#define SCALE_FACTOR            100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_GRAF, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_GRAF);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
```

```
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra         = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_GRAF);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_GRAF;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
```

```
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hDC;

        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
                                                (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_PAINT:
                        {
                                double x, y;

                                hDC = BeginPaint(hWnd, &ps);

                                SetViewportOrgEx(hDC, ORG_X, ORG_Y, NULL);

                                SetMapMode(hDC, MM_LOMETRIC);

                                MoveToEx(hDC, -800, 0, NULL);
                                LineTo(hDC, 800, 0);
                                MoveToEx(hDC, 0, -800, NULL);
                                LineTo(hDC, 0, 800);



                                for (x = -2 * 3.14; x < 2 * 3.14; x += 0.1)
                                        SetPixel(hDC, x * SCALE_FACTOR, sin(x) * SCALE_FACTOR,
                                                RGB(255, 0, 0));


                                EndPaint(hWnd, &ps);

                        }
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

// Mesage handler for about box.
```

```
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch (message)
	{
		case WM_INITDIALOG:
				return TRUE;

		case WM_COMMAND:
			if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
			{
				EndDialog(hDlg, LOWORD(wParam));
				return TRUE;
			}
			break;
	}
   return FALSE;
}
```

Combo Box Kontrolü

Combo Box kontrolü List Box ile Edit Box ya da List Box ile statik kontrolün bileşiminden oluşmaktadır. ComboBox sınıf ismiyle yaratılır. ComboBox kontrolünün özel pencere biçimleri CBS_XXX biçiminde yaratılır. Pencere biçimlerinin çoğu ListBox kontrolündekilere benzer. CBS_DROPDOWNLIST, ComboBox'ın statik kontrolden oluşacağı, CBS_DROPDOWN ise edit kontrolünden oluşacağı anlamına gelir. ComboBox üzerinde işlem yaptırmak için pencereye CB_XXX biçiminde mesajlar gönderilir. Listox kontrolündeki pek çok mesajın ComboBox karşılığı vardır. Örneğin, CB_ADDSTRING, ComboBox'a eleman eklemek için kullanılır.

ComboBox'ta seçilen eleman aktif eleman durumundadır. Yani önce kontrole CB_GETCURSEL mesajı gönderilerek seçili elemanın indeks'i elde edilir. Daha sonra CB_GETLBTEXT mesajıyla o indeksteki eleman alınır. bak combobox

```
#include <windows.h>
#include "resource.h"


BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
			int nCmdShow)
{
	return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL, DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static char *names[] ={"Ali", "Veli", "Selami", "Ayşe", "Fatma", NULL};
	static HWND hComboBox;

	switch (message) {
		case WM_INITDIALOG:
			{
				int i;

				hComboBox = GetDlgItem(hWnd, IDC_COMBO);

				for (i = 0; names[i] != NULL; ++i)
					SendMessage(hComboBox, CB_ADDSTRING, 0,
							(LPARAM) names[i]);
			}
```

```
                break;
        case WM_COMMAND:
                if (LOWORD(wParam) == IDOK && HIWORD(wParam) == BN_CLICKED) {
                        {
                                int index;
                                char buff[100];

                                index = SendMessage(hComboBox, CB_GETCURSEL, 0, 0);
                                SendMessage(hComboBox, CB_GETLBTEXT, index,
                                        (LPARAM) buff);

                                MessageBox(hWnd, buff, "message", MB_OK);

                                //EndDialog(hWnd, 0);
                        }
                        return TRUE;
                }
                if (LOWORD(wParam) == IDCANCEL && HIWORD(wParam) == BN_CLICKED) {
                        EndDialog(hWnd, 0);
                        return TRUE;
                }
                break;
        case WM_KEYDOWN:
                if (wParam== 'A') {
                        MessageBox(hWnd, "F1", "Message", MB_OK);
                        return TRUE;
                }
                break;
        }
        return FALSE;
}
```

UNICODE SİSTEMİ

UNICODE iki bytelık bir karakter kodlama standardıdır. UNICODE tablo içerisinde tüm uluslara ilişkin dillerdeki karakterler, özel semboller yerleştirilmiştir. UNICODE tablonun ilk 128 karakteri ASCII tablosunun tamamen aynısıdır. UNICODE standardı hangi 2 bytelık sayıya hangi karakterin karşı geleceğini belirtmektedir. Fakat UNICODE bir yazının bir dosya içerisinde nasıl bulunacağını belirlemek için ayrı standartlar kullanılmaktadır.

UNICODE bir yazının dosya içerisinde iki tipik saklanma biçimi vardır. UTF16, her karakterin iki byte olarak saklanacağını belirtir. UTF8 daha ekonomik bir formattır. Bu formatta ilk 128 karakter 1 byte ile ifade edilirken diğer karakterler 2 byte'la ifade edilebilmektedir.

UNICODE Sisteminin Tarihsel Gelişimi

Bilgisayar sistemlerinde uzun süre İngilizce karakterlerin yer aldığı ASCII tablosu kullanıldı. Bilgisayar sistemleri geliştikçe İngilizce dışındaki dillerinde ifade edilme gereksinimi arttı. Bunun için il alınan önlem Code Page denilen sayfalandırma sistemi oldu. Code Page sisteminde ASCII tablosunun ikinci yarısı her dil için değişik bir biçimde düzenlenmiştir. yani dillere özgü karakterler ASCII tablosunun ikinci yarısına alınmıştır. Fakat bazı dillerdeki karakter sayısı (örneğin Japonya) bu ikinci yarıya sığmayacak biçimde fazladır. bunun için multibyte denilen bir sistem önerilmiştir. Bu dillerde bazı karakterler ESCAPE karakteri olarak kullanılmaktadır. Bu karakterlerle karşılaşıldığında bu karakterlerden sonraki bazı karakterlerin tek bir karakteri ifade ettiği anlaşılır. Multibyte karakterler C'de tek tırnak içerisine birden fazla karakter yerleştirilerek ifade edilmektedir. Aslında C' de tek tırnak içerisine sizeof(int) kadar karakter yerleştirilebilir. Multibyte sisteminin en kötü tarafı yazıların içeriklerine bakılmadan uzunlukları gibi çıkarımların yapılamamsıdır. İşte program içerisinde Code Page kullanarak her karakteri farklı uzunlukla

ifade etmek yerine her karakterin iki byte ile ifade edildiği UNICODE sistemine geçilmiştir. UNICODE sisteminde program içerisinde her karakter iki byte ile ifade edilmektedir. Fakat dosya içerisinde UTF8 gibi formatlarla durum farklı olabilir.

C'de multibyte karakterler tek tırnak ile ifade edilirken UNICODE karakterler l öneki ile ifade edilmektedir. l öneki, tek tırnak yada çift tırnağa bitişik olmak zorundadır. Örneğin: L'a' L"ali" gibi...

C'de UNICODE karakterleri ifade etmek için wchar_t typeef ismi kullanılmaktadır. wchar_t genellikle short int olarak typedef edilmiş bir isimdir.

typedef short int wchar_t;

wchar_t türü C++'da bir anahtar sözcüktür. Özetle C'de UNICODE, wcahr_t türü ile temsil edilmektedir.

Windows Sistemlerinin UNICODE Uyumluluğu

Windows sistemleri bütünüyle UNICODE karakter sistemini desteklemektedir. Windows 95, 98 ve ME sistemlerinin çekirdeği ASCII tabanlı çalışmaktadır. Biz bu sistemlere UNICODE bir yazı verirsek önce bu yazı ASCII'ye dönüştürülmekte; sonra çekirdek tarafından işlenmektedir. Halbuki Windows NT, 2000 ve Xp sistemlerinin doğal çekirdek çalışması UNICODE üzerinedir. Biz bu sistemlere ASCII bir yazı verdiğimizde yazı önce UNICODE sistemine dönüştürülmekte, sonra işleme sokulmaktadır.

Windows 3.x sistemlerinin UNICODE desteği yoktur. Win32 sistemlerini hepsinde yazı parametresi alan fonksiyonlardan iki tane vardır. Bu fonksiyonların ASCII versiyonlarının sonu A ile, UNICODE versiyonlarının sonu W ile bitmektedir. Örneğin aslında CreateWindow diye bir fonksiyon yoktur. CreateWindowA ve CreateWindowW biçiminde fonksiyonlar vardır. Fakat yazı parametresi almayan fonksiyonlardan birer tane vardır.

windows.h içerisinde UNICODE isimli sembolik sabite bakılarak yazı parametresi alan fonksiyonlar duruma göre A sonekli yada W sonekli biçime dönüştürülmüştür.

```
#ifdef UNICODE
#define CreateWindow CreateWindow
#define GetWindowText GetWindowText
...
#else
#define CreateWindow CreateWindowA
#define GetWindowText GetWindowTextA
...
#endif
```

Görüldüğü gibi eğer biz programın tepesine #define UNICODE bildirimini yaparsak API fonksiyonlarının UNICODE versiyonlarını, aksi durumda ASCII versiyonlarını kullanırız.

Programcı programını öyle bir biçimde yazmalıdır ki programın tepesinde UNICODE sembolik sabitini tanımladığında her şey UNICODE sistemine dönsün. İşte bunun için char yada wchar_t türlerini doğrudan kullanmadan bunlar yerine TCHAR tür ismini kullanmalıdır. TCHAR, duruma göre char yada wchar_t anlamına gelmektedir.

```
#ifdef UNICODE
        typedef wchar_t TCHAR;
#else
```

```
        typedef char TCHAR;
#endif
```

Benzer biçimde PTSTR, LPTSTR, PCSTR, LPCTSTR türleri de UNICODE sembolik sabiti duyarlı türlerdir.

```
#ifdef UNICODE
        typedef wchar_t         *PTSTR, *LPTSTR;
        typedef const wchar_t *PCTSTR, *LPCSTR;
#else
        typedef char            *PTSTR, *LPTSTR;
        typedef const char      *PCTSTR, *LPCSTR;
```

Bu durumda biz UNICODE uyumlu bir gösterici tanımlayacaksak bu t'li türleri tercih etmeliyiz.

Şimdi tek bir problem kalmış durumdadır. İki tırnak içindeki yazıların duruma göre yani UNICODE sembolik sabitine bağlı olarak L önekli biçimde olması gerekir. Bunun için TEXT makrosu kullanılır. windows.h içerisinde şöyle bir TEXT makrosu vardır.

```
#ifdef UNICODE
#define TEXT(str)     LStr
#else
#define TEXT(str)     Str
#endif
```

O halde programcı tüm iki tırnak içerisindeki yazıları TEXT makrosu ile vermelidir. Böylece bu yazılar duruma göre değişebilecektir.

O halde UNICODE uyumlu bir programın oluşturulması aşaması şöyledir:

1. char yada wchar_t kullanılmaz. Bunun yerine TCHAR kullanılır.
2. char * yada wchar_t * kullanılmaz. Bunun yerine TCHAR * yada LPTSTR, LPCTSTR türleri kullanılır.
3. Stringler her zaman TEXT makrosuyla kullanılmalıdır.

Tüm bu yollar izlendiğinde artık UNICODE sisteme geçmek için tek yapılacak şey programın tepesine #define UNICODE bildirimi yerleştirmektir.

TCHAR.H Dosyası Ve Standart C Fonksiyonlarının UNICODE Biçimleri

tchar.h dosyası içerisinde TEXT makrosunun _T ve _TEXT biçimleri de vardır. Fakat bu makrolar UNICODE değil _UNICODE sembolik sabitine duyarlıdır.

Biz TEXT makrosu yerine _T makrosunu kullanmak istersek bunun için tchar.h dosyasını include etmeliyiz. Bu durumda UNICODE versiyona geçmemiz için UNICODE sembolik sabitinin yanı sıra _UNICODE sembolik sabitini de define etmeliyiz.

C90'da (yani klasik C'de) yazı parametresi alan standart C fonksiyonlarının UNICODE biçimleri ayrıca tanımlanmamıştır. Yani buradaki fonksiyonlar, yalnızca 1 byte'lık karakter tablolama sistemlerinde çalışacak biçimdedir ve bu fonksiyonların parametreleri char * biçimindedir.C99'da bu fonksiyonların UNICODE biçimleri tanımlanmıştır fakat c99'u destekleyen derleyiciler yok denecek kadar azdır. Microsoft Visual C derleyicilerinde yazı parametresi alan tüm standart C fonksiyonlarının "str" öneki yerine "wcs"

öneki alan unicode biçimleri de vardır. örneğin strchar fonksiyonunun unicode biçimi wcschar dir. ayrıca tcahr.h içerisinde birde bunların unicode uyumlu isimleri vardır. bu isimler _tcs öneki ile isimlendirilmiştir.

```
#ifdef _UNICODE
        #define _tcschr         wschr
        .....
#else
        #define _tcschr         strchr
        ...
#endif
```

main ve WinMain fonksiyonlarının da UNICODE biçimleri vardır. main fonksiyonunun UNICODE biçimi wmain; UNICODE uyumlu biçimi ide _tmain'dir. Benzer biçimde WinMain fonksiyonunun UNICODE biçimi wWinmain, UNICODE uyumlu biçimi ise _tWinMain'dir.

Özetle UNICODE uyumlu program yazmak demek neredeyse hiç değişik yapmadan programın UNICODE biçimini oluşturmak demektir. UNICODE uyumlu program yazmak için şunları yapmak gerekir.

1. windows.h dosyasının yanı sıra tchar.h dosyası da include edilmelidir.
2. API fonksiyonları normal isimlerle kullanılmalı, standart C fonksiyonlarının UNICODE uyumlu yani _tcs ile başlayan biçimleri kullanılmalıdır.
3. Tüm stringler TEXT, _T yada TEXT makrolarıyla yazılmalıdır (T ve _TEXT makroları tchar.h içerisinde _UNICODE sembolik sabitine duyarlı biçimde, TEXT makrosu ise windows.h içerisinde ve UNICODE sembolik sabitine duyarlı biçimde oluşturulmuştur).
4. karakter kavramı için tchar tür ismi, karakter türünden gösterici kavramı için tchar * yada LPTSTR, LPCTSTR türleri kullanılmalıdır.
5. Programın UNICODE biçimine geçmek için programın tepesine UNICODE ve _UNICODE sembolik sabitleri define edilmelidir.

Örneğin UNICODE uyumlu bir program:

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <stdlib.h>

int _tmain(void)
{
        _tprintf(TEXT("Test"));

        MessageBox(NULL, TEXT("Test"), TEXT("Message"), MB_OK);

        return 0;
}
```

unicode biçimine geçmek için:

```
#define UNICODE
#define _UNICODE

#include <windows.h>
```

```
#include <tchar.h>
#include <stdio.h>
#include <stdlib.h>

int _tmain(void)
{
	_tprintf(TEXT("Test"));

	MessageBox(NULL, TEXT("Test"), TEXT("Message"), MB_OK);

	return 0;
}
```

MODELESS DIALOG PENCERELERİ

Modal diyalog pencerelerinde diyalog penceresi açıldığında akış, diyalog penceresini açan Dialogbox fonksiyonu içerisinde kalır. DialogBox API fonksiyonu kendi içerisinde başka bir mesaj döngüsü oluşturmuş durumdadır. Akış, DialogBox API fonksiyonunda çıktığında kalınan yerden devam eder. Halbuki modeless diyalog pencerelerinde akış, modeless diyalog penceresini yaratan CreateDialog fonksiyonunda kalmaz. Akış, CreateDialog fonksiyonunu hemen sonlandırarak programcının gerçek mesaj döngüsünü işler. Böylece modeless diyalog pencerelerinde hem diyalog penceresi üzerinde işlemler hem de gerçek pencere üzerinde işlemler aynı anda yapılabilmektedir. Örneğin Find & Replace tipik bir modeless diyalog penceresi örneğidir.

Modeless Diyalog Pencerelerinin Yaratılması Ve Kapatılması

Modeless diyalog pencerelerinin yaratılması için CreateDialog API fonksiyonu kullanılır.

```
HWND CreateDialog(
 HINSTANCE hInstance,     // handle to application instance
 LPCTSTR lpTemplate,      // identifies dialog box template name
 HWND hWndParent,         // handle to owner window
 DLGPROC lpDialogFunc     // pointer to dialog box procedure
);
```

Fonksiyonun 1. parametresi PE formatının yüklenme adresidir. 2. parametre, diyalog kaynağının ismidir. 3. parametre, üst pencerenin handle değeri olarak girilmelidir. Son parametre ise diyalog pencere fonksiyonunun adresidir.

Modeless diyalog pencereleri, EndDialog fonksiyonu ile değil DestroyWindow fonksiyonu ile kapatılmalıdır.

```
// testmodeless.c
#include <windows.h>
#include <tchar.h>
#include "resource.h"

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);


HINSTANCE g_hInstance;
HWND g_hModelessDlg;
```

```
int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine,
                     int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;

        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU);
                wndClass.lpszClassName = TEXT("Generic");
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        g_hInstance = hInstance;

        hWnd = CreateWindow(TEXT("Generic"), TEXT("Generic App"),
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                if (g_hModelessDlg == 0 || !IsDialogMessage(g_hModelessDlg, &message)) {
                        TranslateMessage(&message);
                        DispatchMessage(&message);
                }
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message) {
                case WM_COMMAND:
                        if (LOWORD(wParam) == ID_FILE_EXIT)
                                DestroyWindow(hWnd);
                        else if (LOWORD(wParam) == ID_FILE_DIALOG)
                                g_hModelessDlg = CreateDialog(g_hInstance,
                                    MAKEINTRESOURCE(IDD_MODELESS_DIALOG), hWnd, DlgProc);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
```

```
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message) {
                case WM_INITDIALOG:
                        break;
                case WM_COMMAND:
                        if (HIWORD(wParam) == BN_CLICKED && LOWORD(wParam) == IDOK) {
                                DestroyWindow(hWnd);
                                g_hModelessDlg = NULL;
                                break;
                        }
                        return TRUE;
        }
        return FALSE;
}
```

```
//testmodeles1.c

#include <windows.h>
#include <tchar.h>
#include "resource.h"

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);


HINSTANCE g_hInstance;
HWND g_hModelessDlg;

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine,
                        int nCmdShow)
{
        WNDCLASS wndClass;
        HWND hWnd;
        MSG message;


        if (!hPrevInstance) {
                wndClass.style = CS_HREDRAW | CS_VREDRAW;
                wndClass.cbClsExtra = 0;
                wndClass.cbWndExtra = 0;
                wndClass.hInstance = hInstance;
                wndClass.hIcon = LoadIcon(NULL, IDI_WINLOGO);
                wndClass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
                wndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
                wndClass.lpszMenuName = MAKEINTRESOURCE(IDR_MENU);
                wndClass.lpszClassName = TEXT("Generic");
                wndClass.lpfnWndProc = (WNDPROC) WndProc;
                if (!RegisterClass(&wndClass))
                        return -1;
        }

        g_hInstance = hInstance;

        hWnd = CreateWindow(TEXT("Generic"), TEXT("Generic App"),
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
```

```
                0,
                CW_USEDEFAULT,
                0,
                NULL,
                NULL,
                hInstance,
                NULL);

        if (!hWnd)
                return -1;

        ShowWindow(hWnd, SW_RESTORE);
        UpdateWindow(hWnd);
        while (GetMessage(&message, 0, 0, 0)) {
                if (g_hModelessDlg == 0 || !IsDialogMessage(g_hModelessDlg, &message)) {
                        TranslateMessage(&message);
                        DispatchMessage(&message);
                }
        }
        return (message.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        static HWND hEdit;

        switch (message) {
                case WM_CREATE:
                        hEdit = CreateWindow(TEXT("edit"), TEXT(""), WS_CHILD|WS_VISIBLE, 0, 0, 0,
                                0, hWnd, (HMENU) 100, ((LPCREATESTRUCT) lParam)->hInstance, 0);
                        break;
                case WM_COMMAND:
                        if (LOWORD(wParam) == ID_FILE_EXIT)
                                DestroyWindow(hWnd);
                        else if (LOWORD(wParam) == ID_FILE_DIALOG)
                                g_hModelessDlg = CreateDialog(g_hInstance,
                                   MAKEINTRESOURCE(IDD_MODELESS_DIALOG), hWnd, DlgProc);
                        break;
                case WM_SIZE:
                        MoveWindow(hEdit, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                  return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message) {
                case WM_INITDIALOG:
                        break;
                case WM_COMMAND:
                        if (HIWORD(wParam) == BN_CLICKED && LOWORD(wParam) == IDOK) {
                                DestroyWindow(hWnd);
                                g_hModelessDlg = NULL;
                                break;
                        }
                        return TRUE;
        }
```

```
        return FALSE;
}
```

Modeless diyalog pencerelerinin işletilmesi için ana mesaj döngüsüne IsDialogMessage fonksiyonunun eklenmesi gerekir. Bu fonksiyon, gelen mesajın modeless diyalog penceresine ilişkin olup olmadığını tespit eder; eğer mesaj modeless diyalog penceresine ilişkin ise modeless diyalog pencere fonksiyonunu çağırarak mesajı işler. Yani, modeless diyalog pencerelerinin mesajları, DispatchMessage fonksiyonu ile değil IsDialogMessage fonksiyonu ile işlenmelidir.

```
BOOL IsDialogMessage(
 HWND hDlg,          // handle of dialog box
 LPMSG lpMsg         // address of structure with message
);
```

Fonksiyonun 1. parametresi modeless diyalog penceresinin handle değeri; 2. parametresi kuyruktan alınan mesajın parametresidir. Fonksiyon, eğer mesaj modeless diyalog penceresine ilişkin ise 0 dışı bir değere, değilse 0 değerine geri döner. Bu durumda modeless diyalog penceresi içeren tipik bir uygulamanın mesaj döngüsü şöyle olmalıdır:

```
while (GetMessage(&message, NULL, 0, 0)) {
        if (g_hModelessDlg == 0 || !IsDialogMessage(g_hModelessDlg, &message)) {
                TranslateMessage(&message);
                DispatchMessage(&message);
        }
}
```

Anahtar Notlar:

DialogBox fonksyionu, diyalog penceresini otomatik olarak visible hale getirmektedir. Halbuki CreateDialog fonksiyonu otomatik olarak visible hale getirmez, programcının diyalog kaynağında WS_VISIBLE pencere biçimini kullanması gerekir.
